TASK

WordPress - Plugins

Visit our website

Introduction

WORDPRESS WITH A JETPACK

In a previous task, we discussed the value Laravel brings to PHP, and in this task we'll discuss the value PHP can bring to WordPress (WP). Besides the obvious (WP is written in PHP), WP allows you to write custom plugins for it in PHP where you can define just about any application logic you want to. Let's dive in.

GETTING PLUGGED IN

To get started developing a WordPress plugin, you need a local copy of WP to run your tests on. Luckily, you already set this up in the previous task. Open your XAMPP Control Panel, start up your Apache and MySQL server and click the Apache "Admin" button. Browse to **http://localhost/wordpress** to check that it's working.

Open the project in an IDE of your choice, and in the directory `wp-content/plugins`, create a file called `coolness.php`. This file will contain all the scripting for our new plugin, which will add the functionality of keeping track of view counts across your posts — their coolness, if you will.

Add the following comment to the top of the file to describe your plugin. This is a requirement in order for WP to recognise your plugin and allow its installation. Feel free to use your own author name and website.

```
<?php
/*
Plugin Name: Coolness
Plugin URI: https://www.hyperiondev.com/
Description: A plugin that keeps track of post views
Version: 1.0
Author: Harold Hyperion
Author URI: https://www.hyperiondev.com/
Licence: UNLICENCED
*/
```

HOOKING INTO ACTIONS

The typical way of running your own code in WP is by registering lifecycle hooks. While WP runs, it makes many internal calls as users interact with the site. From button click events, to page rendering, to theme updates — just about every event you can think of fires inside WP core with appropriate tags. WP allows you, as a plugin developer, to hook into these events by using those tags. So for example, whenever a post is queried, your script can jump to life and log a view.

But first, we must define a function to do this. Scripting inside WP may look a little alien, but copy the following into your plugin file and we'll talk through it afterwards.

```
function coolness_new_view(){
   // only interested in single posts here
   if (!is_single()) return null;

   global $post; // post in question

   $views = get_post_meta($post->ID, 'coolness_views', true);
   if (!$views) // if $views is undefined
       $views = 0;

   $views++;

   update_post_meta($post->ID, 'coolness_views', $views);
   return $views;
}
```

Your plugin's code will be living in the same environment as all the other active plugins, and so it is important to define a unique function and attribute names. It is customary to prepend your plugin's name to all its functions.

WP has many lifecycle hooks, but they aren't always sufficiently specific for our needs. For example, there is no hook for when a single post is rendered, but there is a generic hook for every time a post query is made. We'll be hooking into it, and so it is important that our function only runs when a single post is requested. The WP-provided `is_single()` function allows us to do just that.

The `global` keyword allows us to get access to variables declared globally outside our function. `$post` is such an example and represents the current queried post. It has some useful predefined attributes, like content, author, and date, but we can set our own custom attributes on it, which WP automatically manages (by interfacing with the DB on our behalf). To do this, we call `get_post_meta()` and `update_post_meta()`. In the function above we use it to get and set the number of views.

To specify when our method should be called, we hook into the `wp_head` tag, like below. This event triggers whenever a frontend page is loaded. Our function is configured only to run when the page loaded is of a single post.

```
add_action('wp_head', 'coolness_new_view');
```

For a comprehensive list of WP actions that you can hook into, see **here**.

ALTERING FRONTEND CONTENT

Now that our plugin can update post views, let's make it display those views on each post as well. First off, in the same file, declare a function to generate some text to display the number of post views:

```
function coolness_views(){
   global $post;
   $views = get_post_meta($post->ID, 'coolness_views', true);
   if (!$views)
       $views = 0;
   return "View count: " . $views;
}
```

Now, the actual rendering of content happens inside whichever theme is active on your site. And furthermore, every theme has its own way of doing such rendering. For demonstrative purposes, please switch your local site's theme to **TwentyTwenty**. This is the one active by default if you installed WP in the year 2020. To change to the appropriate theme, head to your local admin console, go to Customisation → Themes, and click "Activate" on the TwentyTwenty theme.

Now, we'll be making some edits to the theme. Usually, in the industry your company will have its own theme, so you wouldn't need to worry about how your plugin is used inside it — that would be the job of the User experience (UX) designers. But for now, you are the sole developer, so you need to make some theme code changes.

In your IDE, open the file **content.php** in the directory **wp-content/themes/twentytwenty/template-parts**. This file is responsible for single pages and posts, as well as some index pages (e.g. category pages and search pages). You can check the link in its top comment for more details.

About 30 lines down the file, you'll see a **div** tag with class **entry-content**. This wraps the content being rendered (post or page). In it you'll find the following conditional:

```
<?php
if ( is_search() || ! is_singular() && 'summary' ===
                           get_theme_mod( 'blog_content', 'full' ) ) {
  the_excerpt();
} else {
  the_content( __( 'Continue reading', 'twentytwenty' ) );
}
?>
```

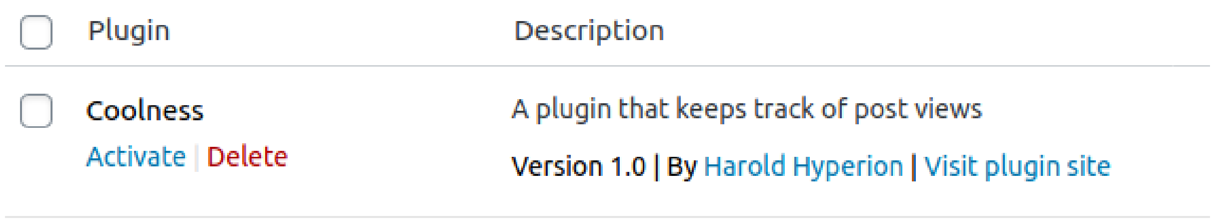
It simply renders either an excerpt or the full content of a page/post (depending on whether it is rendering a single item or not). In either case, you want to put the view count immediately thereafter. Put the following code after the conditional above:

```
// show view count
if (function_exists('coolness_views') and !is_page()){
   echo '<p style="text-align: center">'.coolness_views().'</p>';
}
```

It will output the view count in a **<p>** tag if the currently rendered item is not a page (i.e. a post). It also checks to see if the appropriate **coolness** function exists. If the plugin is not activated by the admin, then the function will not be defined and the theme won't call it (avoiding a function-not-defined error).

ACTIVATING THAT BAD BOY

To see your plugin in action, head to your local WP admin console and under "Plugins", click on "Installed Plugins". *Coolness* should appear in the list. Click on "Activate" to enable it.


	Plugin	Description
☐	Coolness Activate | Delete	A plugin that keeps track of post views Version 1.0 | By Harold Hyperion | Visit plugin site

Now head over to your home page in your browser. If you haven't set up a static home page, then you should see a list of your posts, each with a view count underneath it. Navigate to a post, and refresh the page to (unethically) see its view count go up and up. Take a screenshot of the post with its view counter.

SOME MORE ADVANCED FEATURES

WP gives its plugin developers a lot of useful functionality. One of them is doing internet queries. Let's see how we can use this feature to make a list of the site's most popular posts. Add the following function to your plugin file.

```
function coolness_list(){
   $searchParams = [
       'posts_per_page'=>10,
       'post_type'=>'post',
       'post_status'=>'publish',
       'meta_key'=>'coolness_views',
       'orderby'=>'meta_value_num',
       'order'=>'DESC'
   ];

   $list = new WP_Query($searchParams);

   if ($list->have_posts()){
       global $post;
       echo '<ol>';
       while($list->have_posts()){
           $list->the_post();
           echo '<li><a href="'.get_permalink($post->ID).'">';
           the_title();
           echo '</a></li>';
       }
       echo '</ol>';
   }
}
```

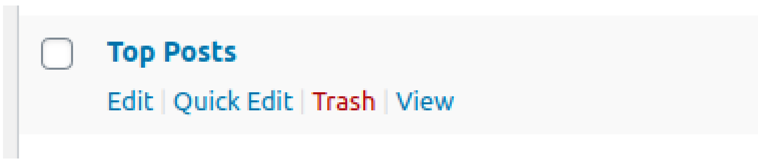
The function starts by declaring some search parameters to be passed into a WP query later. The params specify that we are only interested in the top 10 published posts ordered by our coolness ranking. After constructing the query, we call **`have_posts()`** to check that there is something to display, and then start outputting a list.

WP is set up a little unconventionally internally. Instead of passing around objects as needed, relevant objects are declared and accessed via global variables. So the idiomatic way of looping through a post query in WP is by calling the **`the_post()`** function in a loop and accessing the relevant post via its global variable, **`$post`**.

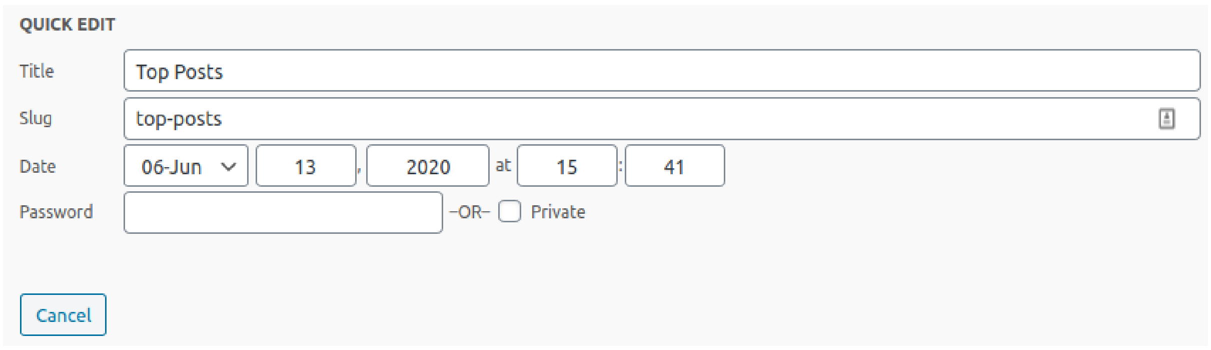
Inside the loop, we output a list item with a link to the relevant post — retrieved via the **`get_permalink()`** function. The title is added via the **`the_title()`** function, which automatically echoes it.

More details about **`WP_Query`** can be found **here**.

There are many ways you could go about displaying this list of top posts, like perhaps on every page, in the admin console, or only on the home page. For the sake of simplicity, we'll be displaying it on one specific page. In your admin console, create a page and title it "Top Posts". You needn't add any content. Immediately publish it. Now head back to your list of pages, and on the one that you just created, click "Quick Edit".



Take note of the slug. It should derive from the page title, but if it doesn't, set it to **`top-posts`** and hit "Update" (bottom right corner; out of screenshot).



QUICK EDIT

Title: Top Posts
Slug: top-posts
Date: 06-Jun 13, 2020 at 15 : 41
Password: –OR– Private

This will allow us to identify the post more easily in our code. Head back to the `content.php` theme file and add the following underneath our previous edit:

```
// show top posts if this is the top-posts page
if (function_exists('coolness_list') and is_page()
   and get_post()->post_name==='top-posts'){

   coolness_list();
}
```

In WordPress version 5, the developers consolidated posts and pages into one object, formally called `WP_Post`. This is why you may come across mixed terminology in parts of the code. To get the page data, you call `get_post()`. And to check its slug you access the `post_name` attribute on it.

As an interesting side note, these kinds of inconsistencies are commonplace in tech projects designed by committee, i.e. a group of people democratically choose what features to add, how they are implemented, what to name them, and so on. This may sound good, but in practice, it often leads to a worse developer experience. Open-source tech projects that are led by one or two people are often praised for their independence and consistency in design — Vue.js, a popular front-end JavaScript framework, is a **good example of this**.

Go check out your new Top Posts page! New pages are usually automatically added to your main menu (at the top of your site). If it wasn't, you'll have to do it manually in your admin console → Customise → Menus → Primary.

Note that posts with 0 views won't be displayed in the list. This is because a post's view count is only set after its first view. So unviewed posts won't have the `coolness_views` attribute and thus be excluded from the query. Take a screenshot of your top posts page after you've confirmed it works.

Extra resource

What has been described in this task really is the tip of the iceberg when it comes to WP's supported plugin features. For more, refer to their **official plugin documentation**.

Compulsory Task 1

Make sure you did everything the guide described. Submit the following:

- A screenshot of a post with its non-zero view count.
- A screenshot of your top posts page, with post links ordered by view count.

Compulsory Task 2

Add the following features to your site by extending the coolness plugin:

- Alter the view count text to have a dynamic plural. It should read "This post has x views.", but if there is 1 view, the text should end in a singular noun.
- Explicitly set new posts' view counts to 0.
  - *Hint: the WP action tag **`save_post`** executes whenever a post or page is altered by an author. Ensure that your function only runs on posts (not pages), and does not change the view count of pre-existing posts (that already have view counts).*
- Have the top posts listing display the relevant view count next to each post link.

Submit your modified version of **coolness.php**.