Web Programming

YJ – Aug 2015

Web Application Frameworks

- A web application framework is a set of resources and tools for software developers to build and manage dynamic websites, web applications, web services and web resources.
- A framework is something that prevents you from re-writing this each time you create a website.
- A framework includes templating capabilities for presenting information within a browser, the programming environment for scripting the flow of information and the application programming interfaces (APIs) for accessing underlying data resources.
- Developers can use the framework to define the 'out-of-the-box' content management capabilities, user authentication features, and administrative tools.

Popular Frameworks For Different Languages

- C#, ASP.NET

Microsoft's .NET platform is heavily used in business applications running the Microsoft Windows operating system.

.NET is more or less a closed ecosystem controlled by Microsoft. Widely adopted by enterprises, the documentation and the support services for the language are comprehensive.

The ecosystem of .NET is based on a licensed software model. This means that when third party tools are required, somebody somewhere will need to pay for a license and that eventually will fall upon you. Choosing .NET is likely to cost more than choosing an open source platform.

Many .NET programmers come from an enterprise background; developing internal systems and some may find adapting to the pace and the lifestyle of a start-up environment challenging.

Popular Frameworks For Different Languages

- JAVA

The long commercial life and wide adoption of Java has created a robust ecosystem of documentation, libraries and frameworks many of which are aimed at e-commerce, security, and complex transactional architectures.

Spring MVC

The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that dispatches requests to handlers, with configurable handler mappings, view resolution, locale, time zone and theme resolution as well as support for uploading files.

Its features:

> Clear separation of roles. Each role can be fulfilled by a specialized object.

> Powerful and straightforward configuration of both framework and application classes

> Adaptability, non-intrusiveness, and flexibility.

> Reusable business code, no need for duplication.

Popular Frameworks For Different Languages

- Python

Python is an open source interpretive language that has been embraced by many in the scientific community for its ease of learning and large set of scientific libraries.

Django

Django's primary goal is to ease the creation of complex, database-driven websites.

Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspectionand configured via admin models.

One caution is that Python is notoriously difficult to scale across multiple cores on a single machine. Developers must still design programs specifically for independence in order to run on both Windows and Linux platforms.

Popular Frameworks For Different Languages

- Ruby

  Ruby is the successful combination of: Smalltalk's conceptual elegance; Python's ease of use and learning, and Perl's pragmatism

  Ruby is a High Level Programming Language; Interpreted like Perl, Python, Tcl/TK.; Object-Oriented Like Smalltalk, Eiffel, Ada, Java.

  Ruby on Rails

  An extremely productive web-application framework.

  You could develop a web application at least ten times faster with Rails than you could with a typical Java framework.

  An open source Ruby framework for developing database-backed web applications.

  Your code and database schema are the configuration!

  No compilation phase required.

  One valid concern is that Ruby does not scale up well on the server side for large numbers of requests to the application.

Popular Frameworks For Different Languages

- JavaScript

There are many JavaScript frameworks available. JavaScript frameworks are an increasingly popular way to build web applications.

jQuery

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML.

jQuery provides a simple interface for the underlying JavaScript. It's easier for many users to learn jQuery first, then dive into the nitty-gritty JavaScript details later.

jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications.

```
$(document).ready(function(){
    $('img').click ( function() {
    // handle the click event on any img element in the page
    });
});
```

Popular Frameworks For Different Languages

- JavaScript

AngularJS

In Angular, both the Model and the View can update the data, thus the "two-way" descriptor. Two-way data binding is a much loved feature of AngularJS that describes the condition where data is bound to an HTML element in the View and that element has the ability to both update and display that data.

Another popular feature is directives, which allow developers to extend HTML by attaching special behaviors to parts of the DOM.

Dependency injection is another great feature of Angular that allows developers to easily include services in their modules.

Backbone.js

As one of the oldest of the JavaScript frameworks. Backbone may be better suited for more advanced JavaScript developers. It's minimalism can be both a blessing and a challenge depending on the experience level of the developer working with it.

Node.js

It is actually a runtime environment runs on server-side and handles all the server-side logic.

Popular Frameworks For Different Languages

- JavaScript

jQuery

```
$('#greet-form input.user-name').on('value', function() {
      $('#greet-form div.user-name').text('Hello ' + this.val() + '!'); }
);
```

AngularJS

```
<input ng-model="user.name" type="text" />
Hello {{user.name}}!


<ul>
    <li ng-repeat="framework in frameworks" title="{{framework.description}}">
        {{framework.name}}
    </li>
</ul>
```

Backbone.js

```
<ul>
    <% _.each(frameworks, function(framework) { %>
          <li title="<%- framework.description %>">
                <%- framework.name %>
          </li> <% }); %>
 </ul>
```

Popular Frameworks For Different Languages

- CSS

Bootstrap

Bootstrap is definitely the most popular and widely used framework, nowadays. It's a beautiful, intuitive and powerful web design kit for creating cross browser, consistent and good looking interfaces. It offers many of the popular UI components with a plain-yet-elegant style, a grid system and JavaScript plugins for common scenarios.

- Scaffolding – global styles, responsive 12-column grids and layouts. Bear in mind that Bootstrap doesn't include responsive features by default. If your design needs to be responsive you have to enable this functionality manually.
- Base CSS – this includes fundamental HTML elements like tables, forms, buttons, and images, styled and enhanced with extensible classes.
- Components – collection of reusable components like dropdowns, button groups, navigation controls (tabs, pills, lists, breadcrumbs, pagination), thumbnails, progress bars, media objects, and more.
- JavaScript – jQuery plugins which bring the above components to life, plus transitions, modals, tool tips, popovers, scrollspy (for automatically updating nav targets based on scroll position), carousel, typeahead (a fast and fully-featured autocomplete library), affix navigation, and more.

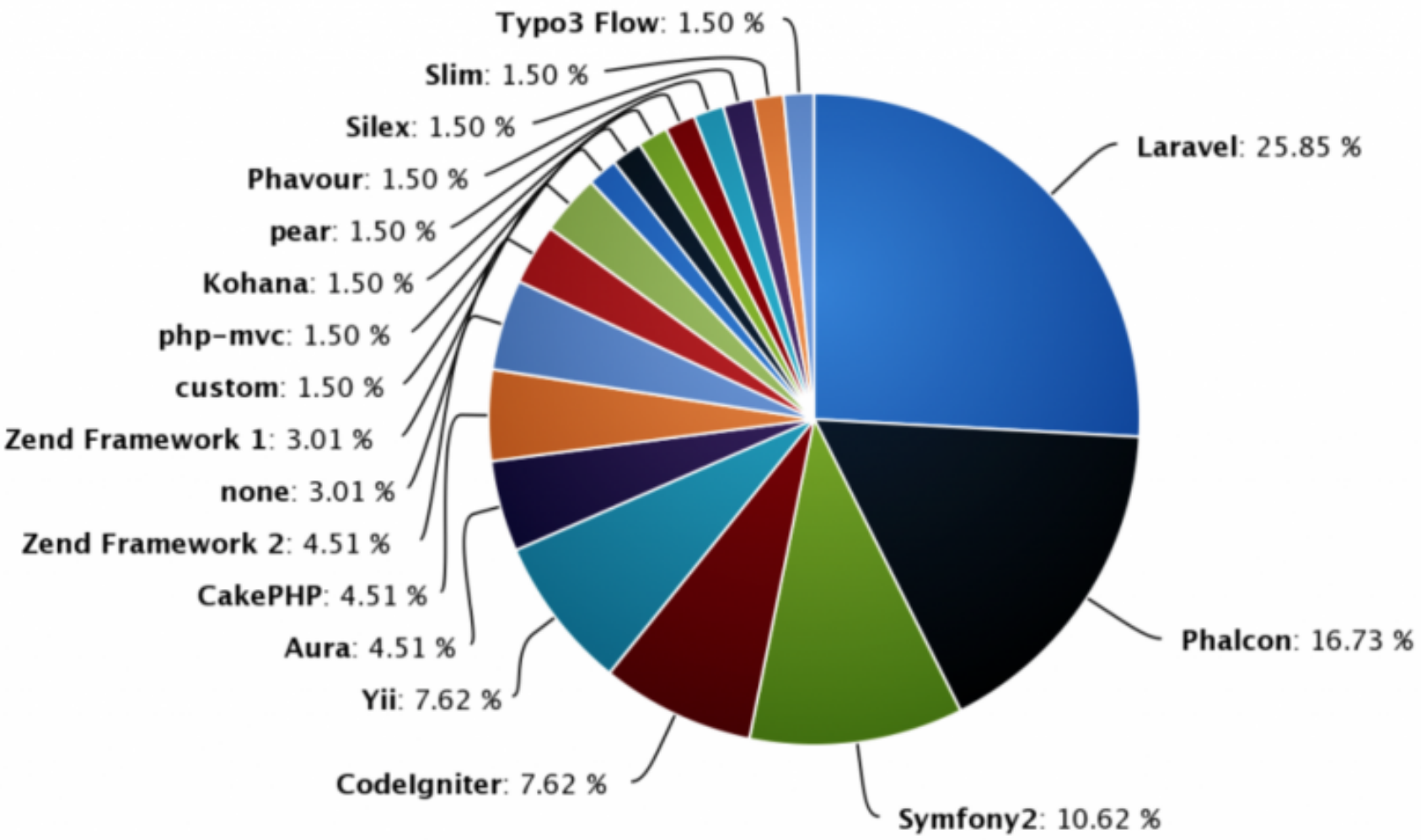
Popular Frameworks For Different Languages

- PHP

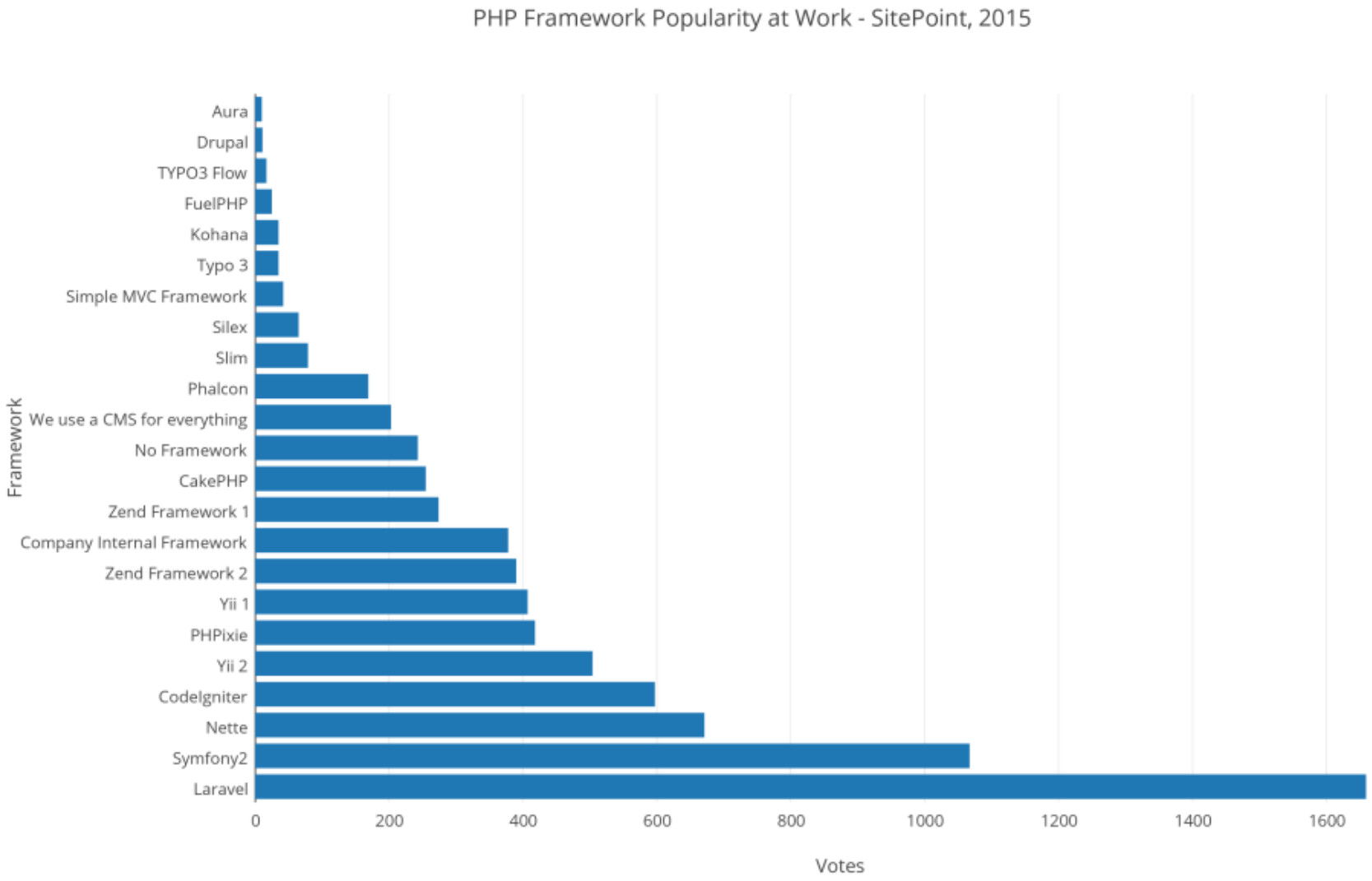
  PHP itself is a very weak and dangerous language. But its easy to learn for beginners. PHP is very special as you can learn the language quickly but it takes a long time to become a good PHP developer. Mostly you you have to see a lot of to know how to use PHP more professional. These frameworks out there try to handle and copy concepts from other better OOP programing languages, frameworks and paradigms.

  1. Laravel
  2. CodeIgniter
  3. CakePHP
  4. Symfony 2
  5. Zend Framework 2
  6. Phalcon
  7. Yii 2
  8. Aura
  9. PHP-MVC
  10. Kohana
  11. FuelPHP
  12. Slim

PHP Framework



PHP Framework



Laravel

Laravel has shaken up the PHP community in a big way just in a couple of years ago. It has been generating a lot of buzz with the promise of making web applications fast and simple to create. Using Laravel, you can build and maintain high-quality web applications with minimal fuss.

- **Simple** - Laravel's functionalities are easy to understand and implement. If you enjoy how simple and easy CodeIgniter is, then you'll love Laravel
- **Elegant** - most of Laravel's functions work seamlessly with very little configuration, relying on industry-standard conventions to lessen code-bloat
- **Well-documented** - Laravel's documentation is complete and always up-to-date. The framework creator makes it a point to update the documentation before releasing a new version, ensuring that people who are learning the framework always have the latest documentation.

Laravel

Installation

Refer to http://laravel.com/docs/5.1

1. Install Composer
2. Install laravel
3. Create project
4. Configuration (.env for database setting)

Laravel

Database

Laravel has a multitude of functions that differentiates it from the rest of the pack. For this section, I only demonstrate two features as foretaste of this popular framework.

- **Database migrations** - makes it easy to keep your database schema up-to-date with other team member's changes.

  Migrations are like version control for your database, allowing a team to easily modify and share the application's database schema. Migrations are typically paired with Laravel's schema builder to easily build your application's database schema.

```php
public function up()
{
    Schema::create(
        'flights',
        function ( Blueprint $table ) {
            $table->increments( 'id' );
            $table->string( 'name' );
            $table->string( 'airline' );
            $table->timestamps();
        }
    );
}
```

```php
public function down()
{
    Schema::drop('flights');
}
```

Laravel

After setup database, we can create models and corresponding tables

Create Models (/app/)

php artisan make:model Article

php artisan make:model Page

Create Tables (/database/migrations/)

php artisan make:migration create_articles_table

php artisan make:migration create_pages_table

```
public function up()
   {
      Schema::create('articles', function(Blueprint $table)
{
   $table->increments('id');
   $table->string('title');
   $table->string('slug')->nullable();
   $table->text('body')->nullable();
   $table->string('image')->nullable();
   $table->integer('user_id');
   $table->timestamps();
});
   }
public function down()
{
   Schema::drop('articles');
}
```

Laravel

Fill data into tables (/database/seeds/PageTableSeeder.php)

```
public function run()
{
  DB::table('pages')->delete();

  for ($i=0; $i < 10; $i++) {
    Page::create([
      'title'   => 'Title '.$i,
      'slug'    => 'first-page',
      'body'    => 'Body '.$i,
      'user_id' => 1,
    ]);
  }
}
```

Laravel

Run a Query

```php
<?php

namespace App\Http\Controllers;

use DB;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show a list of all of the application's users.
     *
     * @return Response
     */
    public function index()
    {
        $users = DB::select('select * from users where name = `somename`');

        return view('user.index', ['users' => $users]);
    }
}
```

Blade (template) file

```
@foreach ($users as $user)
    <div class="page">
        <h4>{{ $user->name }}</h4>
    </div>
     @endforeach
```

Laravel

Routing

```
// show a static view for the home page (app/views/home.blade.php)
Route::get('/', function()
    {
        return view('home');
    });

// about page (app/views/about.blade.php)
Route::get('about', function()
    {
        return view('about');
    });

Route::get('blog/{category?}', function($category = null)
    {
        // get all the blog stuff from database
        // if a category was passed, use that
        // if no category, get all posts
        if ($category)
            $posts = Post::where('category', '=', $category);
        else
            $posts = Post::all();

        // show the view with blog posts (app/views/blog.blade.php)
        return view('blog')
                ->with('posts', $posts);
    });
```

Laravel

❖ Route Configuration (/app/Http/routes.php)

```
// 'prefix' => 'admin' means http://laravel:8888/admin
// 'namespace' => 'Admin' means file location is \App\Http\Controllers\Admin\AdminHomeController@index

Route::group(['prefix' => 'admin', 'namespace' => 'Admin'], function()
    {
        Route::get('/', 'AdminHomeController@index');
        Route::resource('pages', 'PagesController');
    });
```

Laravel

❖ Route resource

```
Route::resource('pages', 'PagesController');
```

Verb	Path	Action	Route Name	Example
GET	/pages	index	pages.index	`href="{{ URL('admin/pages') }}"`
GET	/pages/create	create	pages.create	`href="{{ URL('admin/pages/create') }}"`
POST	/pages	store	pages.store	`<form action="{{ URL('admin/pages') }}" method="POST">`
GET	/pages/{pages}	show	pages.show	`href="{{ URL('admin/pages/'.$page->id) }}"`
GET	/pages/{pages}/edit	edit	pages.edit	`href="{{ URL('admin/pages/'.$page->id.'/edit') }}"`
PUT/PATCH	/pages/{pages}	update	pages.update	`<form action="{{ URL('admin/pages/'.$page->id) }}" method="POST">` `<input name="_method" type="hidden" value="PUT">`
DELETE	/pages/{pages}	destroy	pages.destroy	`<form action="{{ URL('admin/pages/'.$page->id) }}" method="POST">` `<input name="_method" type="hidden" value="DELETE">`

Laravel

Controller

❖ Create Controller

php artisan make:controller Admin/AdminHomeController

```
class AdminHomeController extends Controller
{

    public function index()
    {
        return view('AdminHome')->withPages(Page::all());
    }
```

php artisan make:controller Admin/PagesController

```
.....
// Show the form for editing the specified resource.
public function edit($id)
{
  return view('admin.pages.edit')->withPage(Page::find($id));
}
// Update the specified resource in storage.
public function update(Request $request,$id)
{
  $this->validate($request, [
    'title' => 'required|unique:pages,title,'.$id.'|max:255',
    'body' => 'required',
  ]);

  $page = Page::find($id);
  $page->title = Input::get('title');
  $page->body = Input::get('body');
  $page->user_id = 1;//Auth::user()->id;

  if ($page->save()) {
    return Redirect::to('admin');
  } else {
    return Redirect::back()->withInput()->withErrors('Save Error!');
  }
}
.....
```

Laravel

View

Create /resources/views/AdminHome.blade.php **which is a template file**

```
@foreach ($pages as $page)
 <hr>
 <div class="page">
  <h4>{{ $page->title }}</h4>
  <div class="content">
   <p>
    {{ $page->body }}
   </p>
  </div>
 </div>
 <a href="{{ URL('admin/pages/'.$page->id.'/edit') }}" class="btn btn-success">Edit</a>

 <form action="{{ URL('admin/pages/'.$page->id) }}" method="POST" style="display: inline;">
  <input name="_method" type="hidden" value="DELETE">
  <input type="hidden" name="_token" value="{{ csrf_token() }}">
  <button type="submit" class="btn btn-danger">Delete</button>
 </form>
@endforeach
```

Laravel

Demo

Create a simple blog system with admin panel and user authentication functions under Laravel5 framework.

Basic functions includes:

Homepage
Login/Register
Articles(blog)
Normal Pages
Comments
Admin Panel(edit page, article, comments)

You can follow the instruction on this week's tutorial to complete this demo by yourself.
To gain a deeper understanding of Laravel, please read http://laravel.com/docs/5.1