



MIGRATING A PHP 5 APP TO PHP 7

By Prosper Otemuyiwa



Migrating a PHP 5 App to PHP 7

Prosper Otemuyiwa, Auth0 Inc.

Version 1.0.0, 2017

Abstract

Learn how to migrate a PHP 5 application to PHP 7. Setup, Tools, Development Environment and Implementation.

Contents

1	Introduction	4
1.1	PHP 5 and PHP 7	4
2	Upgrading Your Development Environment to PHP 7	6
2.1	Mac OS X	6
2.2	Windows	7
2.3	Ubuntu	7
2.4	Debian	8
2.5	CentOS / Red Hat Enterprise Linux	8
2.6	phpbrew	9
2.7	Vagrant	11
2.7.1	Laravel Homestead	11
2.7.2	php7dev	12
2.8	Valet	12
2.9	Docker	15
2.9.1	php7-dockerized	15
2.9.2	Laradock	15
2.9.3	phpdocker	16
3	Elementary Language Changes	17
3.1	Spaceship Operator	17
3.2	Array Constants	19
3.3	Null Coalescing Operator	19
3.4	Integer Division	20
3.5	Regular Expressions	20
3.6	Filtered unserialize()	21
3.6.1	Cryptographically Secure Pseudorandom Number Generator (CSRPNG)	22
3.7	session_start config enhancements	22
3.8	Unpack objects with list()	22
3.9	dirname() enhancement	23
3.10	Reflection API Enhancements	23
3.11	Reserved Words	24

4	Scalar Typehinting & Return Type Declarations	26
4.1	Typehinting	26
4.2	Return Types	27
4.3	Strong Type Check	28
5	Error Handling, Expectations and Assertions	29
5.1	Expectations and Assertions	30
6	Closures and Generators	32
6.1	Generator Return Expressions	33
6.2	Generator Delegation	33
7	Object-Oriented Programming Enhancement	35
7.1	Anonymous Classes	35
7.2	Group Use Declarations	36
8	Better Unicode Support	37
8.1	IntlChar	38
9	Deprecated & Removed Features	39
9.1	Removed Extensions and Server APIs	40
9.2	Backward Incompatible Changes	41
10	Uniform Variable Syntax and Static Values	43
10.1	Accessing Static Values	43
11	Migration Tools	45
11.1	PHP 7 MAR	45
11.2	PHP 7 Compatibility Checker	46
11.3	Phan	46
11.4	phpto7aid	46
11.5	PhpStorm PHP 7 Compatibility Inspection	46
12	Practical Migration of Two Apps	48
12.1	Building a PHP5 App	48
12.1.1	Create and Configure Auth0 Client	48
12.1.2	Build the App	53
12.1.3	Run The App	62
12.1.4	Migrate to PHP 7	64
12.2	API	64
12.2.1	Use PHP 7 Features	66
13	Introducing PHP 7.1 Features	71
13.1	Nullable Types	71
13.2	Void Type	72
13.3	Symmetric Array Destructuring	72
13.4	Class Constant Visibility	73

13.5 Multi-Catch Exception Handling	73
13.6 Iterables	74
13.7 Keys Support in list()	74
13.8 Negative String Offsets Support	75
13.9 Conversion of Callables to Closures	75
13.10Asynchronous Signal Handling	76
13.11Support for HTTP/2 Server Push	76
13.12Better Error Retrieval	76
13.13Throw Error on Passing too few Function Arguments	77
14 Performance Evaluation	78
15 Conclusion	85

Chapter 1

Introduction

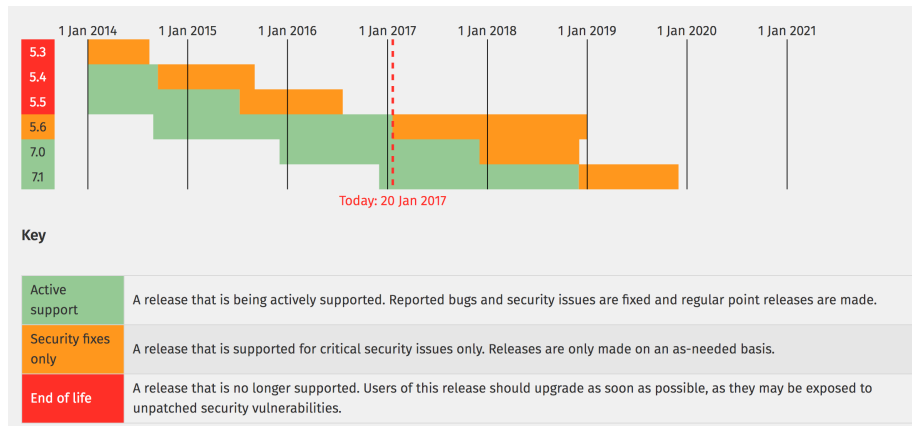
Many PHP applications are still running on PHP 5.x, not ready to take full advantage of the awesome features that PHP 7 offers. A lot of developers have not made the switch because of certain fears of compatibility issues, migration challenges and the strange awkward feeling that migrating their apps will take away a big chunk of their time.

1.1 PHP 5 and PHP 7

PHP 5 has been around for a very long time, over 10 years now. In fact, many production PHP apps are currently running on either PHP 5.2, 5.3 or 5.6. PHP 5 brought a lot of awesome features to PHP such as:

- Robust Support for Object oriented programming.
- Standard PHP Library (SPL)
- Closures.
- Namespaces.
- Magical methods for metaprogramming.
- MySQLi - improved MySQL extension.
- Cleaner Error handling.
- Better support for XML extensions.

Unfortunately, every thing that has a beginning must have an end. PHP 5.6 active support ended January 19, 2017 and it will only receive security support until December 31, 2018.



PHP 5 and 7 release and support duration

PHP 7.0 was officially released on December 3, 2015 with a lot of new features and better performance benefits. It is twice as fast as PHP 5. A summary of the new features are highlighted below:

- Return and Scalar type declarations
- Better Unicode support
- Null Coalescing Operator
- Fatal errors conversion to Exceptions
- Generator Enhancement
- Anonymous Classes
- Secure random number generator
- Removal of deprecated features

and much more! If you aren't using any of the deprecated features in your PHP 5 app, then the transition to PHP 7 will be seamless.

Chapter 2

Upgrading Your Development Environment to PHP 7

The first step to upgrading your application to use PHP 7 features is to migrate your development environment from PHP 5.x to PHP 7.x. We will cover how to upgrade your development environment to run PHP 7.x on Ubuntu, CentOS, Windows and Mac OS machines.

2.1 Mac OS X

If you are a fan of Homebrew¹, you can install PHP 7.0 via homebrew like so:

```
brew tap homebrew/dupes
brew tap homebrew/versions
brew tap homebrew/homebrew-php
brew unlink php56
brew install php70
```

Note: If you were using PHP 5.6, then you should unlink the old PHP by running `brew unlink php56` else unlink whatever version is present before you go ahead to install PHP 7.0.

Another option is to install it via `curl` on your terminal like so:

¹<http://brew.sh>

```
curl -s https://php-osx.liip.ch/install.sh | bash -s 7.0
```

2.2 Windows

If you are fan of WAMP² or XAMPP³, then you can just download the latest versions of the software. It comes packaged with PHP 7.0.

Download

XAMPP is an easy to install Apache distribution containing MariaDB, PHP, and Perl. Just download and start the installer. It's that easy.

Version	Checksum	Size
5.5.38 / PHP 5.5.38	md5 sha1	106 Mb
5.6.28 / PHP 5.6.28	md5 sha1	109 Mb
7.0.13 / PHP 7.0.13	md5 sha1	119 Mb

[Interested in XAMPP Docker Container?](#)

Download and install the last/latest version

Another option is to download the PHP 7.0 distribution for windows from <http://windows.php.net/download#php-7.0>.

2.3 Ubuntu

If you are running Ubuntu on your machine, especially around v14 and 15, you can install PHP 7.0 by running these commands:

²<http://www.wampserver.com/en>

³<https://www.apachefriends.org/download.html>

```
sudo apt-get update
sudo add-apt-repository ppa:ondrej/php
```

```
sudo apt-get install -y php7.0-fpm php7.0-cli php7.0-curl php7.0-gd php7.0-intl php7.0-mysq
```

Note: You can check out how to install PHP 7 and Nginx here⁴ and manually build memcached module for PHP 7.

2.4 Debian

If you are running Debian on your machine, especially around v6, v7 and v8, you can install PHP 7.0 by doing the following:

- Open up your `/etc/apt/sources.list` file, and make sure you have these commands below:

If you are using a Jessie distribution

```
deb http://packages.dotdeb.org jessie all
deb-src http://packages.dotdeb.org jessie all
```

If you are using a Wheezy distribution

```
deb http://packages.dotdeb.org wheezy all
deb-src http://packages.dotdeb.org wheezy all
```

- Fetch and Install the GnuPG key

```
wget https://www.dotdeb.org/dotdeb.gpg
sudo apt-key add dotdeb.gpg
```

- Install PHP 7.0

```
sudo apt-get update
sudo apt-get install php7.0
```

2.5 CentOS / Red Hat Enterprise Linux

If you are running CentOS or Red Hat Enterprise Linux operating system on your machine, you can install PHP 7.0 by running the following commands on your terminal like so:

⁴<https://serversforhackers.com/video/installing-php-7-with-memcached>

```
sudo yum update
rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
rpm -Uvh https://mirror.webtatic.com/yum/el7/webtatic-release.rpm
sudo yum install php70w
sudo yum install php70w-mysql
```

When you are done, run this command `php -v`, you should see something like this:

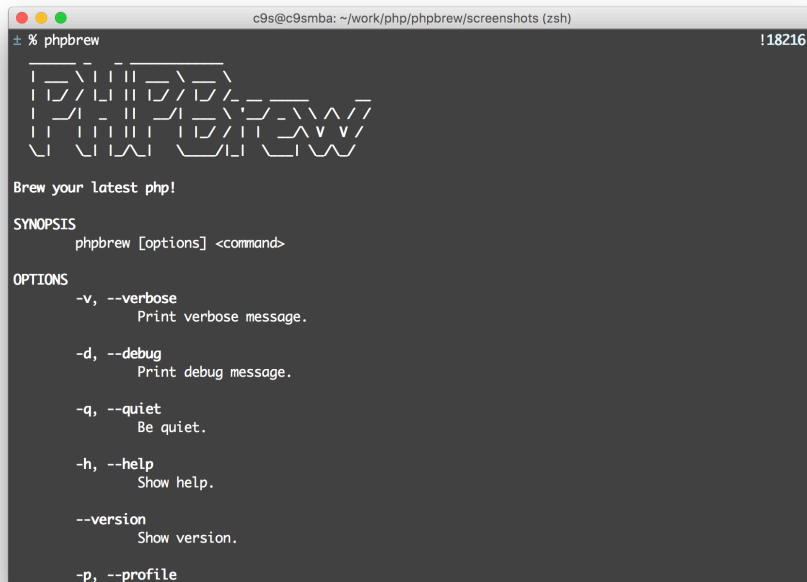
```
PHP 7.0.0 (cli) (built: Dec 2 2015 20:42:32) ( NTS )
Copyright (c) 1997-2015 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2015 Zend Technologies
```

2.6 phpbrew

PHPBrew⁵ is a tool that you can use to build and install multiple versions of PHP on your machine. It can:

- Build PHP with different variants like PDO, MySQL, SQLite, debug etc
- Compile Apache PHP module and separate them by different versions.
- Switch versions very easily and is integrated with bash/zsh shell.
- Install & enable PHP extensions into current environment with ease.
- Install multiple PHP into system-wide environment.
- Detect path for Homebrew and MacPorts.

⁵<https://github.com/phpbrew/phpbrew>

A terminal window with a dark background and light text. The title bar shows 'c9s@c9smba: ~/work/php/phpbrew/screenshots (zsh)'. The prompt is '% phpbrew'. The output shows a ASCII art logo for 'phpbrew', followed by the text 'Brew your latest php!'. Below that is the 'SYNOPSIS' section: 'phpbrew [options] <command>'. The 'OPTIONS' section lists: '-v, --verbose Print verbose message.', '-d, --debug Print debug message.', '-q, --quiet Be quiet.', '-h, --help Show help.', '--version Show version.', and '-p, --profile'.

phpbrew

You can install it on your machine like so:

```
curl -L -O https://github.com/phpbrew/phpbrew/raw/master/phpbrew  
chmod +x phpbrew
```

Then you can install it into your bin folder like so:

```
sudo mv phpbrew /usr/local/bin/phpbrew
```

Note: Make sure you have /usr/local/bin in your \$PATH environment variable.

You can install PHP 7 by running the following commands:

```
phpbrew self-update  
phpbrew install next as php-7.1.0  
phpbrew use php-7.1.0
```

You can use phpbrew to install PHP 7.0 from GitHub like so:

```
phpbrew install github:php/php-src@PHP-7.0 as php-7.0.0
```

Most times, we use PHP with other extensions such as MySQL, PDO, OpenSSL etc. You can use **phpbrew** to build your PHP environment with various variants like so:

```
phpbrew install 7.0.0 +mysql+mcrypt+openssl+debug+sqlite
```

This command above will build PHP with MySQL, mcrypt, OpenSSL, debug and SQLite.

2.7 Vagrant

Vagrant provides a simple, elegant way to manage and provision Virtual Machines. The development environments that run on Vagrant are packaged via **Vagrant boxes**. Vagrant boxes are completely disposable. If something goes wrong, you can destroy and re-create the box in minutes! One of such boxes I recommend is **Laravel Homestead**.

Note: You can check out these awesome free courses on learning how to use Vagrant⁶ on <https://serversforhackers.com>

2.7.1 Laravel Homestead

Laravel Homestead is an official, pre-packaged Vagrant box that provides you a wonderful development environment without requiring you to install PHP, a web server, and any other server software on your local machine. Homestead runs on any Windows, Mac, or Linux system. It includes the following:

- Ubuntu 16.04
- Git
- PHP 7.1 (Latest version of PHP)
- Nginx
- MySQL
- MariaDB
- Sqlite3
- Postgres
- Composer
- Node (With Yarn, PM2, Bower, Grunt, and Gulp)
- Redis
- Memcached
- Beanstalkd

Here are the steps to get started with Laravel Homestead:

⁶<https://serversforhackers.com/series/vagrant>

1. Install VirtualBox 5.1⁷, or VMWare⁸, and Vagrant⁹.
2. Now that you have Vagrant and VirtualBox or VMware installed, go ahead and download the Laravel Homestead box like so:

```
vagrant box add laravel/homestead
```

You can follow the instructions on the Laravel Homestead documentation¹⁰ to find out more about the installation process.

I recommend Windows users to take a stab at using Laragon¹¹. It provides an alternative but suitable and powerful environment like Laravel Homestead.

2.7.2 php7dev

Another Vagrant image is **php7dev**¹² by Rasmus Lerdorf (Creator of PHP). It is a Debian 8 Vagrant image which is preconfigured for testing PHP apps and developing extensions across many versions of PHP. You can gloriously switch between PHP versions by using the **newphp** command.

Follow the instructions on the README¹³ to find out how to install, configure and use it.

2.8 Valet

Valet¹⁴ is a PHP development environment for Mac minimalists. It was built by Taylor¹⁵ and Adam Wathan¹⁶ of the Laravel community. It is a fast blazing development environment that uses roughly 7MB of RAM. It requires Homebrew.

Laravel Valet configures Mac to use PHP's built-in web server in the background when your machine starts. With Valet, if you create a project folder called **auth0-php**, then you can just open **auth0-php.dev** in your browser and it will serve the contents of the folder automatically.

⁷<https://www.virtualbox.org/wiki/Downloads>

⁸<https://www.vmware.com>

⁹<https://www.vagrantup.com/downloads.html>

¹⁰<https://laravel.com/docs/5.3/homestead>

¹¹<https://laragon.org>

¹²<https://github.com/rlerdorf/php7dev>

¹³<https://github.com/rlerdorf/php7dev>

¹⁴<https://github.com/laravel/valet>

¹⁵<https://twitter.com/taylorotwell>

¹⁶<https://twitter.com/adamwathan>

You can share whatever you are working on locally with someone in another part of the world by just running this command:

```
valet share
```

```
Tunnel Status      online
Version            2.0.25/2.0.25
Region             United States (us)
Web Interface      http://127.0.0.1:4040
Forwarding         http://af79e2d6.ngrok.io -> todoapp.dev:80
Forwarding         https://af79e2d6.ngrok.io -> todoapp.dev:80

Connections        ttl    opn    rt1    rt5    p50    p90
                   2      0      0.02  0.01  0.23  0.46

HTTP Requests
-----
GET /favicon.ico   200 OK
GET /              200 OK
```

Valet uses Ngrok under the hood to share

You can even serve a local site over encrypted TLS using HTTP/2 by invoking a command like so:

```
valet secure blog
```

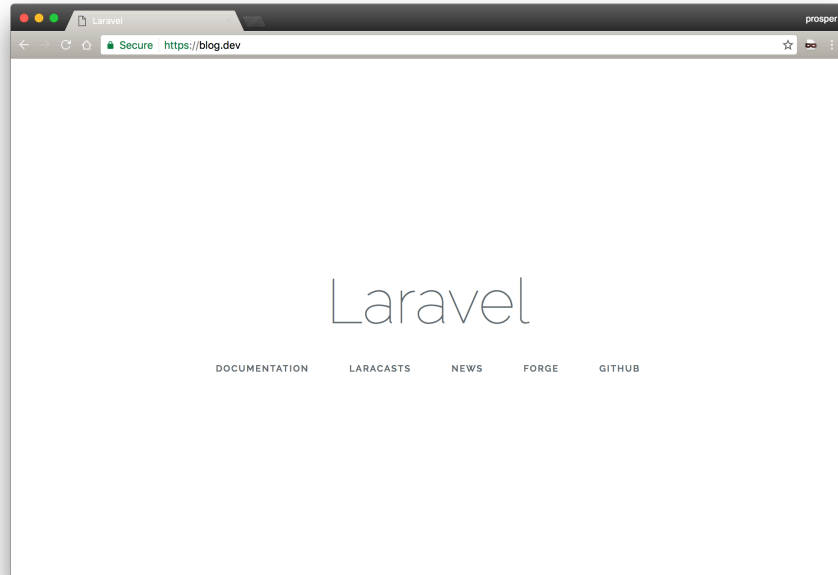
where `blog` is the name of the site or project folder. Valet generates a fresh local TLS certificate everytime you run the command to secure the site.

```
Tunnel Status      online
Version            2.0.25/2.0.25
Region             United States (us)
Web Interface      http://127.0.0.1:4040
Forwarding         http://af79e2d6.ngrok.io -> todoapp.dev:80
Forwarding         https://af79e2d6.ngrok.io -> todoapp.dev:80

Connections        ttl    opn    rt1    rt5    p50    p90
                   2      0      0.02  0.01  0.23  0.46

HTTP Requests
-----
GET /favicon.ico   200 OK
GET /              200 OK
```

Invoke the secure command



Site is served over https locally

Woot! Woot!, So awesome.

Out of the box, Valet supports Laravel¹⁷, Lumen¹⁸, Symfony¹⁹, Zend²⁰, CakePHP²¹, Wordpress²², Bedrock²³, Craft²⁴, Statamic²⁵ and Jigsaw²⁶. However, you can extend Valet with your own custom drivers²⁷.

Follow the instructions on the laravel valet documentation²⁸ to find out how to install and get started with it.

¹⁷<https://laravel.com>

¹⁸<https://lumen.laravel.com>

¹⁹<https://symfony.com>

²⁰<https://framework.zend.com>

²¹<https://cakephp.org>

²²<https://wordpress.org>

²³<https://roots.io/bedrock>

²⁴<https://craftcms.com>

²⁵<https://statamic.com>

²⁶<http://jigsaw.tighten.co>

²⁷<https://laravel.com/docs/5.3/valet#custom-valet-drivers>

²⁸<https://laravel.com/docs/5.3/valet>

2.9 Docker

Docker is an open-source engine that automates the deployment of any application as a lightweight, portable, self-sufficient container that will run virtually anywhere.

Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries and anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

2.9.1 php7-dockerized

php7-dockerized²⁹ is a simple PHP 7 Docker and Compose environment that is bundled with Nginx and MySQL.

Follow the instructions on setting up a local PHP 7 development environment with docker and compose!³⁰.

2.9.2 Laradock

Laradock³¹ is a docker PHP development environment that gives you a wonderful development environment without requiring you to install PHP 7, Nginx, MySQL, Redis, and any other software on your machines.

You can get started by doing the following:

- Clone Laradock inside your project like so:

```
git clone https://github.com/Laradock/laradock.git
```

- Enter the laradock folder and run this command:

```
docker-compose up -d nginx mysql redis beanstalkd
```

- Open your `.env` file and set the following:

```
DB_HOST=mysql  
REDIS_HOST=redis  
QUEUE_HOST=beanstalkd
```

²⁹<https://github.com/hamptonpaulk/php7-dockerized>

³⁰<https://medium.com/code-school/setting-up-a-local-php7-development-environment-with-docker-compose-e9531baed291#.bezir0x7n>

³¹<https://github.com/laradock/laradock>

You can follow the instructions on the laradock documentation³² to find out how to install and configure it.

2.9.3 phpdocker

phpdocker.io³³ is a PHP and Docker generated environment. It supports PHP 7 up until 7.1 beta. Follow the instructions to set it up like so:

- Clone <https://github.com/phpdocker-io/phpdocker.io>
- Copy `app/config/parameters.yml.dist` into `app/config/parameters.yml`
- Run `composer install`
- Run `bower install`
- Run `php bin/console assets:install --symlink --relative`
- Run `docker-compose up -d`

Don't hesitate to submit an issue on the `phpdocker-io` repo if you hit a road-block.

Note: Chris Fidao³⁴ has a fantastic course on Docker. With his course on shippingdocker.com³⁵, you'll learn how to use Docker in *development*, *testing* and *production*.

There are different ways of setting up a PHP 7 development environment. The few I have mentioned here should give you a lot of options in getting your machine ready to effectively test PHP 7 features.

³²<https://github.com/laradock/laradock/blob/master/README.md>

³³<https://github.com/phpdocker-io/phpdocker.io>

³⁴<https://twitter.com/fideloper>

³⁵<https://shippingdocker.com>

Chapter 3

Elementary Language Changes

3.1 Spaceship Operator

PHP 7 ships with a new operator, `<=>`, for simplifying the evaluation of arithmetic operations. With this operator, it is easier to evaluate less than, equal to, or greater than. The results will either be -1, 0 or 1. Ruby and PERL programmers are familiar with this operator.

This is how it works. If we have two operands `$x` and `y`, *andwedo*, `**x <=> $y**`, then

- if `$x` is less than `$y`, the result will be -1
- if `$x` equals `$y`, the result will be 0
- if `$x` is greater than `$y`, the result will be 1

```
function evaluate($x, $y) {  
    return $x <=> y;  
}
```

```
evaluate(9, 8);
```

```
// Result  
1
```

A Good real world case for this operator is in the simplification of comparison methods and using it for switch operations like so:


```

$data = [
    ['name' => 'Ado', 'cars' => 2],
    ['name' => 'Tony', 'cars' => 4],
    ['name' => 'Ramirond', 'cars' => 3],
    ['name' => 'Woloski', 'cars' => 12]
];

function sortByCars($x, $y) {
    return $x['cars'] <=> $y['cars'];
}

usort($data, 'sortByCars');

print_r($data);

// Result
Array
(
    [0] => Array
        (
            [name] => Ado
            [cars] => 2
        )

    [1] => Array
        (
            [name] => Ramirond
            [cars] => 3
        )

    [2] => Array
        (
            [name] => Tony
            [cars] => 4
        )

    [3] => Array
        (
            [name] => Woloski
            [cars] => 12
        )
)

```

It sorted the array easily with less code. Without the spaceship operator, I would have to write the `sortByCars` method like so:

```
function sortByCars($x, $y)
{
    if ($x['cars'] == $y['cars']) {
        return 0;
    }

    return ($x['cars'] < $y['cars']) ? -1 : 1;
}
```

3.2 Array Constants

Before now, constants defined with the `define()` method can only accept scalar values. In PHP 7, you can have constant arrays using the `define()` method like so:

```
// PHP 7
define('CARS', [
    'fine' => 'Mercedes',
    'strong' => 'Volkswagen',
    'ugly' => 'chevrolet'
]);

echo CARS['fine'];

// Result
Mercedes
```

3.3 Null Coalescing Operator

The purpose of this new operator, `??`, is to allow developers to set values from user inputs without having to check if the value has been set. Before PHP 7, this is how you evaluate input. Check this out:

```
$occupation = isset($_GET['occupation']) ? $_GET['occupation'] : 'bricklayer';
```

If the value of `$_GET['occupation']` exists, it returns the value else it assigns `bricklayer` to the `$occupation` variable. In PHP 7, you can simply shorten that line of code using the `??` operator like so:

```
// PHP 7
```

```
$occupation = $_GET['occupation'] ?? 'bricklayer';
```

It automatically checks whether the value is set and assigns the value to `$occupation` variable if it is, else it returns `bricklayer`.

The Null coalescing operator also allows you to chain expressions like so:

```
// PHP 7
```

```
$_ENV['occupation'] = 'software engineer';
```

```
$occupation = $_GET['occupation'] ?? $_ENV['occupation'] ?? 'bricklayer';
```

```
// Result
```

```
software engineer
```

This will assign the first defined value to the `$occupation` variable.

3.4 Integer Division

PHP 7 introduced a new function `intdiv()` which returns the result of an integer division operation as int.

```
// PHP 7
```

```
$result = intdiv(10, 4);
```

```
// Result:
```

```
2
```

3.5 Regular Expressions

Handling regular expressions just got easier in PHP 7. A new `preg_replace_callback_array()` function has been added to perform a regular expression search and replace using callbacks.

```
$message = 'Haaaalaaaaaa, Girls and people of Instagrants';
```

```
preg_replace_callback_array(  
    [  
        '~[a]+~i' => function ($match) {  
            echo strlen($match[0]), ' matches for "a" have been found!';  
        }  
    ],
```

```

        '~[b]+~i' => function ($match) {
            echo strlen($match[0]), ' matches for "b" found';
        },
        '~[p]+~i' => function ($match) {
            echo strlen($match[0]), ' matches for "p" found';
        }
    ],
    $message
);

```

```

// Result
4 matches for "a" have been found
6 matches for "a" have been found
1 matches for "a" have been found
1 matches for "a" have been found
1 matches for "a" have been found
1 matches for "p" found
1 matches for "p" found

```

3.6 Filtered unserialize()

The `unserialize()` function has been existing since PHP 4. It allows you to take a single serialized variable and convert back into a PHP value.

In PHP 7, the **options** parameter has been added. You can now whitelist classes that can be unserialized like so:

```

// converts all objects into __PHP_Incomplete_Class object
unserialize($obj, ["allowed_classes" => false]);

```

```

// converts all objects into __PHP_Incomplete_Class object except those of FirstClass and S
unserialize($obj, ["allowed_classes" => ["FirstClass", "SecondClass"]]);

```

```

// default behaviour (same as omitting the second argument) that accepts all classes
unserialize($obj, ["allowed_classes" => true]);

```

It was introduced to enhance security when unserializing objects on untrusted data.

Note: In PHP 7.1, the `allowed_classes` element of the **options** parameter is now strictly typed. `unserialize()` returns false if anything other than an array or boolean is given.

3.6.1 Cryptographically Secure Pseudorandom Number Generator (CSRPNG)

`random_bytes()` and `random_int()` have been added to the CSRPNG functions in PHP 7.

- `random_bytes()` returns a random string of a given length
- `random_int()` returns a random integer from a range

```
// return a random string of given length  
echo random_bytes(12);
```

```
// Result:  
3 .C5 4V
```

```
// return a random integer within this range  
echo random_int(0, 5000);
```

```
// Result:  
4497
```

Note: The results of `random_bytes` and `random_int` will be different for you because they are randomly generated. The results in the code above were gotten at the time I ran both functions.

3.7 session_start config enhancements

The `session_start()` method now accepts an array of values that can override the session config in `php.ini` file.

`session.lazy_write` which is on by default can be turned off by explicitly stating it in the `session_start()` method like so:

```
session_start([  
    'lazy_write' => false,  
    'cache_limiter' => 'private'  
]);
```

3.8 Unpack objects with list()

The `list()` language construct now allows you to unpack objects implementing the `ArrayAccess` interface.

```

$fruits = new ArrayObject(['banana', 'mango', 'apple']);

list($a, $b, $c) = $fruits;

echo $a. PHP_EOL;
echo $b. PHP_EOL;
echo $c. PHP_EOL;

// Result:
banana
mango
apple

```

Note: In **PHP 7.0.0** `list()` expressions can no longer be completely empty. In **PHP 5**, `list()` assigns the values starting with the right-most parameter. In **PHP 7**, `list()` starts with the left-most parameter. This is true when working with arrays with indices.

3.9 `dirname()` enhancement

The `dirname()` in PHP 5 returns a parent directory's path. In PHP 7.0.0, an optional *levels* parameter has been added to the function to allow you as a developer determine how many levels up you want to go when getting a path.

```

$path = '/Unicodeveloper/source/php-workspace/laravel/vavoom';

// Go three levels up and return the path
dirname($path, 3);

// Result:
/Unicodeveloper/source

```

3.10 Reflection API Enhancements

PHP 7 introduces two new reflection classes. One is the `ReflectionGenerator` class that reports information about generators and the other is the `ReflectionType` class that reports information about a function's return type.

ReflectionType API

- `ReflectionType::allowsNull` — Checks if null is allowed

- `ReflectionType::isBuiltin` — Checks if it is a built-in type
- `ReflectionType::__toString` - gets the parameter type name

ReflectionGenerator API

- `ReflectionGenerator::__construct` — Constructs a `ReflectionGenerator` object
- `ReflectionGenerator::getExecutingFile` — Gets the file name of the currently executing generator
- `ReflectionGenerator::getExecutingGenerator` — Gets the executing `Generator` object
- `ReflectionGenerator::getExecutingLine` — Gets the currently executing line of the generator
- `ReflectionGenerator::getFunction` — Gets the function name of the generator
- `ReflectionGenerator::getThis` — Gets the `$this` value of the generator
- `ReflectionGenerator::getTrace` — Gets the trace of the executing generator

Two new methods have also been added to the `ReflectionParameter` and `ReflectionFunctionAbstract` classes.

ReflectionParameter API

- `ReflectionParameter::hasType` - Checks if parameter has a type
- `ReflectionParameter::getType` - Gets a parameter's type

ReflectionFunctionAbstract API

- `ReflectionFunctionAbstract::hasReturnType` - Checks if the function has a specified return type.
- `ReflectionFunctionAbstract::getReturnType` — Gets the specified return type of a function

3.11 Reserved Words

PHP 7 now allows globally reserved words such as `new`, `private`, `for` as property, constant, and method names within classes, interfaces, and traits.

```
class Car {
    private $type, $who, $costs;

    public function new($carType) {
        $this->type = $carType;
        return $this;
    }
}
```

```
public function for($who) {
    $this->who = $who;
    return $this;
}

public function costs($price) {
    $this->price = $price;
    return $this;
}

public function __toString() {
    return $this->type . ' ' . $this->who . ' ' . $this->price. PHP_EOL;
}
}

$car = new Car();
echo $car->new('Mercedes Benz')->for('Wife')->costs(14000);

// Result:
Mercedes Benz Wife 14000
```


Chapter 4

Scalar Typehinting & Return Type Declarations

4.1 Typehinting

With PHP 5, you could typehint a function parameter with Classes, Interfaces, callable and array types only. For example, if you want a parameter of a certain type `string` to be passed into a function, you would have to do a check within the function like so:

```
// php 5
function getBookNo($number) {
    if (! is_integer($number)) {
        throw new Exception("Please ensure the value is a number");
    }

    return $number;
}
```

```
getBookNo('books');
```

PHP 7 eliminates the need for the extra check. With PHP 7, you can now typehint your function parameters with `string`, `int`, `float`, and `bool`.

```
// PHP 7
function getBookNo(int $number) {
    return $number;
}
```

```

getBookNo('books');

// Error raised
PHP Fatal error:  Uncaught TypeError: Argument 1 passed to getBookNo() must be..

// Continuation of the error message
..of the type integer, string given, called in ....

PHP 7 will throw a Fatal error as seen above once you typehint with scalar
values.

```

4.2 Return Types

PHP 7 supports return types for functions. This feature has been available in several strongly typed languages for a long time. Now, you can easily enforce a function to return a certain type of data like so:

```

function divideValues(int $firstNumber, int $secondNumber): int {
    $value = $firstNumber / $secondNumber;
    return $value;
}

echo divideValues(8, 9);

// Result:
0

```

In the function above, we want the return value to be an integer, regardless of whatever the division turns out to be. Now the default weak(coercive) type checking in PHP comes to play again here. The value returned should be a float and it should throw a Fatal Type Error but it is automatically coerced into an integer.

Enable strict mode by placing `declare(strict_types=1);` at the top of the file and run it again. It should throw a PHP Fatal Type error like so:

```

// Error raised
PHP Fatal error:  Uncaught TypeError: Return value of divideValues() must...

// Continuation of the error message
...be of the type integer, float returned in .....

```

4.3 Strong Type Check

By default, PHP 5 and 7 allow for coercion when dealing with operations such as numeric strings. An example is this:

```
function getBookNo(int $number) {
    return "This is it: " . $number;
}
```

```
echo getBookNo("8");
```

```
// Result:
```

```
This is it: 8
```

I passed in a string and it coerced it to an integer and allowed it to run successfully. Now in PHP 7, you can be strict and ensure no form of automatic conversion occurs by declaring a strict mode at the top of your PHP file like so:

```
declare(strict_types=1);
```

```
function getBookNo(int $number) {
    return "This is it: " . $number;
}
```

```
echo getBookNo("8");
```

```
// Result:
```

```
PHP Fatal error: Uncaught TypeError: Argument 1 passed to getBookNo() must..
```

```
// Continuation of the error message
```

```
..be of the type integer, string given, called in .....
```

In PHP 5, if you pass in a float value, it automatically strips out the decimal parts and leaves you with an integer. Now in PHP 7, If you pass in a float value too, it will throw a Fatal error. This feature comes in handy when building software for financial institutions.

Note: Remember something like this in JavaScript? Where you have to write use "strict"; at the top of your JavaScript file.

Chapter 5

Error Handling, Expectations and Assertions

Many fatal and recoverable fatal errors have been converted to exceptions in PHP 7. Most errors are now reported by throwing `Error` exceptions. The `Exception` class now implements a `Throwable` Interface.

Take a look at the hierarchy below:

```
\Throwable
  \Exception (implements \Throwable)
    \LogicException
      \BadFunctionCallException
      \BadMethodCallException
      \DomainException
      \InvalidArgumentException
      \LengthException
      \OutOfRangeException

    \RuntimeException
      \OutOfBoundsException
      \OverflowException
      \RangeException
      \UnderflowException
      \UnexpectedValueException
  \Error (implements \Throwable)
```

```
\AssertionError
\ArithmeticError
\DivisionByZeroError
\ParseError
\TypeError
```

So you can catch specific errors like so:

```
try {
    // evaluate something
} catch (\ParseError $e) {
    // do something
}
```

When you typehint a function parameter, and a wrong type is passed in as an argument, PHP 7 throws a **TypeError**.

Note: In PHP 7.1, you can catch multiple errors and exceptions in one catch block like so:

```
try {
    // Some code...
} catch (ExceptionTypeA | ExceptionTypeB | ExceptionTypeC $e) {
    // Code to handle the exception
} catch (\Exception $e) {
    // ...
}
```

This is particularly useful when one method throws different type of exceptions that you can handle the same way.

Note: A new `error_clear_last()` method has been added to clear the most recent error. Once used, calling `error_get_last()` will be unable to retrieve the most recent errors.

Check out the Catching Multiple Exception Types¹ RFC.

5.1 Expectations and Assertions

Assertions are a debugging and development feature. The `assert()` function in PHP 7 is now a language construct, where the first parameter can also be an expression instead of just been a string or boolean. They have been optimized to have zero cost in production. You can now enable or disable assertions from the `PHP_INI` file like so:

¹<https://wiki.php.net/rfc/multiple-catch>

```
zend.assertions = 1 // Enable assertion
zend.assertions = 0 // Disable assertion
zend.assertions = -1 // (production mode), don't generate or execute code
```

Assertions can now throw an Exception when it fails. You can enable that from the INI file like so:

```
assert.exceptions = 1 // Throw exceptions

// or

assert.exceptions = 0 // Issue warnings, which has always been the case.
```

The `assert()` can now take in two arguments where the second argument is a custom error message. It can also be an instance of an `Exception`. An example is shown below:

```
class ProjectException extends AssertionError {}

public function checkAuthenticityOfProject() {

    /* ... */

    $projException = new ProjectException('$project was not a Project object');
    assert('$project instanceof \Unicodeveloper\Project', $projException);
}
```

Note: With this new feature, you might not need to depend on assertion libraries anymore while developing and testing your code.

Check out the Expectations RFC² for more information.

²<https://wiki.php.net/rfc/expectations>

Chapter 6

Closures and Generators

There is now a better and more performant way of binding an object scope to a closure and calling it. Before PHP 7, you would bind an object to a closure like so:

```
class NameRegister {
    private $name = "Prosper";
}

// Closure
$name = function() {
    return $this->name;
};

$name = $name->bindTo(new NameRegister, 'NameRegister');
echo $name();
```

With PHP 7, you now have a `call` method on the Closure class. So you can bind an object to a closure easily like so:

```
class NameRegister {
    private $name = "Prosper";
}

$name = function() {
    echo $this->name;
};

$name->call(new NameRegister());
```

Check out the PHP Manual: Closure::call¹ for more information.

6.1 Generator Return Expressions

Generators were introduced in PHP 5.5. Prior to PHP 7, if you tried to return anything, an error would be thrown. Now, you can use a `return` statement within a generator.

You can get the returned value by calling the `Generator::getReturn()` method. Look at the code below:

```
$square = function (array $number) {
    foreach($number as $num)
    {
        yield $num * $num;
    }

    return "Done calculating the square. What next?";
};

$result = $square([1,2,3,4,5]);

foreach($result as $value)
{
    echo $value . PHP_EOL;
}

echo $result->getReturn(); // grab the return value

// Result:
1
4
9
16
25
Done calculating the square. What next?
```

6.2 Generator Delegation

Generators can now delegate to another generator by using `yield from` like so:

¹<https://secure.php.net/manual/en/closure.call.php>


```
function square(array $number) {
    foreach($number as $num)
    {
        yield $num * $num;
    }

    yield from addition($number);
};

function addition(array $number) {
    foreach($number as $num)
    {
        yield $num + $num;
    }
}

foreach(square([1,2,3,4,5]) as $value)
{
    echo $value . PHP_EOL;
}
```

// Result:

```
1
4
9
16
25
2
4
6
8
10
```

Chapter 7

Object-Oriented Programming Enhancement

7.1 Anonymous Classes

An Anonymous class is essentially a local class without a name. Anonymous classes offer the ability to spin up throwaway objects. These objects have closure-like capabilities. An anonymous class is defined like so:

```
new class($constructor, $args) {  
}
```

A real world case is a situation where you want to have objects that implement some interfaces on the fly. Rather than having several files, where you have to define the class and then instantiate it, you can leverage anonymous classes like so:

```
$meme = new class implements MemeInterface {  
    public function memeForm($form) {  
        return $form;  
    }  
};  
  
$app = new App($meme);
```

7.2 Group Use Declarations

Group use declaration helps make the code shorter and simpler. Before now, if you are trying to use multiple classes, functions and constants from the same namespace, you have to write it like so:

```
// PHP 5
namespace Unicodeveloper\Emoji;

use Unicodeveloper\Emoji\Exceptions\UnknownMethod;
use Unicodeveloper\Emoji\Exceptions\UnknownEmoji;
use function Unicodeveloper\Emoji\Exceptions\checkForInvalidEmoji;
use const Unicodeveloper\Emoji\Exceptions\INVALID_EMOJI;

class Emoji {

}
```

With PHP 7, you can group them like so:

```
// PHP 7
namespace Unicodeveloper\Emoji;

use Unicodeveloper\Emoji\Exceptions\{
    UnknownMethod, UnknownEmoji, function checkForInvalidEmoji, const INVALID_EMOJI
};

class Emoji {

}
```

Chapter 8

Better Unicode Support

In PHP 7, all you need is the hexadecimal code appended to `**^` and you'll have your symbol/emoji as an output. An example is this:

```
function getMoney() {  
    echo "\u{1F4B0}";  
}
```

```
getMoney();  
getMoney();  
getMoney();  
getMoney();
```



Figure 8.1: Unicode Result

The enhancements were made possible from the Unicode Codepoint Escape Syntax RFC¹.

¹https://wiki.php.net/rfc/unicode_escape

8.1 IntlChar

You can as well get the name equivalent of a unicode character, say “1F4B0” via the new IntlChar class like so:

```
echo IntlChar::charName("\u{1F4B0}");
```

You can get the character from the name like so:

```
var_dump(IntlChar::charFromName("LATIN CAPITAL LETTER A"));
var_dump(IntlChar::charFromName("SNOWMAN"));
var_dump(IntlChar::charFromName("TURTLE"));
```

Note: The IntlChar class contains about 600 constants and 59 static methods.

This was made possible from the IntlChar RFC². The PHP manual has extensive documentation on the IntlChar³ class.

²<https://wiki.php.net/rfc/intl.char>

³<http://php.net/manual/en/class.intlchar.php>

Chapter 9

Deprecated & Removed Features

Using deprecated features in PHP will trigger an `E_DEPRECATED` error.

1. PHP 4 Style constructors are deprecated, and will be removed in the future. An example of a PHP 4 style of writing constructors(having the same name with the class) is this:

```
class Economy {
    function economy() {
        /* ... */
    }
}
```

2. Static calls to methods that are actually not *static* are deprecated.

```
class Economy {
    function affordPrimaryEducation() {
        echo 'I think I might not be able to afford it with this economy';
    }
}
```

```
Economy::affordPrimaryEducation();
```

```
// Result:
```

```
Deprecated: Non-static method Economy::affordPrimaryEducation() should not be called..
```

```
// Continuation of error message
```

```
..statically in .....
```

3. The salt option for the `password_hash()` function has been deprecated to prevent developers from generating their own salts which are mostly insecure.
4. The `capture_session_meta` SSL context option has been deprecated. `stream_get_meta_data()` can now be used to get SSL metadata.
5. The `ldap_sort()` function has been deprecated.
6. The alternative PHP tags shown below have been removed:

PHP Script tags

```
<script language="php">
</script>
```

PHP ASP tags

```
<% %>
```

7. The `date.timezone` warning that was always emitted in PHP 5 when a time or date-based function was used and a default timezone had not been set has been finally removed. Check out the RFC¹.
8. Before PHP 7, it was allowed to have multiple parameters with the same name like so:

```
function getUp($why, $why) {
/* */
}
```

In PHP 7, this results in an error like:

```
// Fatal error: Redefinition of parameter $why in....
```

9.1 Removed Extensions and Server APIs

The `ext/mysql`, `ext/mssql`, `ereg` and `sybase_ct` extensions have been removed. All the `mysql_` functions have been removed! You should either use the `ext/mysqli` extension or use the `ext/pdo` extension which has an object-oriented API.

The `aolserver`, `apache`, `apache_hooks`, `apache2filter`, `caudium`, `continuity`, `isapi`, `milter`, `nsapi`, `phhttpd`, `pi3web`, `roxen`, `thhttpd`, `tux` and `webjames` SAPIs have also been removed.

¹https://wiki.php.net/rfc/date.timezone_warning_removal

9.2 Backward Incompatible Changes

Here are some backward incompatible changes that you should be aware of. These are changes that have been introduced to PHP 7 but will break in lesser versions of PHP.

- `set_exception_handler()` is no longer guaranteed to receive Exception objects.
- Internal constructors always throw exceptions on failure: Prior to PHP 7, some internal classes would return **NULL** when the constructor failed. Now, they will throw an *Exception*.
- Error handling for `eval()` should now include a catch block that can handle the `ParseError`² object.
- The almighty **E_STRICT** notices now have new behaviors. It's no longer too strict.

E_STRICT notices severity changes	
Situation	New level/behaviour
Indexing by a resource	E_NOTICE
Abstract static methods	Notice removed, triggers no error
"Redefining" a constructor	Notice removed, triggers no error
Signature mismatch during inheritance	E_WARNING
Same (compatible) property in two used traits	Notice removed, triggers no error
Accessing static property non-statically	E_NOTICE
Only variables should be assigned by reference	E_NOTICE
Only variables should be passed by reference	E_NOTICE
Calling non-static methods statically	E_DEPRECATED

Source: *PHP Manual*

- `list()` can no longer unpack string variables. `str_split()` should be used when performing this form of operation.
- `global` can no longer accept *variable variables* unless you fake it by using the curly brace like so `global ${foo->bar}`.
- An **E_WARNING** will be emitted and **NULL** will be returned when internal functions try to perform float to integer automatic conversions.
- Prefixing comments with `#` in `php.ini` file is no longer allowed. Only semi-colons(`;`) should be used.
- Dividing by 0 will emit an **E_WARNING** and also one of either **+INF**, **-INF**, or **NAN**.

²<https://php.net/manual/en/class.parseerror.php>

- `$HTTP_RAW_POST_DATA` was deprecated in PHP 5.6.0 and finally removed in PHP 7.0.0. Use `php://input`³ as a replacement.
- Switch statements can no longer have multiple default blocks. An **E_COMPILE_ERROR** will be triggered if you try to define more than one default block.
- Functions can not have multiple parameters with the same name. function `slap($hand, $hand, $strength)`. An **E_COMPILE_ERROR** will be triggered as a result of this function.
- Static calls made to a non-static method with an incompatible context will now result in the called method having an undefined `$this` variable and a deprecation warning being issued.

You can check out the few other PHP core functions⁴ that have changed.

³<https://php.net/manual/en/wrappers.php.php#wrappers.php.input>

⁴<https://secure.php.net/manual/en/migration70.changed-functions.php>

Chapter 10

Uniform Variable Syntax and Static Values

Uniform Variable Syntax brings a much needed change to the way variable-variable expressions are constructed. It allows for a number of new combinations of operators that were previously disallowed, and so introduces new ways to achieve old operations in a more polished code.

```
// nesting ::  
$foo::$bar::$baz // access the property $baz of the $foo::$bar property  
  
// nesting ()  
foo()() // invoke the return of foo()  
  
// operators on expressions enclosed in ()  
(function () {} )() // IIFE syntax from JS  
  
$foo['bar']['baz']           // old meaning           // new meaning  
$foo->$bar['baz']           ${$foo['bar']['baz']}   ($$foo)['bar']['baz']  
$foo->$bar['baz']()         $foo->{$bar['baz']}   ($foo->$bar)['baz']  
$foo->$bar['baz']()         $foo->{$bar['baz']}() ($foo->$bar)['baz']()  
Foo::$bar['baz']()         Foo::{$bar['baz']}() (Foo::$bar)['baz']()
```

10.1 Accessing Static Values

In PHP 5.x, if you try to access a static value like so, an error will be triggered:

```
class Auth0 {
    static $lock = 'v10';
}
```

```
echo 'Auth0::$lock;
```

```
// Result
```

Parse error: syntax error, unexpected '::' (T_PAAMAYIM_NEKUDOTAYIM), expecting ',' or ';'!

Now, In PHP 7.x, it throws no error, it simply works!

```
// PHP 7
```

```
class Auth0 {
    static $lock = 'v10';
}
```

```
echo 'foo::$lock;
```

```
// Result
```

```
v10
```

Chapter 11

Migration Tools

One of the most frustrating part of our jobs as software developers is having to work on large old codebases. In a situation where you are tasked with migrating a large PHP 5.x application that has probably been in existence for about 10 years, how would you go about it?

Professionally, production codebases should be backed up with test suites. But let's face reality, there are lots of old codebases that exists without tests.

If your codebase is backed with a comprehensive test suite, then it is easy for you to make changes to incorporate PHP 7 features without messing up the software.

The easiest and most obvious way of migrating old codebases without test suites is to clone the app on your local machine, install PHP 7 and run the app. You can walk through the errors and deprecation warnings shown in the terminal, and manually fix them step-by-step by incorporating PHP 7 features. But this can be very challenging and time consuming. Why can't we automate this process?

Currently there is no tool out there that performs a 100% automatic conversion of your PHP 5.x codebase to PHP 7, but the tools I'll mention in the next section will help in making your migration painless.

11.1 PHP 7 MAR

php7mar¹ is a command-line tool that generates reports on PHP 5.x codebase based on PHP 7 compatibility. The reports contain line numbers, issues noted, and suggested fixes along with documentation links.

¹<https://github.com/Alexia/php7mar>

Note: This tool does not fix code. It only gives you reports about all the PHP files in your codebase. Happy fixing!

11.2 PHP 7 Compatibility Checker

php7cc² is a command-line tool designed to make migration from PHP 5.3 - 5.6 to PHP 7 really easy. php7cc reports:

- **Errors:** Fatal, Syntax, Notice. These are highlighted in red.
- **Warnings:** These are highlighted in yellow.

11.3 Phan

phan³ is a static analyzer for PHP that attempts to prove incorrectness rather than correctness. Phan looks for common issues and verifies type compatibility on various operations when type information is available or can be deduced. Phan checks for lots of things including PHP7/PHP5 backward compatibility.

11.4 phpto7aid

phpto7aid⁴ is a tool that is used to identify PHP 5 code that will not work in PHP 7. It tries to aid you as much as possible in resolving this issues, by either providing the exact solution or giving hints on how to solve the issue.

11.5 PhpStorm PHP 7 Compatibility Inspection

PhpStorm⁵ is a very smart PHP IDE, developed by JetBrains⁶.

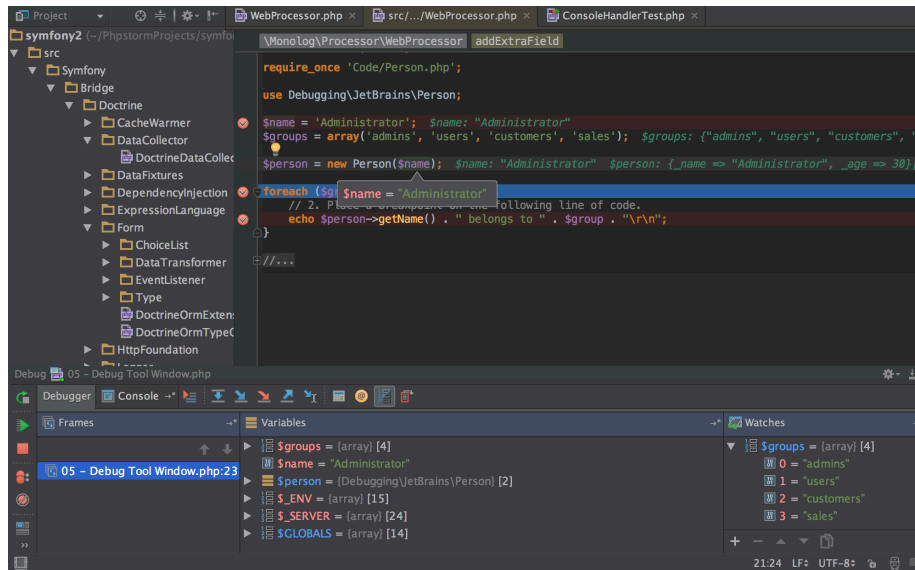
²<https://github.com/sstalle/php7cc>

³<https://github.com/etsy/phan>

⁴<https://github.com/gisostallenberg/php-to-7-aid>

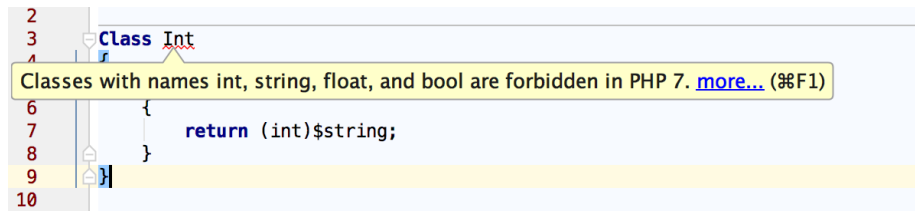
⁵<https://www.jetbrains.com/phpstorm>

⁶<https://www.jetbrains.com>



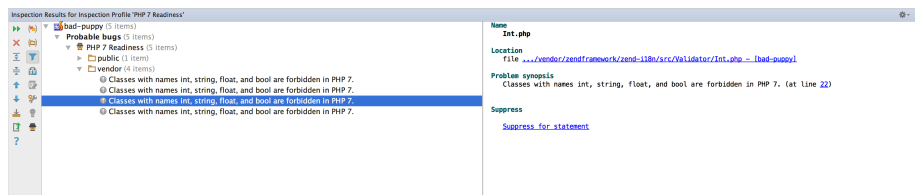
Source: Jetbrains.com

PhpStorm 10 ships with a *PHP 7 Compatibility Inspection* tool that can show you exactly what code is going to cause errors if you are running PHP7.



Source: Jetbrains.com

The image below shows a typical example of an application that has classes with names that are reserved in PHP 7. Selecting **Run Inspection By Name** option from the **Code** menu, and then selecting the **PHP 7 Compatibility** section will give you results like this one below:



Source: Jetbrains.com

Chapter 12

Practical Migration of Two Apps

Let's do a practical migration of two apps, a basic web app and an API.

12.1 Building a PHP5 App

We will build this PHP 5 app very quickly. The scope of the app can be found below:

- A user will be able to register on the app.
- A user will be able to log into the app.
- A user will be assigned a random Star Wars Code Name.
- A user will be able to log out of the app.

Building this app will require us to set up a database to store the users, write our registration and login code and manage the users session. Now, we won't employ the use of any framework because we don't want any form of overhead. Ordinarily, building this app will take a lot of time and setup but there is a service we can use to eliminate the hassle. Oh, yeah, let's use Auth0¹ to save us time and make our app secure.

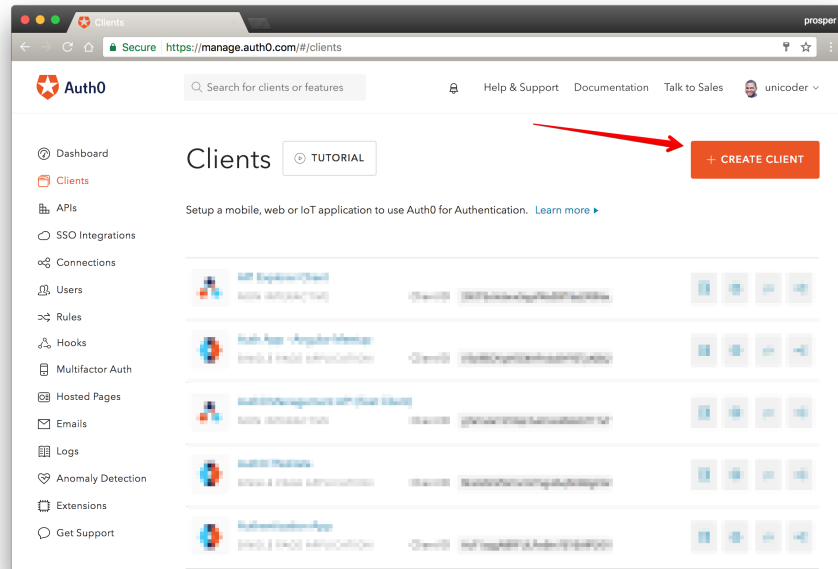
12.1.1 Create and Configure Auth0 Client

First thing we'll need to do is sign up for a free Auth0 account² and configure a new client.

¹<https://auth0.com>

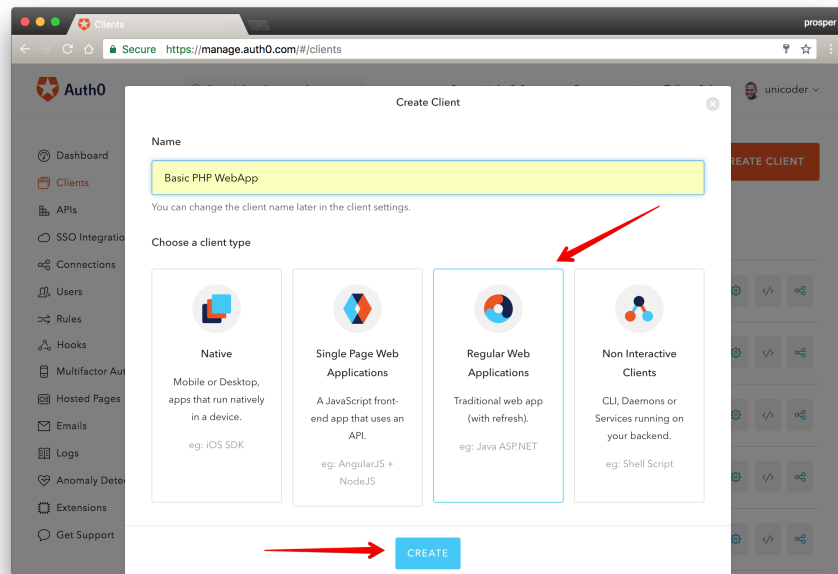
²<https://auth0.com/signup>

Now head over to clients³ and create a new one choosing Regular web Application as the client type. Let's name it as something like Basic PHP WebApp.



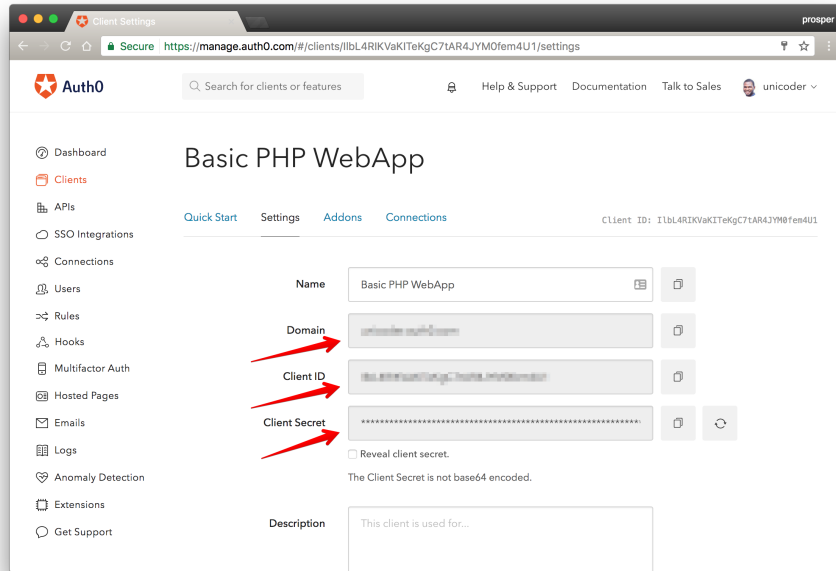
Click on the Create Client button

³<https://manage.auth0.com/#/clients>



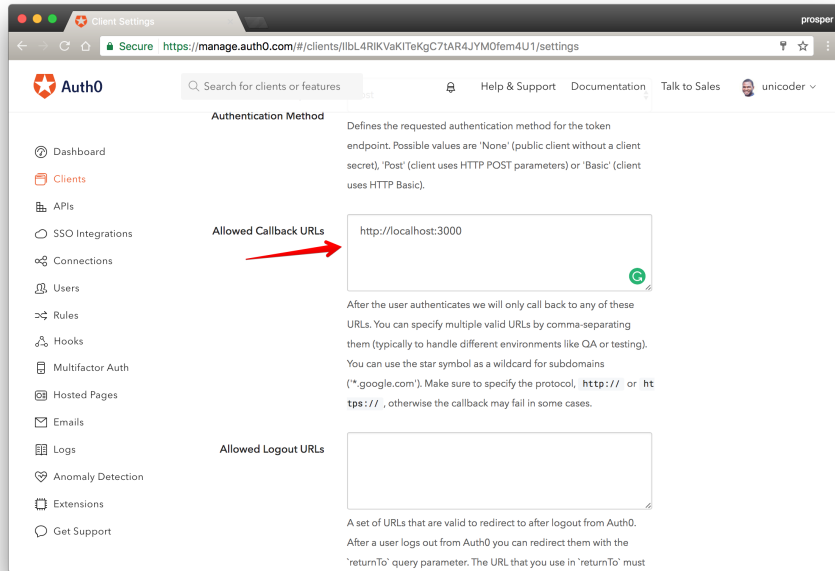
Create a regular web application

Now that we have our client created, we need to take note of three properties: **Domain**, **Client ID** and **Client Secret**. All of them can be found on the **Settings** tab of the client that we've just created.



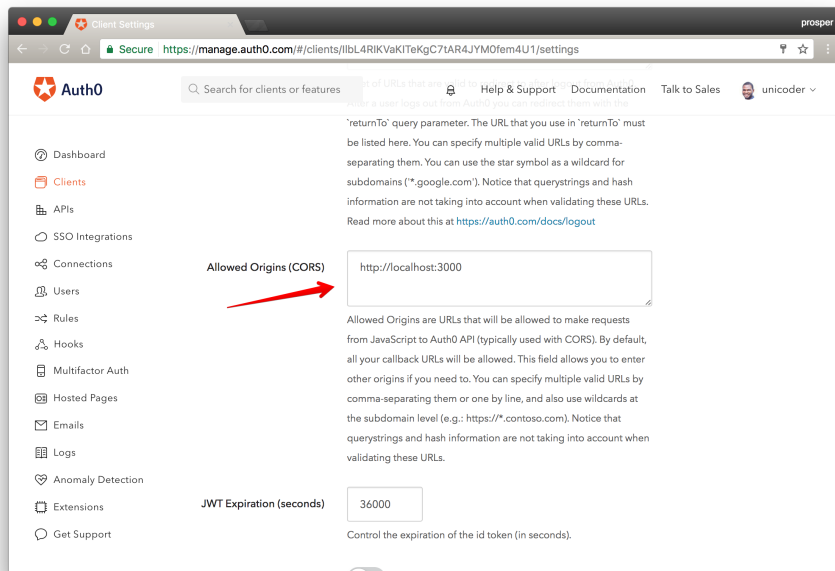
Grab the Domain, Client ID and Client Secret

The last configuration that we need to do, before updating our code, is to add `http://localhost:3000` as an Allowed Callback URLs in our Auth0 client. Just scroll down on the Settings tab, you'll see the field like so:



Set Allowed Callback URL

Oh, just one more config, I promise. We need to add `http://localhost:3000` as **Allowed Origins (CORS)**. Scroll down a bit more, you'll see the field like so:



Set Allowed Origins(CORS)

That's all for now.

12.1.2 Build the App

Create a `composer.json` file in a new directory and add this to it like so:

```
{
  "name": "basic php webapp",
  "description": "Basic sample for securing a WebApp with Auth0",
  "require": {
    "vlucas/phpdotenv": "2.3.0",
    "auth0/auth0-php": "~4.0"
  },
  "license": "MIT"
}
```

composer.json

All we need is the `phpdotenv` package for reading environment variables and the `auth0-php` package that makes it easy to use the Auth0 service.

Create a `public` folder inside the directory and add two files, `app.css` and `app.js` in it.

```
body {
  font-family: "proxima-nova", sans-serif;
  text-align: center;
  font-size: 300%;
  font-weight: 100;
}
input[type=checkbox],
input[type=radio] {
  position: absolute;
  opacity: 0;
}
input[type=checkbox] + label,
input[type=radio] + label {
  display: inline-block;
}
input[type=checkbox] + label:before,
input[type=radio] + label:before {
  content: "";
  display: inline-block;
  vertical-align: -0.2em;
}
```

```

width: 1em;
height: 1em;
border: 0.15em solid #0074d9;
border-radius: 0.2em;
margin-right: 0.3em;
background-color: white;
}
input[type=radio] + label:before {
border-radius: 50%;
}
input[type=radio]:checked + label:before,
input[type=checkbox]:checked + label:before {
background-color: #0074d9;
box-shadow: inset 0 0 0 0.15em white;
}
input[type=radio]:focus + label:before,
input[type=checkbox]:focus + label:before {
outline: 0;
}
.btn {
font-size: 140%;
text-transform: uppercase;
letter-spacing: 1px;
border: 0;
background-color: #16214D;
color: white;
}
.btn:hover {
background-color: #44C7F4;
}
.btn:focus {
outline: none !important;
}
.btn.btn-lg {
padding: 20px 30px;
}
.btn.disabled {
background-color: #333;
color: #666;
}
h1,
h2,
h3 {
font-weight: 100;
}
#logo img {

```

```

    width: 300px;
    margin-bottom: 60px;
  }
  .home-description {
    font-weight: 100;
    margin: 100px 0;
  }
  h2 {
    margin-top: 30px;
    margin-bottom: 40px;
    font-size: 200%;
  }
  label {
    font-size: 100%;
    font-weight: 300;
  }
  .btn-next {
    margin-top: 30px;
  }
  .answer {
    width: 70%;
    margin: auto;
    text-align: left;
    padding-left: 10%;
    margin-bottom: 20px;
  }
  .login-page .login-box {
    padding: 5px 0;
  }

```

app.css

```

$(document).ready(function() {

    var lock = new Auth0Lock(AUTHO_CLIENT_ID, AUTHO_DOMAIN, { auth: {
      redirectUrl: AUTHO_CALLBACK_URL
      , responseType: 'code'
      , params: {
        scope: 'openid'
      }
    }
    });

    $('<span>.btn-login</span>').click(function(e) {
      e.preventDefault();
      lock.show();
    });

```

```
});
```

```
app.js
```

Go ahead and create a `.htaccess` file inside the directory like so:

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . index.php [L]
```

```
.htaccess
```

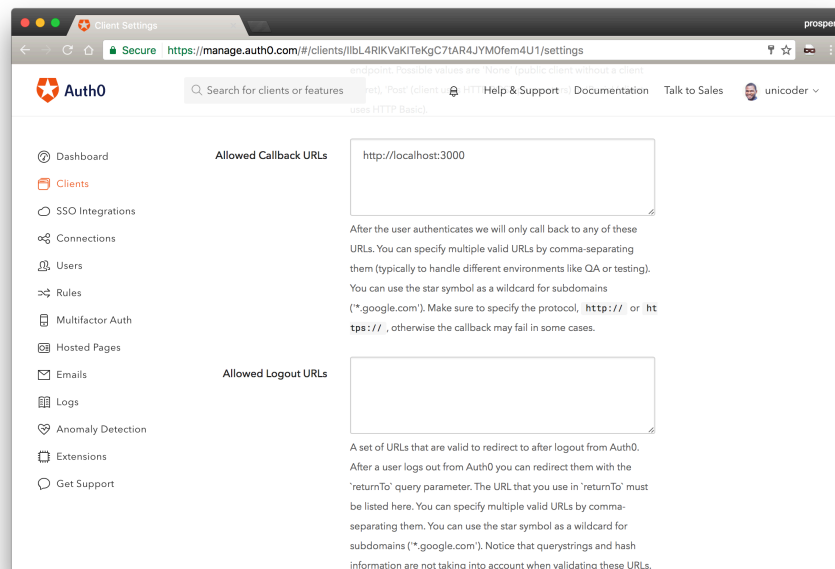
Create a `.env` file. This file will contain our Auth0 credentials.

```
AUTH0_DOMAIN='blahabababababa.auth0.com'
AUTH0_CLIENT_ID='xxxxxxxxx'
AUTH0_CLIENT_SECRET='xxxxxxxxx'
AUTH0_CALLBACK_URL='http://localhost:3000'
```

```
.env
```

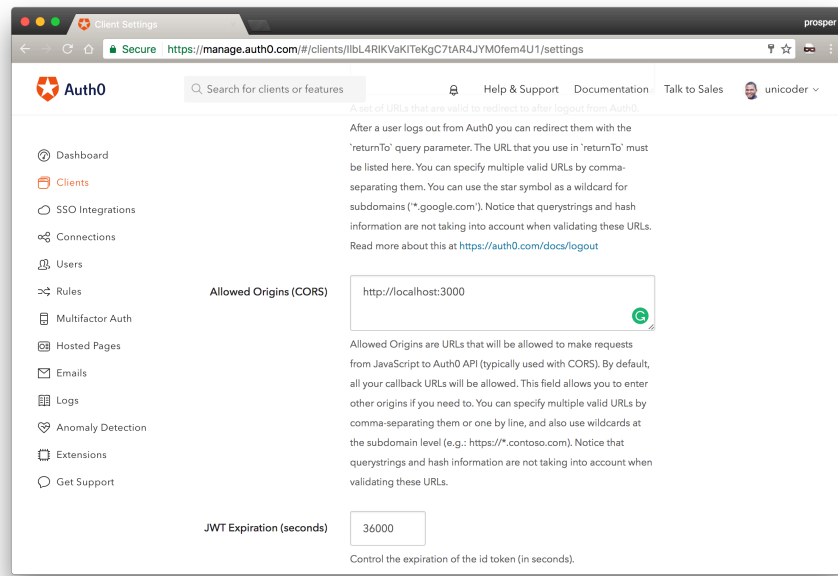
Note: Replace these values with the `client_id`, `client_secret` and domain values from your Auth0 dashboard.

Add the value of `callback_url` to the **Allowed Callback URLs** in your *Settings* on the dashboard.



Auth0 dashboard: Allowed Callback Urls

Also, do not forget to add the same value to the **Allowed Origins(CORS)** in your *Settings* on the dashboard.



Auth0 dashboard: Allowed Origin CORS

We need a file to invoke the `dotenv` library and load the values that we have deposited in the `.env` file. Create a new file, `dotenv-loader.php` like so:

```
<?php
// Read .env
try {
    $dotenv = new Dotenv\Dotenv(__DIR__);
    $dotenv->load();
} catch(InvalidArgumentException $ex) {
    // Ignore if no dotenv
}
```

dotenv-loader.php

Finally, let's create the `index.php` file where all our app logic will reside. Like I mentioned earlier, it's just a basic app so don't be worried about separation of concerns.

This is how the file should look like:


```

<?php

// Require composer autoloader
require __DIR__ . '/vendor/autoload.php';

require __DIR__ . '/dotenv-loader.php';

use Auth0\SDK\API\Authentication;

$domain      = getenv('AUTHO_DOMAIN');
$client_id   = getenv('AUTHO_CLIENT_ID');
$client_secret = getenv('AUTHO_CLIENT_SECRET');
$redirect_uri = getenv('AUTHO_CALLBACK_URL');

$auth0 = new Authentication($domain, $client_id);

$auth0auth = $auth0->get_oauth_client($client_secret, $redirect_uri, [
    'persist_id_token' => true,
    'persist_refresh_token' => true,
]);

$starWarsNames = ['Darth Vader', 'Ahsoka Tano', 'Kylo Ren', 'Obi-Wan Kenobi', 'R2-D2', 'Snow'];

$userInfo = $auth0auth->getUser();

if (isset($_REQUEST['logout'])) {
    $auth0auth->logout();
    session_destroy();
    header("Location: /");
}

?>
<html>
  <head>
    <script src="http://code.jquery.com/jquery-3.0.0.min.js" type="text/javascript"></script>
    <script src="https://cdn.auth0.com/js/lock/10.0/lock.min.js"></script>

    <script type="text/javascript" src="//use.typekit.net/iws6ohy.js"></script>
    <script type="text/javascript">try{Typekit.load();}catch(e){}</script>

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="icon" type="image/png" href="/favicon-32x32.png" sizes="32x32">

    <!-- font awesome from BootstrapCDN -->
    <link href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">

```

```

<link href="//maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css"
</link>

<script>
    var AUTHO_CLIENT_ID = '<?php echo getenv("AUTHO_CLIENT_ID") ?>';
    var AUTHO_DOMAIN = '<?php echo getenv("AUTHO_DOMAIN") ?>';
    var AUTHO_CALLBACK_URL = '<?php echo getenv("AUTHO_CALLBACK_URL") ?>';
</script>

<script src="public/app.js"> </script>
<link href="public/app.css" rel="stylesheet">

</head>
<body class="home">
    <div class="container">
        <div class="login-page clearfix">
            <?php if(!$userInfo): ?>
            <div class="login-box auth0-box before">
                
                <p>Heard you don't want to migrate to PHP 7? Dare us!</p>
                <a class="btn btn-primary btn-login">SignIn</a>
            </div>
            <?php else: ?>
            <div class="logged-in-box auth0-box logged-in">
                <h1 id="logo">Star Wars Welcomes You to the Family!</h1>
                

                <h2>Welcome <span class="nickname"><?php echo $userInfo['nickname'] ?></span>
                <h2> Assigned Codename : <b><?php echo $starWarsNames[rand(0, 6)]; ?></b> </h2>
                <a class="btn btn-primary btn-lg" href="?">Logout</a>
            </div>
            <?php endif ?>
        </div>
    </div>
</body>
</html>

```

I know it seems overwhelming to just hit you with that block of code at once. Just relax, let's analyze the code together.

```

// Require composer autoloader
require __DIR__ . '/vendor/autoload.php';

require __DIR__ . '/dotenv-loader.php';

```

import the autoloader and environment loader

This is where we require the dotenv loader and composer autoloader. The autoloader makes it possible for us to import any class from the PHP packages installed in the app.

```
use Auth0\SDK\API\Authentication;

$domain      = getenv('AUTHO_DOMAIN');
$client_id   = getenv('AUTHO_CLIENT_ID');
$client_secret = getenv('AUTHO_CLIENT_SECRET');
$redirect_uri = getenv('AUTHO_CALLBACK_URL');

$auth0 = new Authentication($domain, $client_id);

$auth0auth = $auth0->get_oauth_client($client_secret, $redirect_uri, [
    'persist_id_token' => true,
    'persist_refresh_token' => true,
]);

$starWarsNames = ['Darth Vader', 'Ahsoka Tano', 'Kylo Ren', 'Obi-Wan Kenobi', 'R2-D2', 'Snow'];

$userInfo = $auth0auth->getUser();
```

Grab Auth details and user information

Auth0\SDK\API\Authentication is the Auth0 authentication class. It has the methods to retrieve a user's profile when logged in. `$domain`, `$client_id`, `$client_secret`, `$redirect_uri` are variables that will house the values gotten from the `.env` file with the aid of the `getenv` method.

Then, we moved on to instantiating the `Authentication` class.

The `$auth0->get_oauth_client()` method by default stores user information in the PHP session, and we also instructed it to save the `access_token` and `id_token` that Auth0 server returns during the process of successfully authenticating a user.

`$starWarsNames` array contains some characters from Star Wars⁴. Later in the code, a user will be assigned a random code name from this array.

`$auth0auth->getUser()` retrieves the user information.

```
if (isset($_REQUEST['logout'])) {
    $auth0auth->logout();
    session_destroy();
}
```

⁴<http://www.starwars.com>

```

    header("Location: /");
}

```

Log out a user, destroy all sessions and redirect to index page

This piece of code above checks if the user submitted a request to log out, clears the session and redirects the user back to the homepage.

```

<script src="http://code.jquery.com/jquery-3.0.0.min.js" type="text/javascript"></script>
<script src="https://cdn.auth0.com/js/lock/10.0/lock.min.js"></script>

```

Include Auth0 lock widget and jQuery

We are making use of Auth0 Lock widget. We are also using jQuery to call the lock methods and handle the button click event.

```

<!-- font awesome from BootstrapCDN -->
<link href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet" />
<link href="//maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css" rel="stylesheet" />

```

Include bootstrap and font-awesome

We pulled in bootstrap and font-awesome for beautification.

```

<script>
    var AUTHO_CLIENT_ID = '<?php echo getenv("AUTHO_CLIENT_ID") ?>';
    var AUTHO_DOMAIN = '<?php echo getenv("AUTHO_DOMAIN") ?>';
    var AUTHO_CALLBACK_URL = '<?php echo getenv("AUTHO_CALLBACK_URL") ?>';
</script>

```

In the code above, we fed the Auth0 credentials to some JavaScript variables.

```

<div class="container">
    <div class="login-page clearfix">
        <?php if(!$userInfo): ?>
            <div class="login-box auth0-box before">
                
                <p>Heard you don't want to migrate to PHP ?? Dare us!</p>
                <a class="btn btn-primary btn-login">SignIn</a>
            </div>
            <?php else: ?>
                <div class="logged-in-box auth0-box logged-in">
                    <h1 id="logo">Star Wars Welcomes You to the Family!</h1>
                    

                    <h2>Welcome <span class="nickname"><?php echo $userInfo['nickname'] ?></span></h2>
                    <h2> Assigned Codename : <b><?php echo $starWarsNames[rand(0, 6)]; ?></b> </h2>
                    <a class="btn btn-primary btn-lg" href="?logout">Logout</a>
                </div>
            <?php endif ?>
        </div>
    </div>

```

In the code above, if the `$userInfo` is not set, then it means the user has not logged in yet, so we display the `signin` button. If the user has signed in, then we grab the user's info and display it along with the `logout` button.

12.1.3 Run The App

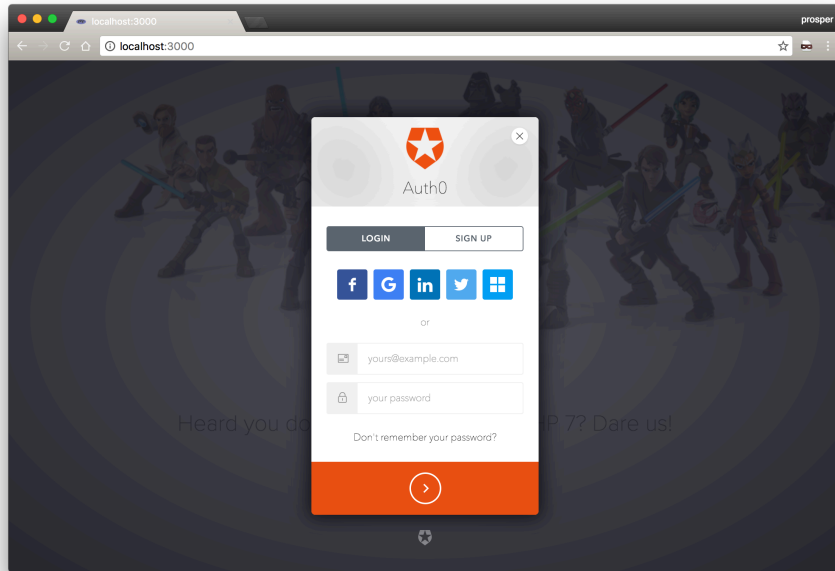
Go to your terminal and run `composer install` to install the dependencies. Next, run your PHP 5.x server. If your PHP server is accessible from the terminal, then you can run it via `php -S localhost:3000`.

Open your browser and test the app. The index page should look like this:



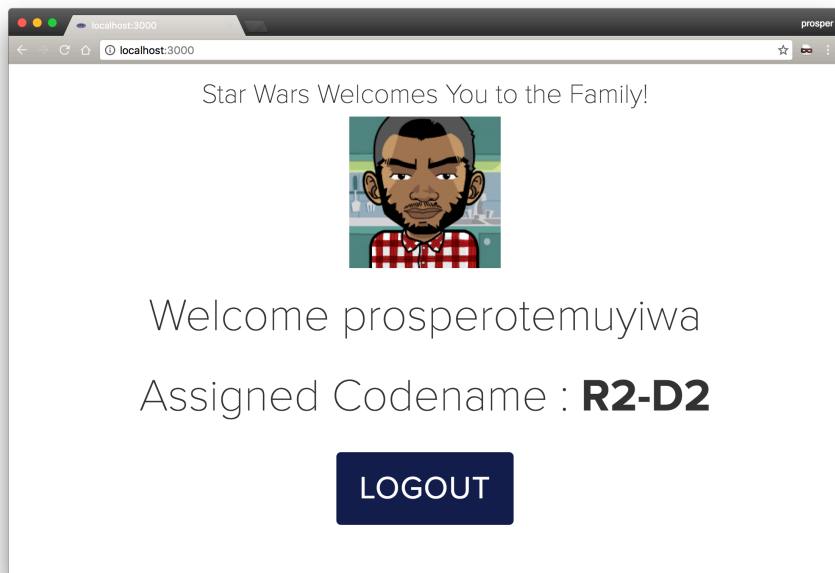
Index Page

Now, signup & signin.



Sign In

When you are logged in, you should be assigned a Star Wars codename like so:



Logged In

Our app is now running successfully on a PHP 5.x server. You can grab the source code from Github⁵ to ensure that everything works as expected.

12.1.4 Migrate to PHP 7

We are currently running a PHP 5.x app. Let's migrate it to PHP 7. The good thing like I mentioned earlier is that most times you might not have to change anything in the codebase. Let's see if that holds true for this app.

Upgrade your server to at least PHP 7.0.0 and run this app again.

```
prosperotemuyiwa@PROSPERS-MacBook-Pro ~ % cd /Users/prosperotemuyiwa/source/php-workspace/basic-webapp && php -S localhost:3000
PHP 7.1.0 Development Server started at Thu Jan 26 09:58:12 2017
Listening on http://localhost:3000
Document root is /Users/prosperotemuyiwa/source/php-workspace/basic-webapp
Press Ctrl-C to quit.
[Thu Jan 26 09:58:17 2017] 1:51499 200 : /
[Thu Jan 26 09:58:17 2017] 1:51502 200 : /public/app.js
[Thu Jan 26 09:58:17 2017] 1:51503 200 : /public/app.css
[Thu Jan 26 09:58:22 2017] 1:51504 302 : /logout
[Thu Jan 26 09:58:22 2017] 1:51520 200 : /
[Thu Jan 26 09:58:22 2017] 1:51521 200 : /public/app.css
[Thu Jan 26 09:58:22 2017] 1:51522 200 : /public/app.js
[Thu Jan 26 09:58:31 2017] 1:51527 200 : /?code=4X104p2BvxwgPAXB
[Thu Jan 26 09:58:31 2017] 1:51528 200 : /public/app.js
[Thu Jan 26 09:58:31 2017] 1:51529 200 : /public/app.css
```

PHP 7 Server running

Signup, Login and try to logout. There are no errors.

Awesome, now our first app is running on PHP 7 successfully!

12.2 API

We will clone an API. It is a simple Chuck Norris API. It has been built already with PHP 5 in mind.

Go ahead and clone the project from Github⁶ and run `composer install` to install all the dependencies. Then run the app on a PHP 5.x server.

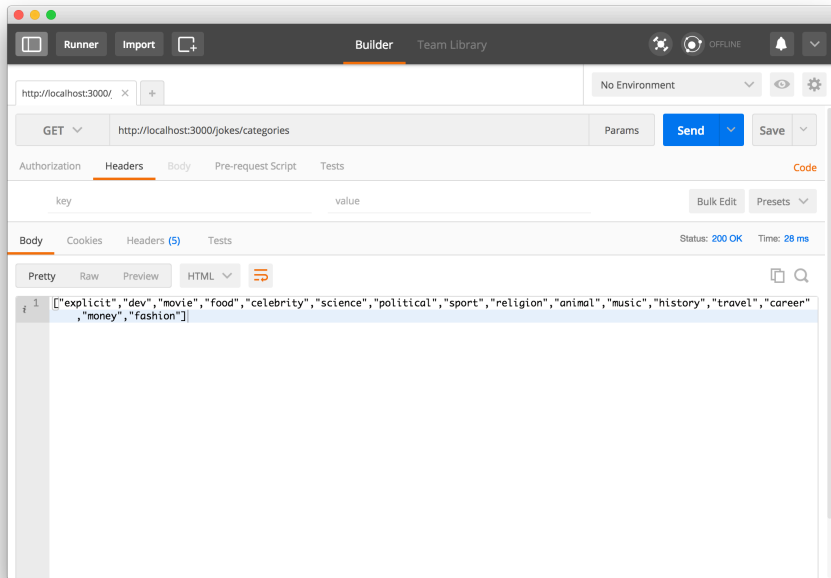
Open up Postman⁷ and test the API like so:

Run `http://localhost:3000/jokes/categories` like so:

⁵<https://github.com/auth0-blog/starwars-phpapp>

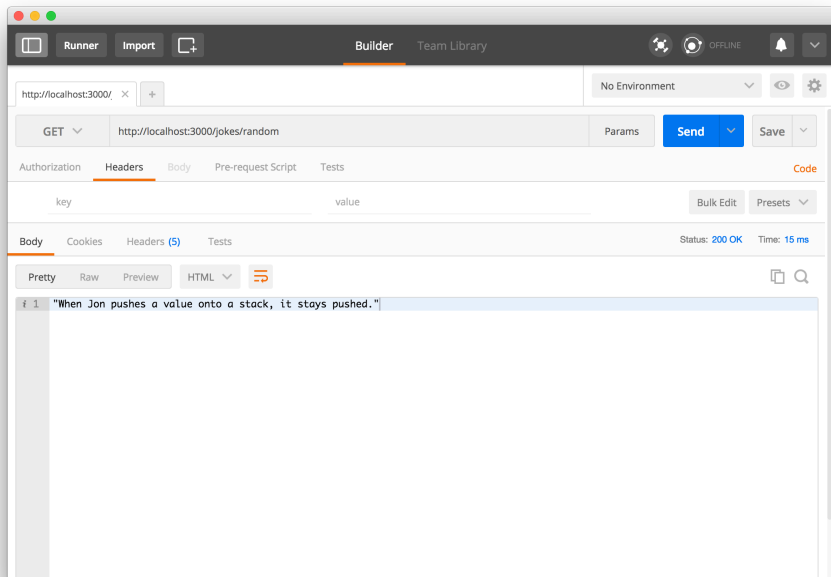
⁶<https://github.com/auth0-blog/basic-api>

⁷<https://www.getpostman.com>



API showing categories

Run `http://localhost:3000/jokes/random` like so:



API showing random jokes

The app is working fine, no errors!

12.2.1 Use PHP 7 Features

Let's refactor this app and inject some PHP 7 features.

This is the directory structure of our API app at the moment:

```
----basic-api
|
|----src
|  |
|  ----Main.php
|
|----vendor
|
|----.gitignore
|
|----.htaccess
|
|----composer.json
|
|----composer.lock
|
|----index.php
|
|----README.md
```

This is how our `Main.php` file looks like right now:

```
<?php

namespace App;

use Exception;

class Main {

    public function getCategories() {
        return $this->getCategoryData();
    }

    private function getCategoryData() {
```

```

        return [
            "explicit",
            "dev",
            "movie",
            "food",
            "celebrity",
            "science",
            "political",
            "sport",
            "religion",
            "animal",
            "music",
            "history",
            "travel",
            "career",
            "money",
            "fashion"
        ];
    }

    public function getRandomJokes($randomNumber) {

        if( !is_integer($randomNumber)) {
            throw new Exception("The random number should be an integer. Please try again.")
        }

        $jokes = [
            "Jon Skeet's code doesn't follow a coding convention. It is the coding convention.",
            "Jon Skeet can divide by Zero.",
            "Jon Skeet points to null, null quakes in fear.",
            "Jon Skeet is the traveling salesman. Only he knows the shortest route.",
            "When Jon pushes a value onto a stack, it stays pushed.",
            "Drivers think twice before they dare interrupt Jon's code.",
            "Jon Skeet does not sleep... He waits.",
            "Jon Skeet can stop an infinite loop just by thinking about it.",
            "Jon Skeet uses Visual Studio to burn CDs.",
            "Jon Skeet has the key to Open Source. He just doesn't want to close it."
        ];

        return $jokes[$randomNumber];
    }
}

```

Let's start by adding **PHP 7 return type declarations** to the methods in this class like so:

```

<?php

namespace App;

class Main {

    public function getCategories(): array {
        return $this->getCategoryData();
    }

    private function getCategoryData(): array {
        return [
            "explicit",
            "dev",
            "movie",
            "food",
            "celebrity",
            "science",
            "political",
            "sport",
            "religion",
            "animal",
            "music",
            "history",
            "travel",
            "career",
            "money",
            "fashion"
        ];
    }

    public function getRandomJokes($randomNumber): string {

        if( !is_integer($randomNumber)) {
            throw new Exception("The random number should be an integer. Please try again.");
        }

        $jokes = [
            "Jon Skeet's code doesn't follow a coding convention. It is the coding convention.",
            "Jon Skeet can divide by Zero.",
            "Jon Skeet points to null, null quakes in fear.",
            "Jon Skeet is the traveling salesman. Only he knows the shortest route.",
            "When Jon pushes a value onto a stack, it stays pushed.",
            "Drivers think twice before they dare interrupt Jon's code.",
            "Jon Skeet does not sleep... He waits.",
            "Jon Skeet can stop an infinite loop just by thinking about it.",
        ];
    }
}

```

```

        "Jon Skeet uses Visual Studio to burn CDs.",
        "Jon Skeet has the key to Open Source. He just doesn't want to close it."
    ];

    return $jokes[$randomNumber];
}
}

```

PHP 7 Return Type Declarations added in Main.php

Another PHP 7 feature we can add is *function parameter typehinting*. We have a method, `getRandomJokes($randomNumber)` that accepts a `$randomNumber` which is an integer.

Let's refactor that method, `getRandomJokes()`. We'll eliminate the `if` condition and just typehint the `$randomNumber` parameter like so:

```

public function getRandomJokes(int $randomNumber): string {

    $jokes = [
        "Jon Skeet's code doesn't follow a coding convention. It is the coding convention.",
        "Jon Skeet can divide by Zero.",
        "Jon Skeet points to null, null quakes in fear.",
        "Jon Skeet is the traveling salesman. Only he knows the shortest route.",
        "When Jon pushes a value onto a stack, it stays pushed.",
        "Drivers think twice before they dare interrupt Jon's code.",
        "Jon Skeet does not sleep... He waits.",
        "Jon Skeet can stop an infinite loop just by thinking about it.",
        "Jon Skeet uses Visual Studio to burn CDs.",
        "Jon Skeet has the key to Open Source. He just doesn't want to close it."
    ];

    return $jokes[$randomNumber];
}

```

Now if you try to pass in a value besides an integer like so:

```

$router->get('/jokes/random', function() use ($app){
    echo json_encode($app->getRandomJokes("dsdsds"));
});

```

index.php

PHP 7 will throw a Type Error like so:

```

[Thu Jan 26 15:48:37 2017] PHP Fatal error: Uncaught TypeError: Argument 1 passed to App\Main::getRandomJokes() must be of the type integer, string given, called in /Users/prosperotemuyiwa/source/php-workspace/basic-api/index.php on line 19 and defined in /Users/prosperotemuyiwa/source/php-workspace/basic-api/src/Main.php:34
Stack trace:
#0 /Users/prosperotemuyiwa/source/php-workspace/basic-api/index.php(19): App\Main->getRandomJokes('dsdsds')
#1 [internal function]: {closure}()
#2 /Users/prosperotemuyiwa/source/php-workspace/basic-api/vendor/bramus/router/src/Bramus/Router/Router.php(329): call_user_func_array(Object(Closure), Array)
#3 /Users/prosperotemuyiwa/source/php-workspace/basic-api/vendor/bramus/router/src/Bramus/Router/Router.php(253): Bramus\Router\Router->handle(Array, true)
#4 /Users/prosperotemuyiwa/source/php-workspace/basic-api/index.php(32): Bramus\Router\Router->run()
#5 {main}
    thrown in /Users/prosperotemuyiwa/source/php-workspace/basic-api/src/Main.php on line 34
[Thu Jan 26 15:48:37 2017] - 1.01338 [200]: /jokes/random - Uncaught TypeError: Argument 1 passed to App\Main::getRandomJokes() must be of the type integer, string given, called in /Users/prosperotemuyiwa/source/php-workspace/basic-api/index.php on line 19 and defined in /Users/prosperotemuyiwa/source/php-workspace/basic-api/src/Main.php:34
Stack trace:
#0 /Users/prosperotemuyiwa/source/php-workspace/basic-api/index.php(19): App\Main->getRandomJokes('dsdsds')
#1 [internal function]: {closure}()
#2 /Users/prosperotemuyiwa/source/php-workspace/basic-api/vendor/bramus/router/src/Bramus/Router/Router.php(329): call_user_func_array(Object(Closure), Array)
#3 /Users/prosperotemuyiwa/source/php-workspace/basic-api/vendor/bramus/router/src/Bramus/Router/Router.php(253): Bramus\Router\Router->handle(Array, true)
#4 /Users/prosperotemuyiwa/source/php-workspace/basic-api/index.php(32): Bramus\Router\Router->run()
#5 {main}
    thrown in /Users/prosperotemuyiwa/source/php-workspace/basic-api/src/Main.php on line 34

```

PHP 7 TypeError

We have been able to add some PHP 7 features. The app also runs on a PHP 7 server and everything just works fine!

The source code of the PHP 7 version of the API can be found on the [php7](#) branch on [GitHub](#)⁸.

⁸<https://github.com/auth0-blog/basic-api/tree/php7>

Chapter 13

Introducing PHP 7.1 Features

PHP 7.1 was released in December 3, 2016 and it came bundled with some new features. Let's take a good look at these features.

13.1 Nullable Types

Return types were introduced in PHP 7.0. The PHP team took it a step further by allowing nullable types for parameters and return types in PHP 7.1. This simply means null can be returned or allowed as parameters if you define them like so:

```
function whatIsYourName(): ?string {  
    return null;  
}  
  
$result = whatIsYourName() ? "Great" : "Nah";  
echo is_null(whatIsYourName());  
  
// Result:  
Nah  
1
```

The question mark, `?` behind the `string` return type ensures that method is allowed to return null.

From the code above, you can see it returns `1` which means `true`. And the value of `whatIsYourName()` returns null.

13.2 Void Type

Another return type, `void` has been added in PHP 7.1. Functions that have a `void` return type can decide not to have a `return` statement or use an empty `return` statement.

Note: `Null` is not a valid return value for a `void` function.

```
function getGeniusBrain(): void {  
    echo "There is no genius brain around";  
}
```

```
getGeniusBrain();
```

```
// Result:
```

```
There is no genius brain around
```

If you change the `void` return type to say, `string`. PHP 7 will throw a Fatal Type Error.

13.3 Symmetric Array Destructuring

In PHP 7.1, you can now use the short array syntax, `[]`, to destructure arrays for assignments. The normal way is to use `list()`. You can use, `[]`, like so:

```
$fruits = ['mango', 'banana', 'apple'];
```

```
[$mango, $banana, $apple] = $fruits;
```

```
echo $mango.PHP_EOL;
```

```
echo $banana.PHP_EOL;
```

```
echo $apple.PHP_EOL;
```

```
// Result
```

```
mango
```

```
banana
```

```
apple
```

You can also use, `[]`, with `foreach` like this:

```
foreach ($fruits as [$a, $b, $c]) {  
    /** **/  
}
```

13.4 Class Constant Visibility

In PHP 7.1, you can now specify the visibility of class constants. I really love this feature. Let's take a look.

```
<?php

class Baba {

    const children = 7;
    public const CONCUBINES = 2;
    protected const BUSINESSES = 10;
    private const WIVES = 4;
}

echo Baba::children;
echo Baba::CONCUBINES;

// Result:
72

echo Baba::BUSINESSES;
// Result:
Fatal error: Uncaught Error: Cannot access protected const Baba::BUSINESSES in ...

echo Baba::WIVES;
// Result:
Fatal error: Uncaught Error: Cannot access private const Baba::WIVES in ...
```

So, we can now restrict constants with `protected` and `private` access modifiers so that they are not accessible outside a class. Sweet!

13.5 Multi-Catch Exception Handling

I mentioned this feature in Chapter 5. PHP 7.1 allows us to catch and respond to multiple exception types with the exact same logic like so:

```
try {

} catch( InvalidPaymentException | NullPaymentException | NotEnoughPayException $e) {
    // use just one logic for all three exceptions
}
```


Woot! Woot! Really cool. You don't have to handle them with multiple catch blocks again.

13.6 Iterables

A new pseudo-type, `iterable`, has been introduced in PHP 7.1.0. It is similar to `callable`. It will be used for type checking parameter or return values that will be used either in a `foreach` loop or `yield from` statement like so:

```
function performSomeOperation(iterable $iter) {
    foreach($iter as $i) {
        // Do something
    }
}
```

So, you can use `iterable` when you want to accept arrays and objects that implements the `Traversable` Interface. For example, any class that implements `IteratorAggregate` is an iterable.

A new function, `is_iterable` has also been added to determine if a value is iterable . Checkout the RFC here¹.

13.7 Keys Support in list()

The `list()` language construct can now have keys. So it can now handle associative arrays. Check this out:

```
$fruits = ['yellow' => 'mango', 'white' => 'banana' , 'orange' => 'orange'];

list('yellow' => $mango, 'white' => $banana, 'orange' => $orange) = $fruits;

echo $mango.PHP_EOL;
echo $banana.PHP_EOL;
echo $orange.PHP_EOL;

// Result:
mango
banana
orange
```

¹<https://wiki.php.net/rfc/iterable>

13.8 Negative String Offsets Support

Support for negative string offsets has been added to the string functions accepting offsets. So, a negative offset is interpreted as being an offset from the end of the string like so:

```
$name = 'prosper';  
  
echo $name[-2];  
  
// Result:  
e
```

13.9 Conversion of Callables to Closures

In PHP 7.1.0+, you can now convert callables into Closure objects with a new static method, `fromCallable` like so:

```
class Car  
{  
    public function exposeLicense()  
    {  
        return Closure::fromCallable([$this, 'drive']);  
    }  
  
    private function drive($duration)  
    {  
        var_dump($duration);  
    }  
}  
  
$car = (new Car)->exposeLicense();  
$car(25000);  
  
// Result:  
int(25000)
```

13.10 Asynchronous Signal Handling

PHP 7.1.0 has introduced a new function, `pcntl_async_signals()` to enable asynchronous signal handling without using ticks².

```
pcntl_async_signals(true); // turn on async signals

pcntl_signal(SIGHUP, function($sig) {
    echo "SIGHUP\n";
});

posix_kill(posix_getpid(), SIGHUP);
```

13.11 Support for HTTP/2 Server Push

Support for server push has been added to the CURL extension which requires version 7.46 and above.

This can be leveraged through the `curl_multi_setopt` function with the new `CURLMOPT_PUSHFUNCTION` constant. The constants `CURL_PUSHT_OK` and `CURL_PUSH_DENY` have also been added so that the execution of the server push callback can either be approved or denied.

13.12 Better Error Retrieval

Three new functions have been introduced in PHP 7.1.0 to enable errors related to multi and share handles to be retrieved.

```
int curl_multi_errno(resource $mh);
int curl_share_errno(resource $rh);
string curl_share_strerror(int $errno);
```

²<http://php.net/manual/en/control-structures.declare.php#control-structures.declare.ticks>

13.13 Throw Error on Passing too few Function Arguments

In PHP 7.1+, when a function is passed too few arguments, an Error Exception will be thrown like so:

```
function work($price) {}  
}
```

```
work();
```

```
// Result
```

```
Fatal error: Uncaught ArgumentCountError: Too few arguments to function work()
```

Note: The `ext/mcrypt` extension has been deprecated in favour of `OpenSSL`. You can check out for the deprecated features³ and other changes⁴ in 7.1.

³<http://php.net/manual/en/migration71.deprecated.php>

⁴<http://php.net/manual/en/migration71.changed-functions.php>

Chapter 14

Performance Evaluation

PHP 7 runs on the new Zend engine 3.0, thus making your apps see up to 2x faster performance and 50% better memory consumption than PHP 5.6. It also allows you to serve more concurrent users without adding any hardware.

Rasmus Ledorf¹, *Creator of PHP* and inventor of the SQL LIMIT clause did some benchmarking with a few popular PHP projects with the various versions of PHP from PHP 5.4 up until PHP 7.0 and also benchmarked against HHVM 3.6.1.

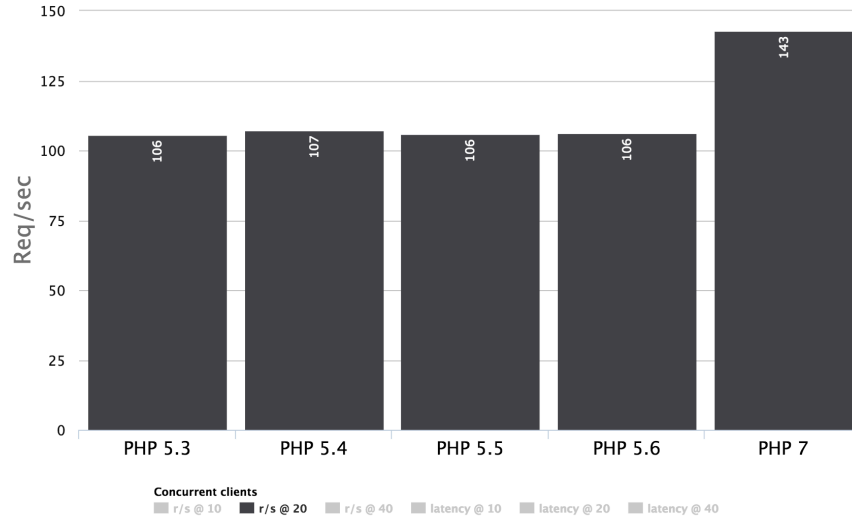
Let's take a good look at the benchmarks. The test box specs Rasmus used are:

- Gigabyte Z87X-UD3H i7-4771 4 cores @ 3.50GHz w/ 16G of Ram @ 1600MHz
- Hyperthreading enabled for a total of 8 virtual cores
- Toshiba THNSNHH256GBST SSD
- Linux debian 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt9-2 (2015-04-13) x86_64 GNU/Linux
- MySQL 5.6.24
- Nginx-1.6.2 + php-fpm for all tests unless indicated otherwise
- Quiet local 100Mbps network
- Siege benchmark tool run from a separate machine

¹<https://twitter.com/rasmus>

ZenCart 1.5.4

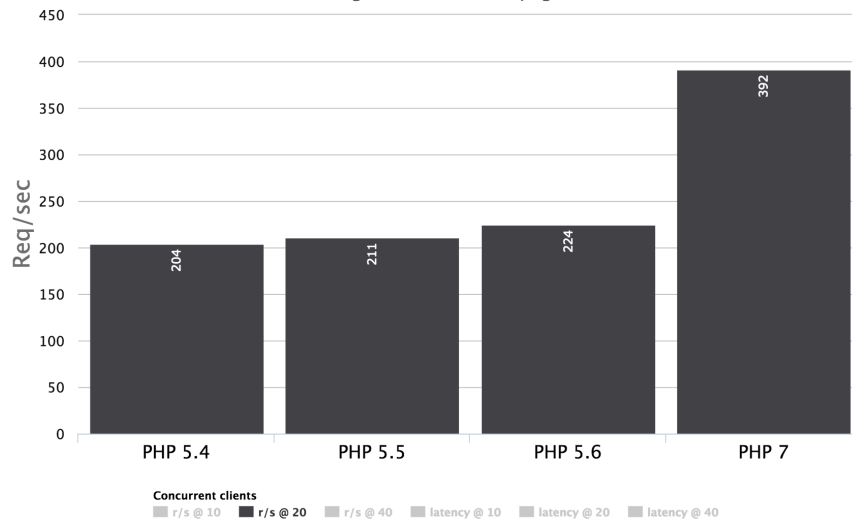
front page with demo data



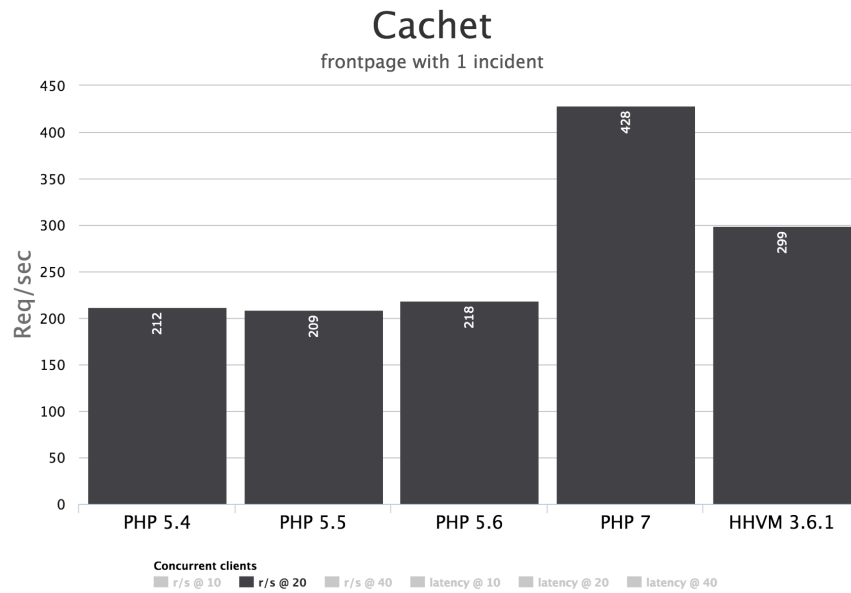
ZenCart 1.5.4

Moodle-2.9-dev

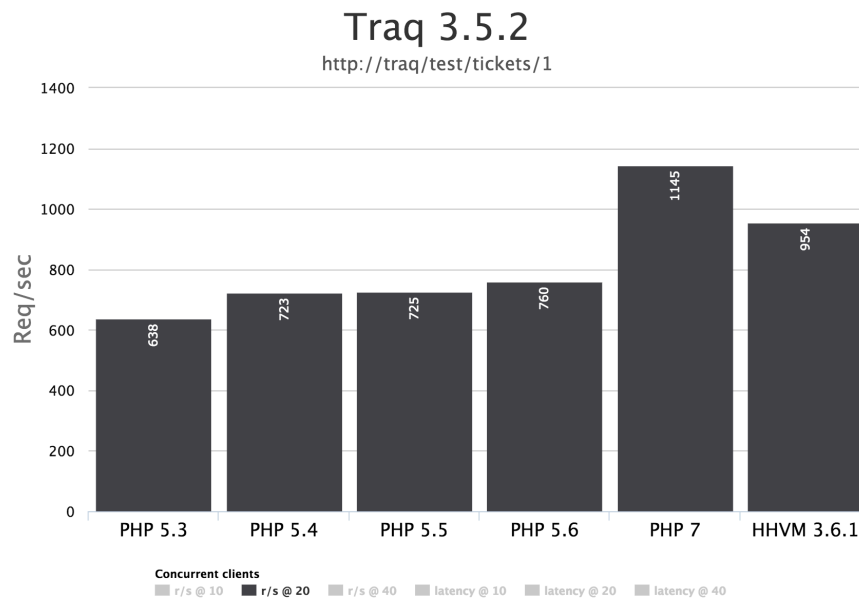
Hitting the default front page



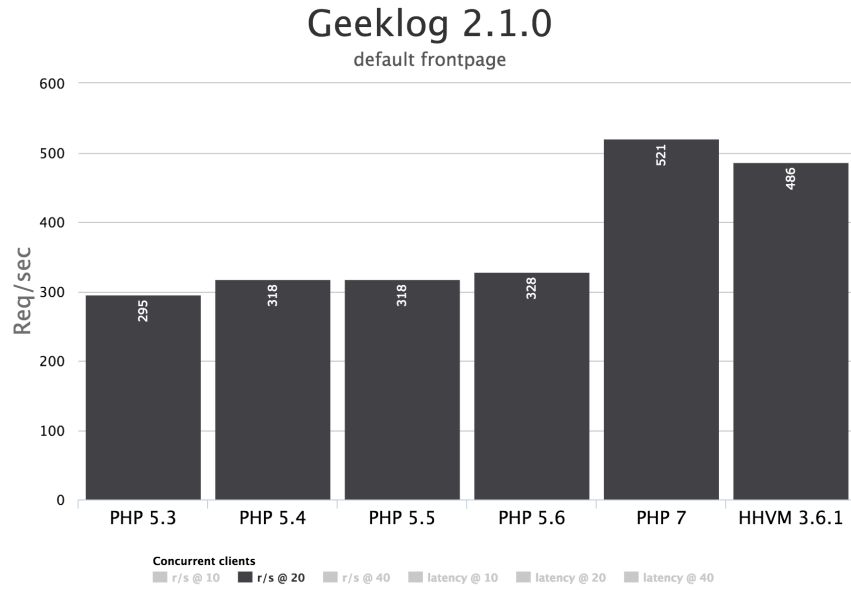
Moodle 2.9-dev



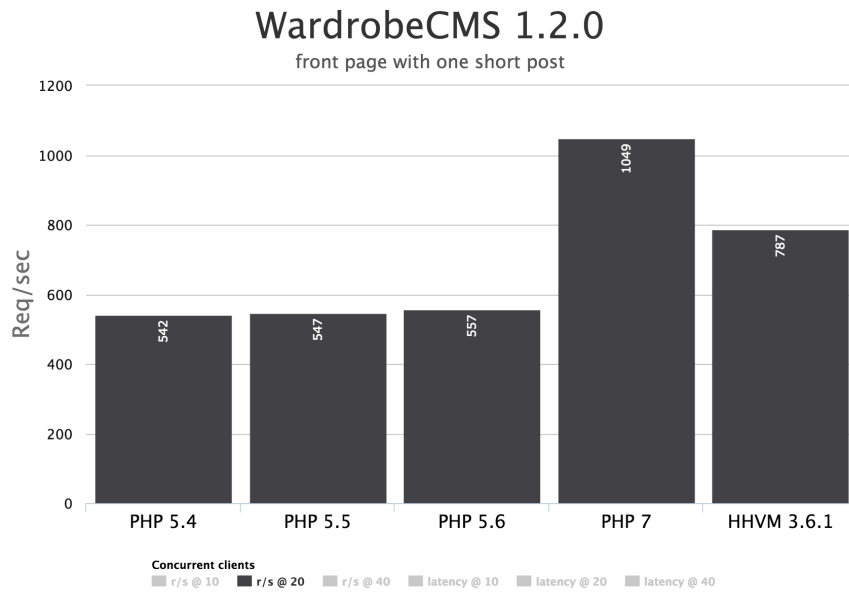
Cachet



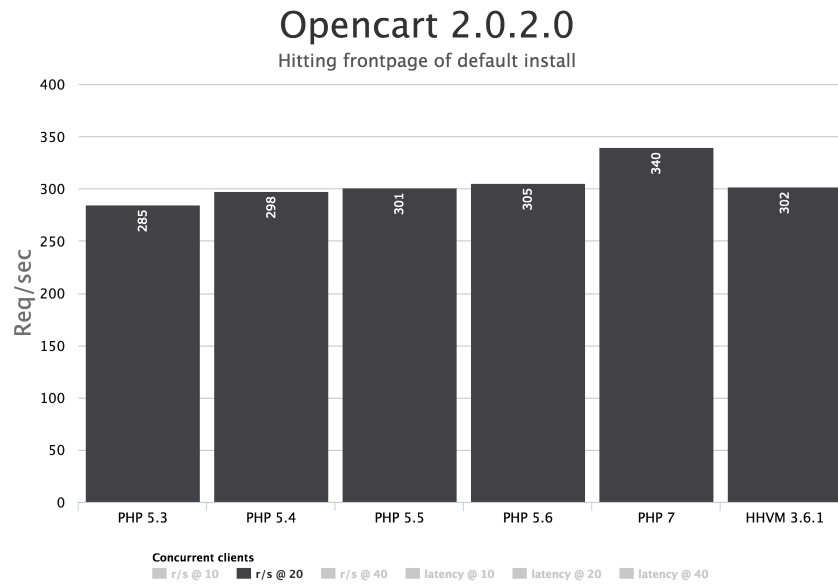
Traq 3.5.2



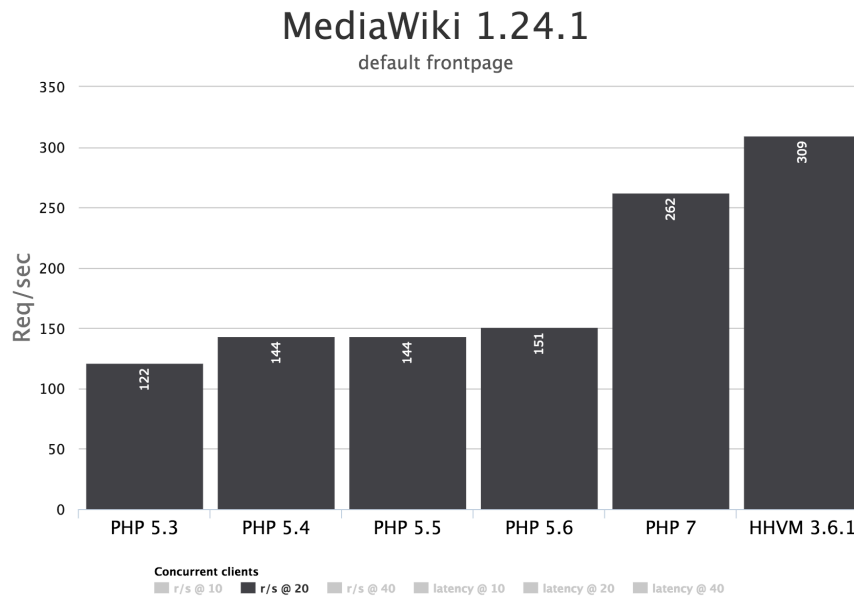
Geeklog 2.1.0



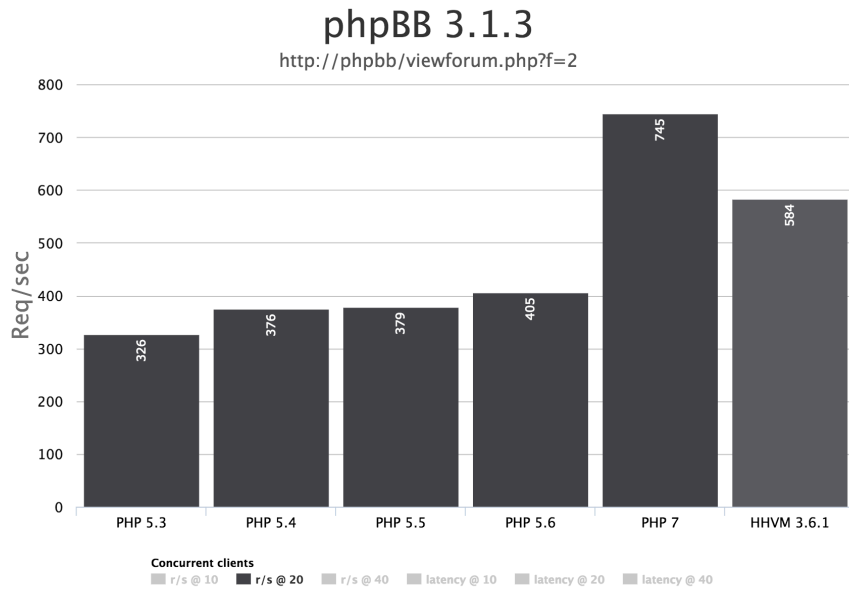
Wardrobe CMS 1.2.0



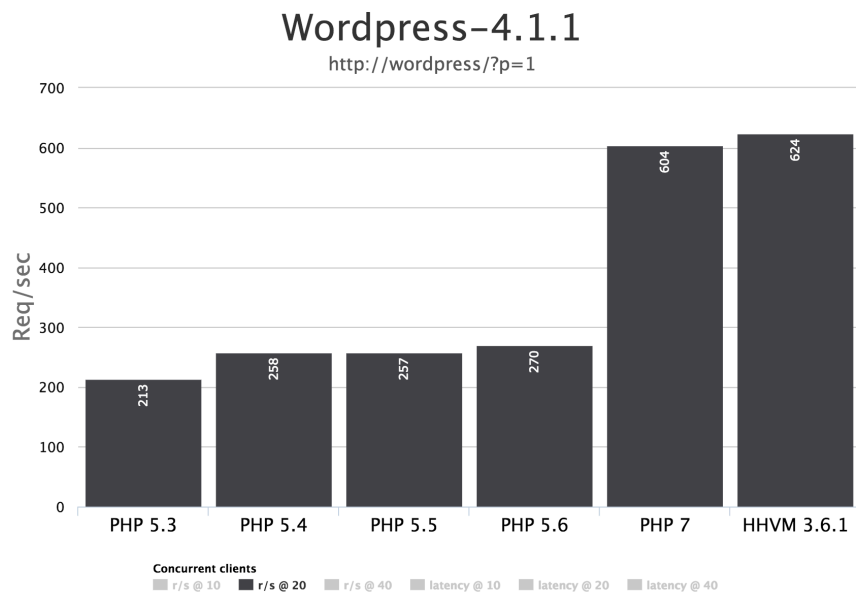
Opencart 2.0.2.0



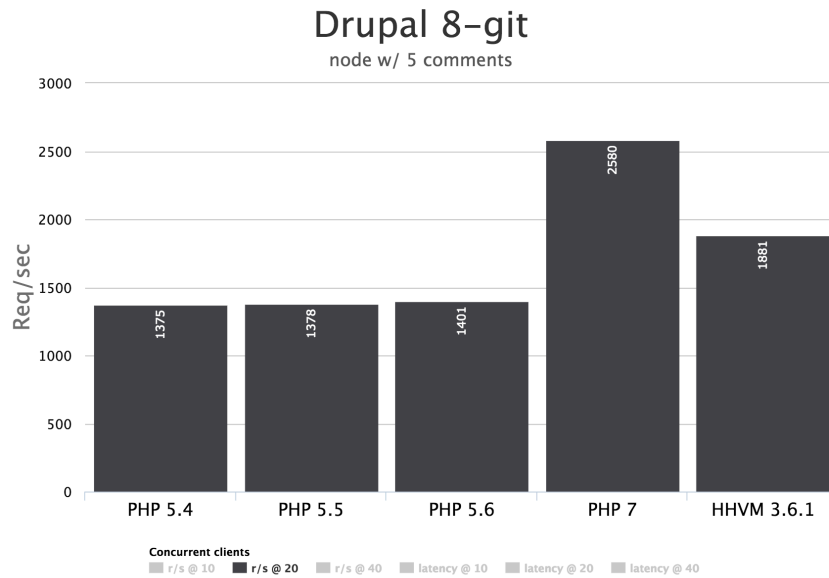
MediaWiki 1.24.1



phpBB 3.1.3



Wordpress 4.1.1



Drupal 8

From the results above, you can see that we can make double the amount of requests in lesser time in PHP 7 than PHP 5.

These specs can be found in the [Speeding Up The Web With PHP 7](#) talk he gave at Fluent Conf, 2015.

Check out the following benchmarks:

- [php7-benchmarks](#)²
- [php7 final version vs hhvm benchmark](#)³
- [hhvm vs php7 performance showdown - Wordpress, Nginx](#)⁴

²<https://github.com/martin-helmich/php7-benchmarks>

³<https://kinsta.com/blog/the-definitive-php-7-final-version-hhvm-benchmark>

⁴<http://blog.wpoven.com/2016/04/14/hhvm-vs-php-7-performance-showdown-wordpress-nginx>

Chapter 15

Conclusion

We have successfully covered how to upgrade your development and server environments from PHP 5 to PHP 7, gone through the features PHP 7 offers and also migrated two apps from PHP 5 to PHP 7.

Woot! Woot! It's been quite a journey highlighting everything PHP 7 has to offer. PHP has grown tremendously over the years from a toy language to a full-blown fast and enterprise language.

The PHP Manual¹ and RFC² documents remain the most complete go-to reference for any of the new PHP 7 features. You can always leverage them for more information.

Thanks for being patient learning all there is to PHP 7. I'm confident that you are now ready to migrate your PHP 5.x apps to PHP 7!

¹<http://php.net/manual/en/index.php>

²<https://wiki.php.net/rfc>