WEB 306 – PHP: Databases and Framework

Day 14 – Wordpress

1. Wordpress

1.1 About Wordpress

Wordpress is a CMS (content management system) which is written in PHP. It was created by Matt Mullenweg in 2003 to be a simple blogging platform but grew to be the most popular CMS in the world. It is estimated that Wordpress is used on 30% of websites on the internet.

Wordpress Resources

1. Wordpress Website
2. Download Wordpress
3. Wordpress Codex
4. WP Beginner
5. Scotch.io
6. Stack Overflow
7. Themosis Website

1.2 Wordpress.org vs. Wordpress.com

There are two versions of Wordpress. One is a free version which can be installed on your own web host and can be downloaded from wordpress.org. The other is a site for hosting Wordpress blogs on wordpress.com. Setting up a blog on **wordpress.com** is free if you don't want to customize many aspects of the blog. In order to customize practially anything about your blog you must pay a monthly fee which is actually higher than the average web host's fees and you still won't have as much control. For this reason, you should really never use **wordpress.com** because you could just use the **wordpress.org** version and have more control for a smaller price.

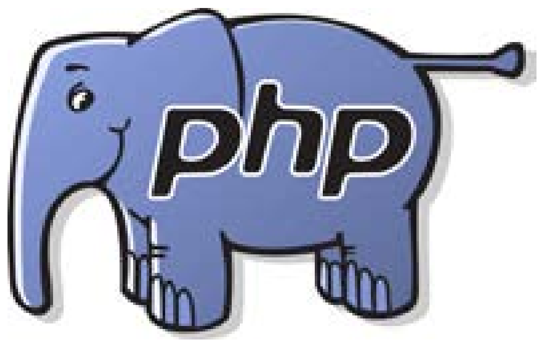
1.3 Installing Wordpress

To install an average Wordpress site, just go to wordpress.org/download and download the zip file. You would then put this file in your `xampp/htdocs` folder or in your virtual host folder and extract it. In most cases you can simply install Wordpress using the famous 5-minute install by visiting the directory you extracted your files to in the browser and following the steps. However, if this process fails for some reason or if you are using a host which does not allow for this type of install then you need to open the wp-config.php file and connect to the database.

```
define('DB_NAME', 'coolsite');
define('DB_USER', 'root');
define('DB_PASSWORD', '');
define('DB_HOST', ' localhost');
```

1.4 Wordpress Themes

Wordpress allows you to change the design of a site by changing the "theme" or by installing a new theme from the themes library. As a PHP developer it would normally be your job to either customize a theme or to design a theme from scratch rather than simply installing a premade one.



1.5 Wordpress Plugins

Wordpress plugins are used to add additional functionality to a Wordpress site. You can install new plugins using the Wordpress plugin library. As a PHP developer you may also be required to create new Wordpress plugins.

PHP Resources

1. PHP Website
2. XAMPP
3. Documentation
4. W3Schools
5. CSS Tricks
6. Scotch.io
7. Stack Overflow

2. Basic Child Themes

The simplest way to customize a Wordpress theme is by using a child theme which inherits the design and functionality of an existing theme which is used as a parent theme but allows you to overwrite it without breaking it. I like to use a theme called **WP Forge** to create child themes because the code is somewhat simpler to edit than many Wordpress themes which can be very complex.

There are two main files which are needed to create a child theme using Wordpress: a `style.css` file and a `functions.php` file, both of which must be in the root directory of the child theme. The `style.css` file may have all of the following settings which are specified in a comment at the top of the file but it must have a **Theme Name**, a **Template**, which specifies the name of the parent theme folder and a **Text Domain**, which specified the name of the child theme folder.

```
/*
Theme Name: Cool Theme
Template: wp-forge
Text Domain: cooltheme
Theme URI: http://themeawesome.com/wordpress-child-theme
Description: A theme for A Cool Site
Author: Cool Guy
Author URI: http://themeawesome.com/
Version: 6.4.3
License: GNU General Public License v2 or later
License URI: http://www.gnu.org/licenses/gpl-2.0.html
Tags: light, gray, white, one-column, two-columns, right-sidebar,
custom-background, custom-header, custom-menu, editor-style, featured-
images, full-width-template, microformats, post-formats, rtl-language-
support, sticky-post, translation-ready

Start adding your theme specific styles below.
*/
```

A child theme must also have a `functions.php` file which is used to add links to assets such as CSS and JavaScript files and to customize navigation menus. We will make a function to add our CSS links to the site.

```
  function cool_enqueue_styles() {
    $parent_style = 'wpforge';
```

Get rid of the customizer CSS - It just gets in the way. Both of these functions are needed to totally remove a CSS file.

```
    wp_dequeue_style('customizer');
    wp_deregister_style('customizer');
```

Deactivate the parent CSS for now or weird things happen.

```
    wp_dequeue_style($parent_style);
    wp_deregister_style($parent_style);
```

Requeue our parent CSS.

```
    wp_enqueue_style($parent_style, get_template_directory_uri() .
        '/style.css');
```

Queue our custom CSS under the parent CSS to override it.

```
    wp_enqueue_style('cooltheme', get_stylesheet_directory_uri() .
        '/style.css',
```

This is an array of CSS files to override.

```
        array( $parent_style ),
```

This inputs the version number specfied in the CSS file.

```
        wp_get_theme()->get('Version')
    );
  }
```

Call our `cool_enqueue_styles()` function and give it really high priority using `9999`.

```
  add_action( 'wp_enqueue_scripts', 'cool_enqueue_styles', 9999);
```

By default JavaScript is put in the `<head>` tags. This function just moves them to the bottom of the page, before the closing `</body>` tag.

```
  function remove_head_scripts() {
    remove_action('wp_head', 'wp_print_scripts');
    remove_action('wp_head', 'wp_print_head_scripts', 9);
    remove_action('wp_head', 'wp_enqueue_scripts', 5);

    add_action('wp_footer', 'wp_print_scripts', 5);
    add_action('wp_footer', 'wp_print_head_scripts', 5);
    add_action('wp_footer', 'wp_enqueue_scripts', 5);
  }

  add_action( 'wp_enqueue_scripts', 'remove_head_scripts');
```

Make a function to add our custom JavaScript to the site.

```
  function cool_scripts() {
```

Queue our JS file at the bottom of the page.

```
    wp_enqueue_script('cooljs', get_stylesheet_directory_uri() .
         '/js/bundle.js',
```

An array of dependencies for our JS file, in this case, jQuery, then the version number and a boolean value for whether this script goes in the footer.

```
        array('jquery'), '1', true
    );
  }

  add_action( 'wp_enqueue_scripts', 'cool_scripts', 9999);
```

3. Themosis

Child themes are good for clients with basic needs or small budgets. However, creating custom Wordpress themes from scratch will result in a higher level of performance and customization. The process of creating Wordpress themes is a bit arduous and dated though. Because of this a group of developers created an MVC framework called Themosis, which was inspired by Laravel and which allows developers to create Wordpress themes using many of the same components as Laravel.

To install Themosis, use the following Composer command with the name of your project at the end.

```
  composer create-project themosis/
  themosis dogblog
```

Like Laravel, Themosis uses a `.env` file. To use it, rename it from `.env.local` to just `.env`. Then open `environments.php` and set this file to return false.

```
  return false;
```

Then update the `.env` file with your database credentials.

```
  DB_NAME=dogblog
  DB_USER=root
  DB_PASSWORD=
  DB_HOST=127.0.0.1
  WP_HOME=http://dogblog.local
  APP_URL=http://dogblog.local/cms
```

Laravel Resources

1. Laravel Website
2. Laravel Documentation
3. Laracasts
4. Laravel News
5. Scotch.io
6. Stack Overflow

4. Models

4.1 Post Model

Themosis uses models, like Laravel and can make use of Laravel's Eloquent ORM interface. By default they give us a `Post` model which needs to have three properties added to it.

```
namespace Theme\Models;

use Illuminate\Database\Eloquent\Model;


/**
 * Class Post.
 * Help you retrieve data from your $prefix_posts table.
 *
 * @package Theme\Models
 */
class Post extends Model {
   /**
    * The post model class fetch its records
    * from the "wp_posts" table.
    * The DB prefix is already defined for you.
    */
   protected $table = 'posts';

   /**
    * Eloquent needs to know which column in
    * the wp_posts table is the primary key.
    */
   protected $primaryKey = 'ID';
   /**
    * Eloquent needs to know which column in
    * the wp_posts table is the primary key.
    */
   protected $timestamps = false;
}
```

4.2 Dog Model

Because Wordpress stores most data in posts, most models would need to collect data from the Post model. As a result, if we wanted to use a different type of data, such as a dog, it would be simplest to make it a child class of the `Post` class. This class will use the `Illuminate\Database\Eloquent\Builder` namespace, access the `Model` class's boot method and then the `addGlobalScope()` method which will use the `Builder` class and the `where()` in order to query the posts table for posts with a particular post type, in this case "dogs".

```
namespace Theme\Models;
```

```
use Theme\Models\Post;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Builder;

class Dog extends Post {
   protected static function boot() {
        parent::boot();

        static::addGlobalScope('dogs', function(Builder $builder)
        {
             $builder->where('post_type', 'dogs');
        });
   }
}
```

4.3 Page Model

Wordpress also allows for the creation of pages such as an About page. The model for a Page would work the same way as the Dog model except it would query for a post type of "page".

```
namespace Theme\Models;

use Theme\Models\Post;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Builder;

class Page extends Post {
   protected static function boot() {
        parent::boot()

        static::addGlobalScope('page', function(Builder $builder)
   {
             $builder->where('post_type', 'page');
        });
   }
}
```

5. Routes

Themosis also uses Laravel's route syntax. However, the route names can mean slightly different things. The "home" address always points to the root directory. The "single" address always points to a single blog post. The "page" address always points to any pages which are created with Wordpress. Aside from those and a few other preset addresses, we can also create custom routes the same way we would in Laravel. Then the second parameter of the route would either return a view or point to a controller, just like in Laravel.

```
Route::get('home', 'Posts@index');
Route::get('single', 'Posts@show');
Route::get('dogs', 'Dogs@index');
```

```
Route::get('dogs/{id}', 'Dogs@show');
Route::get('page', 'Page@show');
```

6. Controllers

6.1 Posts Controller

Themosis also uses controllers, like Laravel. Controllers in Themosis would generally be much simpler though. They could just return a view.

```
namespace Theme\Controllers;

use Theme\Models\Post;
use Themosis\Route\BaseController;

class Posts extends BaseController {
   public function index() {
        return view('home');
   }

   public function show() {
        return view('single');
   }
}
```

6.2 Dogs Controller

Like in Laravel, you could also use a controller to query data using the Eloquent ORM and then pass it to the view using an array of variables.

```
namespace Theme\Controllers;

use Theme\Models\Dog;
use Themosis\Route\BaseController;

class Dog extends BaseController {
   public function index() {

        $dogs = Dog::where('post_status', 'publish')->get();

        return view('index', [
             'dogs' => $dogs,
        ]);
   }

   public function show($id) {
        $dog = Dog::find($id);

        return view('index', [
            'dog' => $dog,
        ]);
    }
}
```

6.3 Page Controller

```
namespace Theme\Controllers;

use Theme\Models\Post;
use Themosis\Route\BaseController;

class Page extends BaseController {
    public function show() {
        return view('page');
    }
}
```

7. Custom Post Types, Fields and Taxonomies

One of the big advantages of using Themosis is that it makes it easy to create custom post types and custom fields types. To do so, simply use the `PostType` class with the `make()` and `set()` methods. It is necessary to set several names for post types to account for capitalization and plurals.

```
/**
 * Define your theme custom code.
 */

$dogs = PostType::make('dogs', 'Dogs', 'Dog')->set();
```

The "slug" is used to identify a post type using an all lowercase name with underscores.

```
$slug = $dogs->set('name');
```

A metabox is used to group custom fields together and apply them to a post type.

```
Metabox::make('Info', $slug)->set([
    Field::text('nicknames', ['title' => 'Nicknames:']),
    Field::textarea('fav_treats', ['title' => 'Favourite Treats:']),
    Field::checkbox('good', 'good', ['title' => 'Good Dog?']),
    Field::select('size', [
        'XS' => 'Toy',
        'SM' => 'Small'
        'MD' => 'Medium'
        'LG' => 'Large'
    ], ['title' => 'Size:']),
    Field::color('fur_color', ['title' => 'Fur Colour:'])
]);
```

Taxonomies are used to categorize posts. The default taxonomy which is associated with all posts is "categories". For our dog posts, we could create a custom taxonomy called "breeds" using the `Taxonomy` class.

```
Taxonomy::make('breeds', $slug, 'Breeds', 'Breed')->set();
```

8. Blade Templates

Like Laravel, Themosis allows us to use Blade templates instead of the default PHP templating engine. Themosis also allows you to use another popular templating engine called Twig. We will use Blade due to its familiarity. Themosis provides some new Blade statements which are specific to Wordpress. These include `@wp_head` and `@wp_foot` which inserts the Wordpress data which is defined in the `functions.php` file and which goes in the `<head>` tags or footer of the page.

```
<!DOCTYPE html>
<html lang=”en”>
  <head>
       <meta charset=”UTF-8”>
       <meta name=”viewport” content=”width=device-width,
            initial-scale=1.0”>

       @wp_head
  </head>
  <body>
       <header>
            <h1><a href="{{ home_url() }}">{{ bloginfo( 'name' ) }}
                 </a></h1>
       </header>

       @yield('content')

       @wp_foot
  </body>
</html>
```

Themosis also provides us with a new type of loop called `@query` which allows us to easily query Wordpress posts and loop through them in our templates. You can access data from a Wordpress post inside of these loops using the `Loop` class and its associated methods.

```
@extends('layouts.base')

@section('content')
@query(['post_type' => 'posts', 'posts_per_page' => 3])
  <article {!! Loop::postClass() !!}>
       <h2><a href="{!! Loop::link() !!}">{!! Loop::title() !!}
            </a></h2>
       <h2>Posted by {!! Loop::author() !!}</h2>
       <time>Posted on {!! Loop::date() !!}</time>
       {!! Loop::content() !!}
  </article>
```

```
@endquery
@endsection
```

You can also simply use the `@loop` statement if you do not have any data to query for and simply want to loop through all of the posts.

```
@extends('layouts.base')

@section('content')
@loop
   <article {!! Loop::postClass() !!}>
        <h2><a href="{!! Loop::link() !!}">{!! Loop::title() !!}
             </a></h2>
        <h2>Posted by {!! Loop::author() !!}</h2>
        <time>Posted on {!! Loop::date() !!}</time>
        {!! Loop::content() !!}
   </article>
@endloop
@endsection
```

These loops can not as easily be used to get more complex types of data such as custom fields though. To do so, you could use a combination of Laravel's Blade loops and Wordpress Codex functions.

```
@extends('layouts.base')

@section('content')
@foreach ($dogs as $number => $dogs)
   <article class="dog">
        <h2><a href="{{ home_url('/dogs/' . $dog->ID) }}">
             {{ $dog->post_title }}</a></h2>
        <h2>Posted by {{ get_the_author_meta('display_name',
             $dog->post_author) }}</h2>
        <time>Posted on {{ get_the_date('l, F j, Y', $dog->ID) }}
             </time>
        {!! $dog->post_content !!}

        <ol>
             <li>Breed: {{ get_post_custom($dog->ID)['breed'][0] }}
                  </li>
             <li>Nicknames:
                  {{ get_post_custom($dog->ID)['nicknames'][0] }}</li>
             <li>Size:
                  {{ get_post_custom($dog->ID)['size'][0] }}</li>
             @if (get_post_custom($dog->ID)['good'][0])
             <li>Good Dog</li>
             @else
             <li>Bad Dog</li>
             @endif
             <li>Fur Colour: <div style="width: 100px; height: 100px;
                  background:
                        {{ get_post_custom($dog->ID)['fur_color'][0] }};">
                        </div>
                   </li>
              </ol>
         </article>
  @endforeach
  @endsection
```