

Action Hooks:

1. `init`: Fired after WordPress has finished loading but before any headers are sent. Often used for custom post type and taxonomy registrations.
2. `wp_loaded`: Fired after WordPress has fully loaded.
3. `widgets_init`: Used to register widgets and sidebars.
4. `wp_head`: Allows you to add code or scripts to the `<head>` section of the HTML document.
5. `wp_footer`: Used to add code or scripts just before the closing `</body>` tag.
6. `admin_init`: Similar to `init`, but specifically for the admin area.
7. `admin_menu`: Used to add or modify items in the WordPress admin menu.
8. `save_post`: Fired when a post or page is saved in the admin panel.
9. `shutdown`: Fired just before PHP execution ends.
10. `template_redirect`: Fired just before WordPress determines which template page to load.
11. `wp_enqueue_scripts`: Used to enqueue scripts and styles on the front end of the site.
12. `admin_enqueue_scripts`: Similar to `wp_enqueue_scripts`, but specifically for the admin area.
13. `wp_ajax_{action}`: Used for handling AJAX requests on the front end.
14. `admin_post_{action}`: Used for handling AJAX requests in the admin area.
15. `pre_get_posts`: Allows modification of the main query before it is executed.
16. `wp_insert_post_data`: Used to filter and modify post data before it's inserted into the database.
17. `wp_login`: Fired when a user logs in.
18. `wp_logout`: Fired when a user logs out.
19. `wp_before_admin_bar_render`: Allows modification of the admin toolbar (admin bar).
20. `publish_post`: Fired when a post is published.
21. `transition_post_status`: Fired when a post transitions from one status to another (e.g., from draft to published).
22. `widgets_admin_page`: Used to add content to the widgets admin page.
23. `comment_post`: Fired after a comment is posted.
24. `admin_notices`: Used to display notices in the WordPress admin area.
25. `wp_dashboard_setup`: Used to add widgets and elements to the WordPress dashboard.
26. `admin_print_scripts`: Fired when scripts are printed in the admin area.
27. `admin_print_styles`: Fired when stylesheets are printed in the admin area.
28. `load-{page}`: Fired when a specific admin page is loaded (e.g., `load-post.php`).
29. `auth_redirect`: Fired when authentication is required for a user to access a page.
30. `all`: A catch-all hook that fires on every WordPress page load.

Filter Hooks:

1. `the_content`: Used to filter the content of a post or page before it's displayed.
2. `the_title`: Filters the title of a post or page before it's displayed.
3. `widget_text_content`: Filters the content of a text widget.
4. `wp_nav_menu_items`: Used to modify the items in a navigation menu.
5. `excerpt_length`: Allows you to modify the length of excerpts.
6. `wp_insert_post_data`: Used to filter and modify post data before it's inserted into the database.
7. `the_excerpt`: Filters the post excerpt.
8. `the_meta_key`: Used to filter the meta key when retrieving post metadata.
9. `widget_text_content`: Filters the content of a text widget.
10. `comment_text`: Filters the text of a comment before it's displayed.
11. `comment_form_defaults`: Used to modify the default settings for the comment form.
12. `widget_text_content`: Filters the content of a text widget.
13. `wp_mail_content_type`: Allows you to modify the content type of emails sent via the `wp_mail()` function.
14. `pre_get_posts`: Allows modification of the main query before it is executed.
15. `the_permalink`: Filters the post or page permalink before it's displayed.
16. `wp_get_attachment_image_attributes`: Filters the attributes of an image added via `wp_get_attachment_image()`.
17. `login_redirect`: Used to filter the URL where users are redirected after logging in.
18. `the_author`: Filters the author's display name.
19. `the_time`: Filters the time or date format of a post's publish date.
20. `wp_calculate_image_srcset`: Filters the image srcset attributes.
21. `wp_kses_allowed_html`: Allows customization of the allowed HTML elements and attributes when using `wp_kses()`.
22. `upload_mimes`: Used to filter allowed file types for uploads.
23. `wp_editor_settings`: Filters the settings used for the WordPress visual editor (TinyMCE).
24. `wp_dropdown_categories`: Filters the arguments used in the dropdown category select box.
25. `wp_title`: Filters the title displayed in the HTML `<title>` tag.
26. `the_archive_description`: Filters the archive description text.
27. `the_category`: Filters the category display for a post.
28. `image_send_to_editor`: Filters the HTML markup of images when added to the editor.
29. `wp_insert_attachment_data`: Used to filter attachment data before it's inserted into the database.
30. `get_search_form`: Filters the HTML markup of the search form.

Add filter hook

```
function custom_excerpt_length( $length ) {
    return 5;
}

add_filter( 'excerpt_length', 'custom_excerpt_length' );

add_filter('the_title','custom_title_changes');

function custom_title_changes($title){
    //return $title.' Diwakar Academy';

    if(!is_admin()){
        return "<i>$title</i>";
    }else{
        return $title;
    }
}

function da_title_modify($title){
    if(!is_admin() && !is_page()){
        return 'diwakar '. $title.' academy';
    }else{
        return $title;
    }
}

add_action('the_content','da_content_modify');

function da_content_modify($content){

    return '<div class="main_content">'.$content.'</div>';
}
```

```
add_filter('excerpt_length', 'da_excerpt_length');

function da_excerpt_length($length) {
    return 15;
}

add_action('excerpt_more', 'da_excerpt_more', 999);

function da_excerpt_more($more) {
    return '*****';
}
```

```
add_filter('login_headerurl', 'da_modify_logo_url');

function da_modify_logo_url($url){
    return esc_url(home_url());
}

add_filter('login_headertext', 'da_modify_headertext');

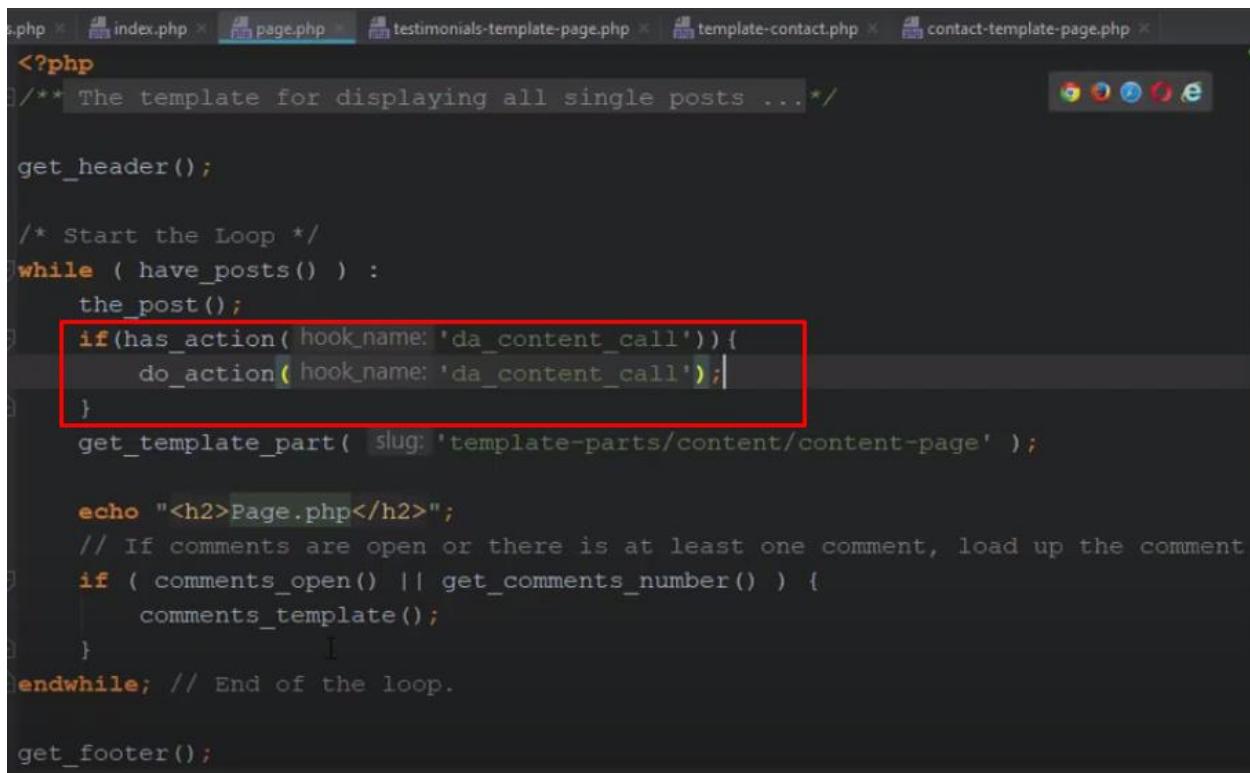
function da_modify_headertext($headertext){
    return 'Diwakar Academy';
}
```

Custom Action Hook

```
function da_add_content_callback(){

    echo "<h2>Custom Action hook</h2>";
}

add_action('da_content_call','da_add_content_callback');
```



/* The template for displaying all single posts ... */

get_header();

/* Start the Loop */
while (have_posts()) :
 the_post();
 if(has_action('hook_name: "da_content_call"')){
 do_action('hook_name: "da_content_call"');
 }
 get_template_part('slug: "template-parts/content/content-page"');

 echo "<h2>Page.php</h2>";
 // If comments are open or there is at least one comment, load up the comment
 if (comments_open() || get_comments_number()) {
 comments_template();
 }
endwhile; // End of the loop.

get_footer();

```
function da_add_content_callback($values){

    print_r($values);
    echo "<h2>Hi ".$values['first_name'].' '.$values['last_name']. '</h2>';
}

add_action('da_content_call','da_add_content_callback');
```

The screenshot shows a code editor with multiple tabs at the top: ns.php, index.php, page.php (which is the active tab), testimonials-template-page.php, template-contact.php, and contact-template-page.php. The code in the editor is:

```
<?php
/* The template for displaying all single posts ...*/
get_header();
global $post;
/* Start the Loop */
while ( have_posts() ) :
    the_post();

    if(has_action( hook_name: 'da_content_call')){
        do_action( hook_name: 'da_content_call', ['first_name'=>'Amit', 'last_name'=>'']
    }
    get_template_part( slug: 'template-parts/content/content-page' );
}
```

Custom filter

```
add_filter('da_modify_title','da_modify_page_title');

function da_modify_page_title($data){
    return "<h4 class='custom-heading'>$data</h4>";
}
```

The screenshot shows a code editor with the same tabs as the previous screenshot. The code has been modified to include a custom filter application:

```
get_header();
global $post;
if(has_action( hook_name: 'da_content_call')){
    //do_action('da_content_call', ['page_id'=> $post->ID]);
}
/* Start the Loop */
while ( have_posts() ) :
    the_post();

    if(has_filter( hook_name: 'da_modify_title')){
        echo apply_filters('da_modify_title',get_the_title());
    }
}
```

```
add_filter('da_add_content', 'da_modify_content_callback');

function da_modify_content_callback($content){
    return '<div>Append Text' . $content . '</div>';
}
```

The screenshot shows a code editor interface with three tabs at the top: 'actions.php', 'index.php', and 'page.php'. The 'page.php' tab is active, indicating the current file being edited. The code in the editor is a PHP template for displaying single posts. It includes standard WordPress functions like 'get_header()', 'global \$post;', and the 'while' loop for posts. It also demonstrates the use of hooks and filters. Specifically, it checks for the 'da_content_call' action and the 'da_modify_title' filter. If the filter is present, it echoes the modified title using 'apply_filters'. If the action is present, it echoes the modified content using 'apply_filters'. Finally, it includes a call to 'get_template_part()' for a specific content template part.

```
<?php

/** The template for displaying all single posts ... */

get_header();
global $post;
if(has_action( 'hook_name: "da_content_call" )) {
    //do_action('da_content_call', ['page_id'=> $post->ID]);
}
/* Start the Loop */
while ( have_posts() ) :
    the_post();

    if(has_filter( 'hook_name: "da_modify_title"' )) {

        echo apply_filters('da_modify_title', get_the_title());
    }

    if(has_filter( 'hook_name: "da_add_content"' )) {
        echo apply_filters('da_add_content', get_the_content());
    }
get_template_part( slug: 'template-parts/content/content-page' );
}
```

- [did_action\(\)](#)

If you want to count the number of times any action is fired, you can invoke this action function.

```
did_action( 'action_name' );
```

- [do_action_ref_array\(\)](#)
-

This action function is identical to `do_action()`, except for one difference. Any arguments passed through it must be an array. When you have a lot of arguments to pass or your arguments are already in an array, this function is super helpful.

```
// here's an example array
$arguments_array = array( 'arg_1', 'foo', true, 'arg_4' );
do_action_ref_array( 'example_action', $arguments_array );
```

- [remove_action\(\)](#)

This action function removes a callback function hooked to the specified action. For instance, you can use this function to remove the default WordPress functions hooked into built-in actions and replace them with your own.

```
remove_action( 'action_name', 'function_to_be_removed', [priority] );
```

There are a few prerequisites to calling the `remove_action()` function:

1. The `function_to_be_removed` and `priority` parameters must be the same as the ones used originally in the `add_action()` function.
2. You cannot call the `remove_action()` function directly. You need to call it from inside another function.
3. If the callback function is registered from a *class*, then removing it has other requirements. You can check out the WordPress Codex documentation for more details.
4. You cannot remove the callback function before it's registered or after it's run.

- [remove_all_actions\(\)](#)

This action function removes everything hooked to an action. The `priority` parameter is optional.

```
remove_all_actions( 'action_name', [priority] );
```

Remember that this function can't be called from the action you would like to deregister callback functions from. That would cause an infinite loop. You can hook into an action that's fired earlier to run this function without any errors.

- [doing_action\(\)](#)

This action function checks whether the specified action is being run or not. It returns a boolean value (`true` or `false`).

```
// check whether the 'action_name' action is being executed
if ( doing_action( 'action_name' ) ) {
    // execute your code here
}
```

Actions Example 1: Show a Maintenance Message to Your Site Visitors

Sometimes, it's best to take your site offline and put up an [Under Maintenance page](#). Thankfully, WordPress provides an easy way to do just that.

```
// show a maintenance message for all your site visitors
add_action( 'get_header', 'maintenance_message' );
function maintenance_message() {
    if (current_user_can( 'edit_posts' ) ) return;
    wp_die( '<h1>Stay Pawsitive!</h1><br>Sorry, we\'re temporaril
}'
```

Let's break down the code and go through each step:

- `get_header` is an action that's triggered before the header template file of the site loads. It's a perfect action to hook into if you want to interrupt the main site from being loaded.
 - Hook into the `get_header` action using the `add_action()` function with the `maintenance_message()` callback function.
 - Define the `maintenance_message()` callback function.
 - `current_user_can('edit_posts')` is a [user capability test function](#) that checks whether the current user is logged in and [can edit posts](#). [Every user registered on a](#)
-

[WordPress site](#) except for those with Subscriber roles has the capability to edit posts.

There are other robust ways to perform this check, but we'll stick with this simple method here.

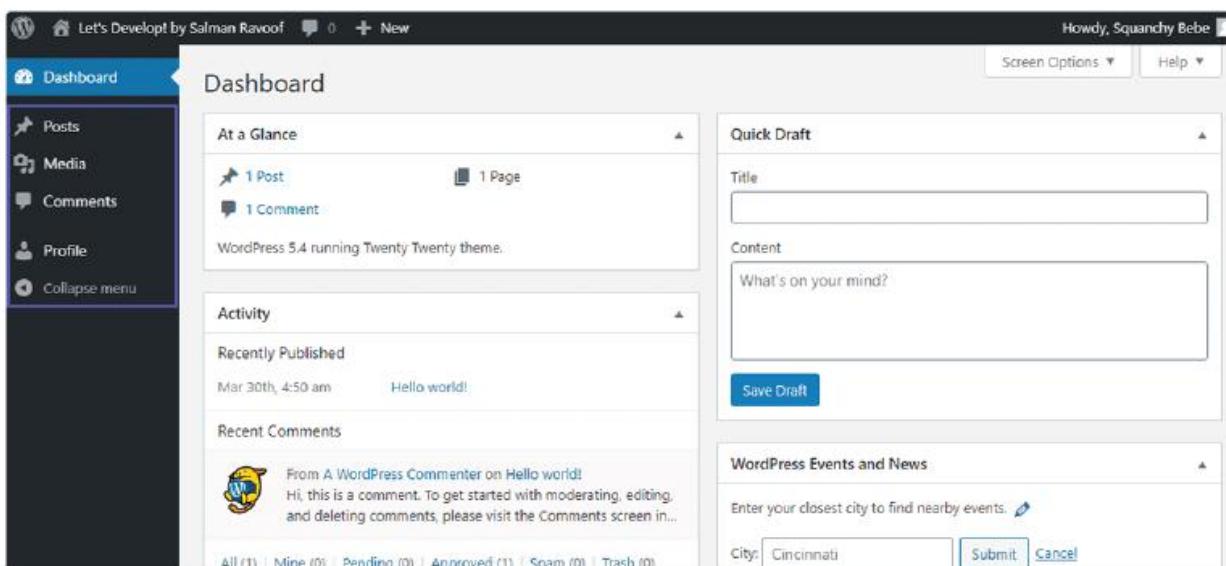
- Use the default `wp_die()` function to kill WordPress execution gracefully and display an HTML page with an error message. You can use HTML syntax in the error message parameter to format it.

Hide Dashboard Menu Items from Non-Admin Users

```
// remove specific dashboard menus for non-admin users
add_action( 'admin_menu', 'hide_admin_menus' );
function hide_admin_menus() {
    if (current_user_can( 'create_users' )) return;
    if (wp_get_current_user()->display_name == "Salman") return;
    remove_menu_page( 'plugins.php' );
    remove_menu_page( 'themes.php' );
    remove_menu_page( 'tools.php' );
    remove_menu_page( 'users.php' );
    remove_menu_page( 'edit.php?post_type=page' );
    remove_menu_page( 'options-general.php' );
}
```

Here's a step-by-step walkthrough of the code snippet above:

- `admin_menu` is an action that's triggered before the admin menu loads in the WordPress dashboard area.
- Hook into the `admin_menu` action using the `add_action()` function using the `hide_admin_menus()` callback function.
- The `hide_admin_menus()` callback function defines the logic of the code. It's run every time the `admin_menu` action fires.
- Inside the callback function, the `current_user_can('create_users')` function checks whether the logged-in user is an admin. Since only site admins have the `create_user` capability, the function ends with a `return` statement if the user is an admin.
- The `wp_get_current_user()` WordPress function retrieves the current user object. With this function, we can check whether the logged-in user has a particular `display_name` set. This is an optional line, in case you want to omit certain non-admin users from being locked out due to this callback function.
- The `remove_menu_page()` WordPress function removes top-level admin menus. In the code example above, I'm removing the following admin menus: Plugins, Themes, Tools,



- [apply_filters_ref_array\(\)](#)

This function is like the `apply_filters()` function, except all the arguments it accepts are bundled up as an array.

```
// an example array
$arguments_array = array( 'some_value', 'foo', false, 'another_value' );

apply_filters_ref_array( 'example_filter', $arguments_array );
```

- [current_filter\(\)](#)

This filter function retrieves the name of the current filter or action being run. You don't need to specify any parameters as it runs within the callback function.

Here's an example of its usage:

```
function example_callback() {
    echo current_filter(); // 'the_title' will be echoed
    return
}
add_filter( 'the_title', 'example_callback' );
```

- [remove_filter\(\)](#)

This filter function removes the callback function attached to the specified filter. It's works exactly like the `remove_action()` function. You can use it to delete default WordPress functions registered with a specific filter, and if necessary replace them with your own functions.

```
remove_filter( 'filter_name', 'function_to_be_removed', [priority] );
```

To unhitch a callback function hooked to a filter, the `function_to_be_removed` and `priority` parameters must be identical to the arguments used when hooking the callback function.

If the filter has been added from within a class, which is usually the case when they're added by plugins, then you need to [access the class variable to remove the filter](#).

Say you want to reset this functionality and send your users to the [default password retrieval page](#) or to a separate page altogether. You can remove this callback function like this:

```
remove_filter( 'lostpassword_url', 'wc_lostpassword_url', 10 );
```

- [remove_all_filters\(\)](#)

This filter function removes all the callback functions registered to a filter.

```
remove_all_filters( 'filter_name', [priority] );
```

- [doing_filter\(\)](#)

This filter function checks whether the filter specified is being executed at the moment.

```
if ( doing_filter( 'save_post' ) ) {  
    // run your code here  
}
```

It returns a boolean value (`true` or `false`).

You should note the difference between this function and the `current_filter()` function, which returns the name of the filter or action being run (a string).

Remove action hook

`remove_action` function is used to remove actions that have been previously added using the `add_action` function.

```
php  
  
if (is_single()) {  
    remove_action('hook_name', 'function_name');  
}
```

[Copy code](#)

```
php  
  
function custom_theme_setup() {  
    // Remove a theme action  
    remove_action('theme_hook', 'theme_function');  
}  
add_action('after_setup_theme', 'custom_theme_setup');
```

[Copy code](#)

Remove filter hook

the `remove_filter` function is used to remove filters that have been previously added using the `add_filter` function.

```
php Copy code  
  
if (is_single()) {  
    remove_filter('hook_name', 'function_name');  
}
```

```
php Copy code  
  
function custom_theme_setup() {  
    // Remove a theme filter  
    remove_filter('theme_hook', 'theme_function');  
}  
add_action('after_setup_theme', 'custom_theme_setup');
```

Remove Filter Hook

In WordPress, you can remove filter hooks using the `remove_filter` function. This function allows you to detach a specific callback function from a filter hook, preventing it from modifying the data or content passed through that hook. Here's how to use `remove_filter` to remove a filter hook:

Syntax:

```
remove_filter($hook_name, $function_to_remove, $priority);
```

- `$hook_name` (string): The name of the filter hook from which you want to remove the callback function.
- `$function_to_remove` (callable): The callback function you want to remove.

- `$priority` (int, optional): The priority at which the callback was originally added. This is optional and should only be specified if the callback was added with a specific priority.

```
// Callback function to add a filter.
function custom_filter_function($content) {
    return $content . '<p>Additional content added by the filter.</p>';
}

// Attach the callback function to the 'the_content' filter with a priority
// of 10.
add_filter('the_content', 'custom_filter_function', 10);

// ... Later in your code, you decide to remove the filter.

// Remove the filter.
remove_filter('the_content', 'custom_filter_function', 10);
```

- 1.We initially attach the `custom_filter_function` callback to the `the_content` filter with a priority of 10 using `add_filter`.
- 2.Later, if we decide to remove this filter, we use `remove_filter` with the same filter hook name, callback function name, and priority (if applicable).

Uses of `remove_action` hook

The `remove_action` hook is used in WordPress to remove or unregister an action hook that was previously added using the `add_action` function. It's particularly useful when you want to prevent certain actions from executing or modify the behavior of a theme or plugin without directly editing its code. Here's how you can use the `remove_action` hook with an example:

```
add_action('wp_footer', 'my_custom_function');
```

This action hook, `wp_footer`, is commonly used to add content or scripts to the footer of a WordPress site. The `my_custom_function` function is the callback function that will be executed when this action hook is triggered.

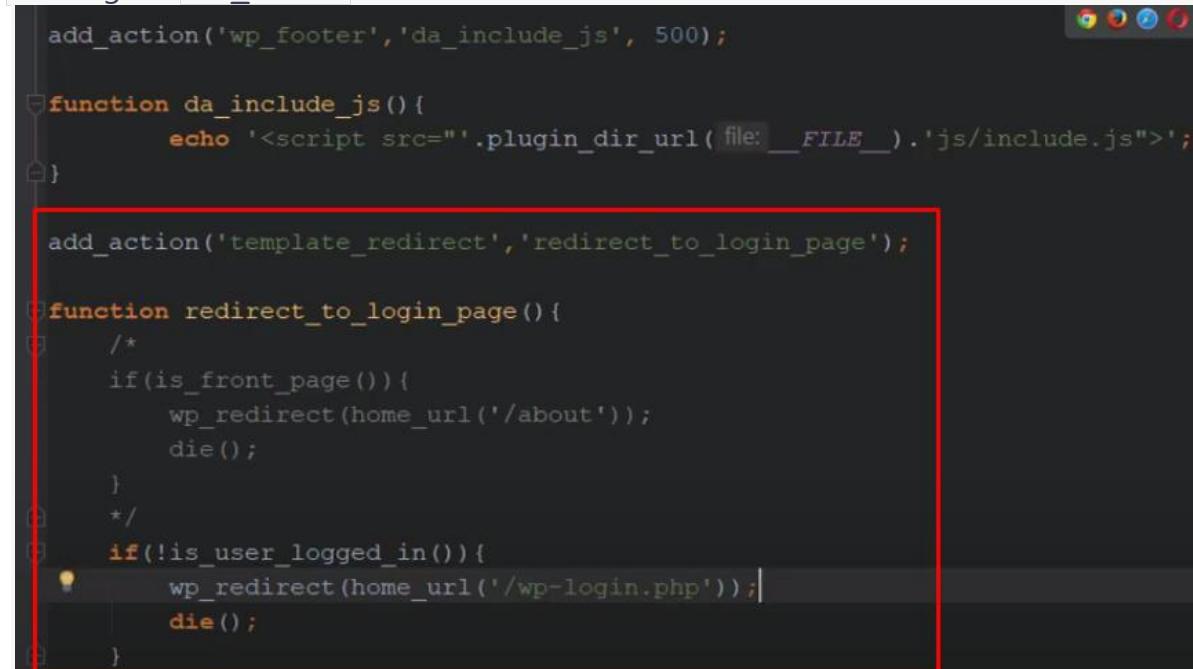
Now, if you want to remove or prevent `my_custom_function` from executing on the `wp_footer` action hook, you can use the `remove_action` hook. Here's how to do it:

```
function remove_my_custom_function() {  
    remove_action('wp_footer', 'my_custom_function');  
}  
  
add_action('init', 'remove_my_custom_function');
```

1. We create a custom function called `remove_my_custom_function`.
2. Inside `remove_my_custom_function`, we use the `remove_action` function to remove the `my_custom_function` callback from the `wp_footer` action hook.
3. We add our custom function `remove_my_custom_function` to the `init` action hook. This ensures that it runs early enough in the WordPress loading process to remove the action before it's executed.

By using `remove_action` in this way, the `my_custom_function` will no longer be triggered when the `wp_footer` action hook is fired. This can be useful for disabling or modifying the behavior of themes or plugins without editing their source code directly, making your customizations more maintainable and compatible with future updates.

Remember that you need to specify the exact same callback function and action hook that you want to remove, as well as ensure that your `remove_action` call occurs after the original `add_action` call.



The screenshot shows a code editor with a dark theme. A red rectangle highlights a section of code in the center. The code is as follows:

```
add_action('wp_footer', 'da_include_js', 500);  
  
function da_include_js(){  
    echo '<script src="'.plugin_dir_url( file: __FILE__ ).'js/include.js">';  
}  
  
add_action('template_redirect','redirect_to_login_page');  
  
function redirect_to_login_page(){  
    /*  
     * if(is_front_page()){  
     *     wp_redirect(home_url('/about'));  
     *     die();  
     * }  
     */  
    if(!is_user_logged_in()){  
        wp_redirect(home_url('/wp-login.php'));  
        die();  
    }  
}
```

Nonce in WordPress

A WordPress nonce is a “number used once” security token to protect URLs and forms from malicious attacks. It helps WordPress to determine whether a request is valid, preventing unauthorized actions and inputs.

These nonce-related functions are essential for adding security to WordPress forms, AJAX requests, and other actions to protect against Cross-Site Request Forgery (CSRF) attacks. Depending on your specific use case, you'll choose the appropriate function to generate and verify nonces in your WordPress code.

1. `wp_nonce_field()`: Generates a nonce and outputs it as a hidden form field. This function is often used in HTML forms to include a nonce for verification when the form is submitted.
2. `wp_create_nonce($action)`: Generates a nonce for a specific action. You provide a unique identifier (action) to associate the nonce with a particular operation.
3. `wp_verify_nonce($nonce, $action)`: Verifies that a nonce matches the provided action. It returns `true` if the nonce is valid and `false` if it is not.
4. `check_admin_referer($action, $query_arg)`: Verifies a nonce used in the administration area. It checks if the nonce in the request matches the one associated with the given action.
5. `check_ajax_referer($action, $query_arg, $die)`: Similar to `check_admin_referer()`, but specifically designed for AJAX requests. It verifies that the nonce in the request matches the one associated with the given action.
6. `wp_nonce_url($url, $action, $name)`: Adds a nonce to a URL. This function can be used to create secure URLs for actions that require nonces.
7. `wp_nonce_ays($action)`: Outputs a "Are you sure you want to do this?" message along with a nonce verification field. It's often used in confirmation dialogs before performing critical actions.
8. `wp_nonce_tick()`: Returns the current "tick" value, which is used to generate and validate nonces. It's primarily an internal function and is not typically used directly in plugins or themes.

Use cases for nonces

- Form submissions (e.g., saving settings, updating a post).
- AJAX requests that modify data.
- Logins and password resets.
- Actions related to user authentication and authorization.
- REST API
- against Cross-Site Request Forgery (CSRF) attacks

```
-      </label>
-      <input type="text" name="txtemail" value="<?php echo isset($row_details['e
-    </p>
-    <?php //wp_nonce_field("student_action_nonce", "student_name_nonce"); ?>
-    <p>
-      <button type="submit" name="btndsubmit">Submit</button>
-    </p>
-  </form>
-
-  <script>
-
-    jQuery(function () {
-      jQuery("#frmsubmitdata").validate({
-        submitHandler: function () {
-          var nonce = "<?php echo wp_create_nonce('student nonce'); ?>";
-          var postdata = jQuery("#frmsubmitdata").serialize() + "&action=myaj
-          jQuery.post("<?php echo admin_url('admin-ajax.php') ?>", postdata,
-            console.log(response);
-          })
-        });
-      });
-    });
-
```



```
add_action("wp_ajax_myajax", "myjaxfunction");

function myjaxfunction() {
  check_ajax_referer('student_nobnce1');
  echo "This is running";
  /*if (wp_verify_nonce($_REQUEST['student_name_nonce'], "student_action_nonce"))
    echo "Status = 1";
  } else {
    echo "Status = 0";
  */
  wp_die();
}

}
```



How to use nonce in wp

1. Generate a Nonce:

2. Include the Nonce in Your Form or Request

```
<form method="post" action="">
<!-- Other form fields --&gt;
&lt;?php wp_nonce_field('my_action', 'my_nonce'); ?&gt;
&lt;input type="submit" value="Submit"&gt;
&lt;/form&gt;</pre>
```

In an AJAX request:

```
var myNonce = '<?php echo $nonce; ?>';
var data = {
    action: 'my_ajax_action',
    nonce: myNonce,
    // Other data to send
};

};
```

3. Verify the Nonce: When processing the form submission or handling the AJAX request on the server side, you should verify the nonce to ensure that the request is legitimate and authorized. Use the `check_admin_referer()` or `check_ajax_referer()` function to do this.

```
if ( ! isset( $_POST['my_nonce'] ) || ! wp_verify_nonce( $_POST['my_nonce'],
'my_action' ) ) {
    // Nonce verification failed, handle the error or exit.}
```

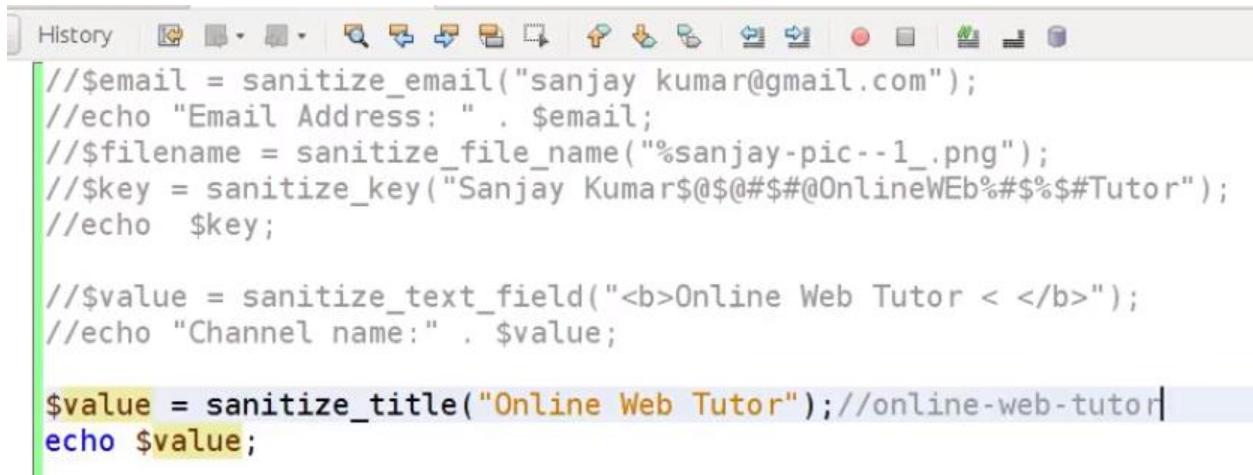
Or for AJAX:

```
if ( ! check_ajax_referer( 'my_action', 'nonce', false ) ) {
    // Nonce verification failed, handle the error or exit.
}
```

4. Perform the Authorized Action: If the nonce verification passes, you can proceed to perform the authorized action, such as saving data, updating settings, or any other operation.

Sanitizing input data to enhance security

WordPress provides several built-in functions for sanitizing input data to enhance security and prevent various types of vulnerabilities, such as SQL injection and Cross-Site Scripting (XSS) attacks. Here is a list of some of the key sanitization functions in WordPress:



```
//$email = sanitize_email("sanjay kumar@gmail.com");
//echo "Email Address: " . $email;
//$/filename = sanitize_file_name("%sanjay-pic--1_.png");
//$/key = sanitize_key("Sanjay Kumar$@#$@#OnlineWeb%#$%#Tutor");
//echo $key;

//$/value = sanitize_text_field("<b>Online Web Tutor < /b>");
//echo "Channel name:" . $value;

$value = sanitize_title("Online Web Tutor"); //online-web-tutor
echo $value;
```

```
-----  
esc_html()  
This function escapes HTML specific characters.  
echo esc_html("<html>Sanjay</html>"); => "&lt;html&gt;Sanjay&lt;/html&gt;"  
-----  
esc_textarea()  
Use esc_textarea() instead of esc_html() while displaying text in textarea. Because  
esc_textarea() can double encode entities.  
-----  
esc_attr()  
This function encodes the <,>, &, " and ' characters. It will never double encode  
entities. This function is used to escape the value of HTML tags attributes.  
-----  
esc_url()  
URLs can also contain JavaScript code in them. So, if you want to display a URL or a  
complete <a> tag, then you should escape the href attribute or else it can cause an  
XSS attack.  
-----  
$url = "javascript:alert('Hello OWTF')";
```



1. **sanitize_text_field(\$str)**: Sanitizes a string by removing potentially harmful characters and tags. This function is often used to sanitize user input from text fields.
2. **sanitize_email(\$email)**: Sanitizes an email address by removing potentially harmful characters and validating the email format.
3. **sanitize_file_name(\$filename)**: Sanitizes a file name by removing potentially harmful characters and ensuring it adheres to the allowed file name characters.
4. **sanitize_title(\$title)**: Sanitizes a title or post name by removing potentially harmful characters and transforming it into a URL-friendly slug.
5. **sanitize_title_for_query(\$title)**: Sanitizes a title for use in database queries, ensuring it is safe to include in SQL statements.
6. **sanitize_key(\$key)**: Sanitizes a key or identifier by removing potentially harmful characters, making it suitable for use in array keys or query variables.
7. **esc_html(\$text)**: Escapes a string for use in HTML content, preventing XSS attacks by encoding special characters.
8. **esc_attr(\$text)**: Escapes a string for use in HTML attributes, ensuring that special characters do not disrupt the HTML structure.
9. **esc_url(\$url)**: Escapes a URL to prevent potential security issues, such as injecting malicious links.
10. **esc_sql(\$sql)**: Escapes a SQL query string to prevent SQL injection attacks when building custom database queries.
11. **wp_kses(\$data, \$allowed_html)**: Sanitizes and validates HTML content based on an array of allowed HTML elements and attributes.
12. **wp_kses_data(\$data)**: Sanitizes and validates HTML content, allowing only safe tags and attributes.
13. **wp_strip_all_tags(\$string, \$remove_breaks)**: Strips all HTML and PHP tags from a string. You can choose to keep or remove line breaks.
14. **balanceTags(\$text, \$force)**: Ensures that HTML tags in a string are properly balanced, preventing broken or mismatched tags.
15. **filter_var(\$variable, \$filter, \$options)**: PHP's built-in function for filtering and validating data. WordPress developers may use this function in custom code to validate various types of input.
16. **intval(\$var, \$base)**: Casts a variable to an integer, which can be useful for sanitizing numeric input.
17. **absint(\$maybeint)**: Casts a value to an absolute integer, ensuring it's a non-negative integer.

Escaping output

Escaping output is a crucial security practice to prevent Cross-Site Scripting (XSS) attacks and maintain the integrity of your website. When you output data to the browser, it's important to make sure that any potentially unsafe content is properly escaped to prevent malicious code execution. WordPress provides several functions to help you escape output in different contexts. Here are some of the key functions:

esc_html(\$text): Use this function to escape and sanitize text for output in HTML content. It encodes special characters like <, >, &, and quotes ("") to their HTML entities.

```
echo esc_html($text);
```

esc_attr(\$text): This function is used to escape text for use in HTML attributes. It encodes special characters and ensures that the content is safe within an attribute.

```
echo '<input type="text" value="' . esc_attr($value) . '">';
```

esc_url(\$url): Use this function to escape URLs to prevent any potential security issues. It ensures that URLs are properly formatted and safe for use in links and attributes.

```
echo '<a href="' . esc_url($url) . '">Link</a>';
```

esc_js(\$javascript): When outputting JavaScript code, use this function to escape it properly. It can help prevent script injection vulnerabilities.

```
echo '<script>' . esc_js($javascript_code) . '</script>';
```

esc_textarea(\$text): Use this function to escape text for output within a <textarea> element. It ensures that line breaks and special characters are properly encoded.

```
echo '<textarea>' . esc_textarea($text) . '</textarea>';
```

esc_sql(\$sql): When constructing custom database queries, use **esc_sql()** to escape and sanitize SQL query strings. This helps prevent SQL injection attacks.

```
$query = "SELECT * FROM {$wpdb->prefix}my_table WHERE name = " . esc_sql($name);
```

wp_kses(\$data, \$allowed_html): Use this function to sanitize and validate HTML content based on an array of allowed HTML elements and attributes. This is useful when you want to allow certain HTML tags while disallowing others.

```
echo wp_kses($data, $allowed_html);
```

wp_kses_post(\$data): Similar to **wp_kses()**, this function is specifically designed to sanitize and validate HTML content within WordPress posts, preserving certain tags allowed in posts.

```
echo wp_kses_post($data);
```

sanitize_text_field(\$text): Although primarily used for sanitizing input, this function can also be used to escape and sanitize text for output in HTML content.

```
echo sanitize_text_field($text);
```

When working with WordPress themes or plugins, it's important to choose the appropriate escaping function based on the context of your output. Always use these functions to escape user-generated content and data coming from untrusted sources to ensure the security of your WordPress website.

```
<?php
/*
Plugin Name: Escaping Functions Demo
Description: A simple plugin to demonstrate the use of escaping functions in
WordPress.
*/
// Display a user comment safely.
function display_comment_safely() {
    $comment_text = get_comment_text(); // Get the user's comment text from
the database.
    echo esc_html($comment_text); // Display the comment text safely.
}
add_shortcode('display_comment', 'display_comment_safely');

// Display a post title safely.
function display_post_title_safely() {
    $post_title = get_the_title(); // Get the post title from the database.
    echo esc_html($post_title); // Display the post title safely.
}
```

```
add_shortcode('display_post_title', 'display_post_title_safely');

// Display an image URL safely within an attribute.
function display_image_url_safely() {
    $image_url = get_post_meta(get_the_ID(), '_image_url', true); // Get an
image URL from custom field.
    echo '';
}
add_shortcode('display_image', 'display_image_url_safely');

// Display JavaScript code safely.
function display_javascript_safely() {
    $javascript_code = 'alert("Hello, world!");';
    echo '<script>' . esc_js($javascript_code) . '</script>';
}
add_shortcode('display_javascript', 'display_javascript_safely');

// Display text content safely within a textarea.
function display_textarea_content_safely() {
    $textarea_content = get_user_meta(get_current_user_id(), 'description',
true); // Get user description.
    echo '<textarea>' . esc_textarea($textarea_content) . '</textarea>';
}
add_shortcode('display_textarea_content', 'display_textarea_content_safely');

// Custom database query with proper escaping.
function custom_database_query() {
    $name = sanitize_text_field($_GET['name']); // Sanitize user input.
    global $wpdb;
    $query = "SELECT * FROM {$wpdb->prefix}my_table WHERE name = " .
esc_sql($name);
    // Execute the query and process the results.
}
```

`esc_html()` is a WordPress function used for escaping and sanitizing HTML content to protect against cross-site scripting (XSS) attacks. It ensures that any potentially dangerous HTML or special characters are properly encoded before being displayed on a web page. The primary purpose of `esc_html()` is to make user-generated or untrusted data safe for output in your WordPress templates.

Here's how you use `esc_html()`:

php

 Copy code

```
<?php
$untrusted_html = '<script>alert("XSS Attack");</script>';
$escaped_html = esc_html($untrusted_html);
echo $escaped_html;
?>
```

- **esc_html()** – Use anytime an HTML element encloses a section of data being displayed. This will remove HTML.

Copy

```
<h4><?php echo esc_html( $title ); ?></h4>
```

- **esc_js()** – Use for inline Javascript.

Copy

```
<div onclick='<?php echo esc_js( $value ); ?>' />
```

- **esc_url()** – Use on all URLs, including those in the src and href attributes of an HTML element.

Copy

```

```

- `esc_attr()` - Use on everything else that's printed into an HTML element's attribute.

[Copy](#)

```
<ul class="<?php echo esc_attr( $stored_class ); ?>">
```

- `esc_textarea()` - Use this to encode text for use inside a textarea element.
- `wp_kses()` - Use to safely escape for all non-trusted HTML (post text, comment text, etc.). This preserves HTML.
- `wp_kses_post()` - Alternative version of `wp_kses()`that automatically allows all HTML that is permitted in post content.
- `wp_kses_data()` - Alternative version of `wp_kses()`that allows only the HTML permitted in post comments.

🔗 Escaping with Localization

[Top ↑](#)

Rather than using `echo` to output data, it's common to use the WordPress localization functions, such as `_e()` or `__()`.

These functions simply wrap a localization function inside an escaping function:

[Copy](#)

```
esc_html_e( 'Hello World', 'text_domain' );
// Same as
echo esc_html( __( 'Hello World', 'text_domain' ) );
```

These helper functions combine localization and escaping:

- `esc_html__()`
- `esc_html_e()`
- `esc_html_x()`
- `esc_attr__()`
- `esc_attr_e()`
- `esc_attr_x()`

- `$super_admins` (array): An array of user IDs that should be granted super admin privileges (multisite). This global is only set by the site owner (e.g., in `wp-config.php`), and contains an array of IDs of users who should have super admin privileges. If set it will override the list of super admins in the database.
- `$wp_query` (object): The global instance of the [WP_Query](#) class.
- `$wp_rewrite` (object): The global instance of the [WP Rewrite](#) class.
- `$wp` (object): The global instance of the [WP](#) environment setup class.
- `$wpdb` (object): The global instance of the [wpdb](#) class.
- `$wp_locale` (object): The global instance of the [WP_Locale](#) class.
- `$wp_admin_bar` (object): The global instance of the [WP Admin Bar](#) class.
- `$wp_roles` (object): The global instance of the [WP Roles](#) class.
- `$wp_meta_boxes` (array): Object containing all registered metaboxes, including their id's, args, callback functions and title for all post types including custom.
- `$wp_registered_sidebars` (array)
- `$wp_registered_widgets` (array)
- `$wp_registered_widget_controls` (array)
- `$wp_registered_widget_updates` (array)

Theme_mods vs. Options

```
$wp_customize->add_setting(
    'custom_theme_css' , array(
        'type'      => 'theme_mod',
    ) );

$wp_customize->add_setting(
    'custom_plugin_css' , array(
        'type'      => 'option',
    ) );
```

Custom CSS

Custom Theme CSS
Theme-specific: safely sticks with each theme when switching themes.

```
.site-content article {
    border: 18px solid #00f;
}
```

Custom Plugin CSS
Theme-agnostic: persists across theme changes.

```
.hentry .mejs-controls .mejs-time-rail .mejs-time-current {
    background: #6bf;
}
```

* This is not pseudo-code, this is it. Copied and pasted from the Modular Custom CSS Plugin.

🔗 Sanitization functions

There are many functions that will help you sanitize your data.

- `sanitize_email()`
- `sanitize_file_name()`
- `sanitize_hex_color()`
- `sanitize_hex_color_no_hash()`
- `sanitize_html_class()`
- `sanitize_key()`
- `sanitize_meta()`
- `sanitize_mime_type()`
- `sanitize_option()`
- `sanitize_sql_orderby()`
- `sanitize_term()`
- `sanitize_term_field()`
- `sanitize_text_field()`
- `sanitize_textarea_field()`
- `sanitize_title()`
- `sanitize_title_for_query()`
- `sanitize_title_with_dashes()`
- `sanitize_user()`
- `sanitize_url()`
- `wp_kses()`
- `wp_kses_post()`

Escaping with Localization

[Top ↑](#)

Rather than using `echo` to output data, it's common to use the WordPress localization functions, such as `_e()` or `__()`.

These functions simply wrap a localization function inside an escaping function:

[Copy](#)

```
esc_html_e( 'Hello World', 'text_domain' );
// Same as
echo esc_html( __( 'Hello World', 'text_domain' ) );
```

These helper functions combine localization and escaping:

- [`esc_html__\(\)`](#)
- [`esc_html_e\(\)`](#)
- [`esc_html_x\(\)`](#)
- [`esc_attr__\(\)`](#)
- [`esc_attr_e\(\)`](#)
- [`esc_attr_x\(\)`](#)

🔗 Escaping any numeric variable used anywhere

[Top ↑](#)

```
echo $int;
```

Depending on whether it is an integer or a float, `(int)`, `absint()`, `(float)` are all correct and acceptable.

At times, `number_format()` or `number_format_i18n()` might be more appropriate.

`intval()`, `floatval()` are acceptable, but are outdated (PHP4) functions.

🔗 Escaping arbitrary variable within HTML attribute

```
echo '<div id=""', $prefix, '-box', $id, '">';
```

Internationalization

Internationalization (often abbreviated as i18n) is the process of making your WordPress plugin translatable, allowing it to be easily translated into different languages. This is done using the built-in localization functions provided by WordPress. Here's a step-by-step guide on how to add internationalization to a WordPress plugin:

Prepare Your Plugin for Translation:

Before you can translate your plugin, you need to make sure it's ready for localization. This involves wrapping all translatable text in your plugin with translation functions. Replace any hardcoded strings with these functions.

Example:

// Hardcoded string (not translatable):

```
echo 'Hello, World!';
```

// Translatable string:

```
echo __('Hello, World!', 'your-plugin-text-domain');
```

Ensure that you replace all such strings with `__()` or `_e()` functions.

Load Your Text Domain:

In your plugin file (e.g., `your-plugin.php`), you need to load your plugin's text domain. This is typically done in the `init` or `plugins_loaded` action hook. The text domain should match the unique identifier of your plugin.

Example:

```
function load_plugin_textdomain() {  
    load_plugin_textdomain('your-plugin-text-domain', false,  
    dirname(plugin_basename(__FILE__)) . '/languages/');  
  
}  
  
add_action('init', 'load_plugin_textdomain');
```

In this example, replace `'your-plugin-text-domain'` with a unique identifier for your plugin. The third argument specifies the path to the directory where your translation files (`.mo` files) are stored.

Create Translation Files:

Next, you need to create translation files for your plugin. These files will contain translations for different languages. Use a tool like Poedit to create and manage translation files. Save these files with the `.po` extension.

Example file structure:

```
/your-plugin-directory/
└── your-plugin.php
    └── languages/
        ├── your-plugin-text-domain-en_US.po (English)
        └── your-plugin-text-domain-fr_FR.po (French)
```

Generate .mo Files:

After you've translated your strings in the `.po` files, you need to compile them into `.mo` files. Many translation tools, like Poedit, can do this for you. These `.mo` files contain the actual translated text.

Add Translations to Your Plugin:

Upload the compiled `.mo` files to the `languages/` directory of your plugin.

Use Translation Functions in Your Plugin:

Now that your plugin is set up for internationalization, use the translation functions (`_()` and `_e()`) to display text throughout your plugin.

Example:

```
// Display a translatable string:
echo __('Hello, World!', 'your-plugin-text-domain');
```

Create and Distribute Language Files:

Encourage users to translate your plugin into their languages using translation tools like GlotPress or by editing the `.po` files directly. You can also contribute your translations to the WordPress.org translation community.

By following these steps, you've made your WordPress plugin translatable and ready for localization, allowing users from different regions to use your plugin in their preferred language.

```
<?php
/*
Plugin Name: My Internationalization Plugin
Description: A simple internationalization example.
*/

// Load the plugin's text domain for translation.
function load_plugin_textdomain() {
    load_plugin_textdomain('my-i18n-plugin', false,
dirname(plugin_basename(__FILE__)) . '/languages/');
}
add_action('plugins_loaded', 'load_plugin_textdomain');

// Add a shortcode to display a translatable message.
function hello_world_shortcode() {
    return __('Hello, World!', 'my-i18n-plugin');
}
add_shortcode('hello_world', 'hello_world_shortcode');
```

- 1.Create a `languages` directory** inside your plugin folder to store translation files.
- 2.Create a `.pot` file for your plugin:** You can use a tool like Poedit to create a `.pot` file, which serves as a template for translations.
- 3.Generate translation files (`.po` and `.mo`):** Use Poedit to create translation files (`.po` and `.mo`) for different languages based on the `.pot` file. Save them in the `languages` directory you created.
- 4.Use the `hello_world` shortcode:** In your WordPress posts or pages, you can use the `[hello_world]` shortcode to display the "Hello, World!" message, which will be translatable into the user's language.

- 1.Activate it in your WordPress admin dashboard.
- 2.Create or upload translation files for your desired languages.
- 3.Use the `[hello_world]` shortcode in a post or page to display the translatable message.

Internationalization plugin code

```
<?php
/**
 * Plugin Name: Sample Plugin
 * Description: A sample WordPress plugin for internationalization.
 */

// Load the text domain for translation

function sample_plugin_load_textdomain() {
    load_plugin_textdomain('sample-plugin', false,
dirname(plugin_basename(__FILE__)) . '/languages/');
}

add_action('plugins_loaded', 'sample_plugin_load_textdomain');

// Add a sample filter that displays a translatable string

function sample_plugin_filter_content($content) {
    $translated_text = __('This is a sample plugin content.', 'sample-
plugin');
    return $content . '<p>' . $translated_text . '</p>';
}

add_filter('the_content', 'sample_plugin_filter_content');
```

Localization

Ajax localization in WordPress refers to the process of dynamically loading and displaying translated content on a web page using the Asynchronous JavaScript and XML (Ajax) technique. It allows you to change the language or region of a WordPress site without requiring a full page reload. This is particularly useful for creating multilingual websites where users can switch between languages without experiencing interruptions in their browsing experience.

Here are the key components and steps involved in implementing Ajax localization in a WordPress site:

1. Internationalization and Localization (i18n and l10n):

Before implementing Ajax localization, you need to ensure that your WordPress theme and plugins support internationalization and localization (i18n and l10n). This involves using functions like `__('text', 'textdomain')` for translatable strings and preparing translation files.

2. Language Switcher:

Create a user interface element, such as a dropdown menu or flags, that allows users to select their preferred language or region.

3. Ajax Requests:

Use JavaScript and Ajax to handle user interactions with the language switcher. When a user selects a different language, send an Ajax request to the server with the selected language as a parameter.

4. Server-Side Handling:

On the server side, handle the Ajax request by detecting the selected language and setting it for the current user's session or in a cookie. You can also store the selected language in the user's profile.

5. Translation Loading:

1. When a page is loaded or reloaded, check the user's language preference (from the session, cookie, or user profile) on the server side.

2.Load the appropriate translation files and set the WordPress locale to the user's selected language.

6. Dynamically Translate Content:

With the user's preferred language set, use WordPress localization functions like `__('text', 'textdomain')` in your theme and plugins to dynamically translate content based on the current language.

7. Update Content via Ajax:

When a user switches the language using the language switcher, send another Ajax request to the server with the selected language.

1.On the server side, update the user's language preference, load the appropriate translation files, and return the translated content.

2.Use JavaScript to replace or update the content on the page without requiring a full page reload.

8. Caching and Optimization:

Implement caching mechanisms to optimize performance. Caching translated content can reduce the server load and improve page load times.

9. Testing and Debugging:

Thoroughly test your Ajax localization implementation to ensure that content is translated correctly, and there are no issues with user experience or functionality.

Implementing Ajax localization in WordPress can be complex, and it requires a good understanding of both WordPress development and JavaScript programming. There are also WordPress plugins available that can assist with multilingual functionality, such as WPML (WordPress Multilingual Plugin) and Polylang, which offer built-in support for Ajax-based language switching. These plugins can simplify the implementation process for multilingual WordPress websites.

Any specific plugins or strategies for internationalization (i18n) and localization (l10n)?

Internationalization (i18n) and localization (l10n) are essential for making your WordPress website accessible to users from different regions and languages. Here are specific plugins and strategies for implementing i18n and l10n in WordPress:

1. Use the WordPress Built-in i18n/l10n Features:

- WordPress comes with built-in support for internationalization and localization. Utilize functions like `__()` and `_e()` for translating text, and use the `load_theme_textdomain()` and `load_plugin_textdomain()` functions to load translation files.

2. WPML (WordPress Multilingual Plugin):

- WPML is a popular and feature-rich plugin for creating multilingual WordPress websites. It allows you to translate posts, pages, custom post types, and even theme and plugin texts. WPML also provides features for language switching and SEO optimization.

3. Polylang:

- Polylang is a free and user-friendly multilingual plugin for WordPress. It enables you to create multilingual content, translate posts, pages, categories, tags, and more. Polylang is known for its simplicity and integration with popular page builders.

4. TranslatePress:

- TranslatePress is another user-friendly translation plugin. It provides a visual translation editor that allows you to translate content directly from the front end of your website. TranslatePress supports SEO-friendly URLs and multilingual SEO optimization.

5. Weglot:

- Weglot is a cloud-based translation service that offers a WordPress plugin for automatic translation of your website content. It's known for its ease of use and ability to translate content into multiple languages quickly.

6. Loco Translate:

- Loco Translate is a free and lightweight plugin that enables you to translate and manage translations of your theme and plugins directly from your WordPress dashboard. It's ideal for developers and site owners who want more control over translations.

7. Poedit:

- Poedit is a standalone translation software that is widely used by developers for creating and managing translation files (PO and MO files) for WordPress themes and plugins. You can use it in conjunction with gettext functions in your code.

8. GlotPress:

- GlotPress is an open-source translation management tool developed by the WordPress community. It's used for translating WordPress core, themes, and plugins. Some companies and projects host their own GlotPress installations for collaborative translation efforts.

9. Language Switcher Plugins:

- Consider using language switcher plugins like "Polylang for WooCommerce" or "WooCommerce Multilingual" (for e-commerce sites) to provide seamless language switching and currency conversion features.

10. RTL (Right-to-Left) Language Support: - For languages that are written from right to left (e.g., Arabic, Hebrew), ensure that your theme and content are compatible with RTL layouts. Many popular themes and plugins provide RTL support out of the box.

11. Custom Styles for Different Languages: - If your website uses CSS styles specific to languages or regions, consider using a plugin or custom code to load different stylesheets based on the selected language.

12. Translation Quality Assurance: - Use professional translators or translation services for critical content to ensure accuracy. Automated translations are convenient but may not always provide the desired quality.

13. SEO for Multilingual Sites: - Implement hreflang tags to inform search engines about the language and regional targeting of your pages. This helps with SEO and ensures the right content is displayed to users based on their language preferences.

Razorpay integration

Donation button example

domains/programmingwithvishal.com/public_html/demo/razorpay/index.php

```
1 <button id="rzp-button1">Pay</button>
2 <script src="https://checkout.razorpay.com/v1/checkout.js"></script>
3 <script>
4 var options = {
5   "key": "rzp_test_UY1y7bu0apmIK4", // Enter the Key ID generated from the Dashboard
6   "amount": "50000", // Amount is in currency subunits. Default currency is INR. Hence, 50000 refers to 50000 paise
7   "currency": "INR",
8   "name": "Acme Corp",
9   "description": "Test Transaction",
10  "image": "https://example.com/your_logo",
11  "handler": function (response){
12    console.log(response);
13  }
14 };
15 var rzp1 = new Razorpay(options);
16 document.getElementById('rzp-button1').onclick = function(e){
17   rzp1.open();
18   e.preventDefault();
19 }
20 </script>
```

Database Method Integration

[index.php](#)

```
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script src="https://checkout.razorpay.com/v1/checkout.js"></script>
<form>
  <input type="textbox" name="name" id="name" placeholder="Enter your
name"/><br/><br/>
  <input type="textbox" name="amt" id="amt" placeholder="Enter amt"/><br/><br/>
  <input type="button" name="btn" id="btn" value="Pay Now"
onclick="pay_now()"/>
```

```
</form>

<script>
    function pay_now(){
        var name=jQuery('#name').val();
        var amt=jQuery('#amt').val();

        jQuery.ajax({
            type:'post',
            url:'payment_process.php',
            data:"amt="+amt+"&name="+name,
            success:function(result){
                var options = {
                    "key": "key",
                    "amount": amt*100,
                    "currency": "INR",
                    "name": "Acme Corp",
                    "description": "Test Transaction",
                    "image": "https://image.freepik.com/free-vector/logo-sample-text_355-558.jpg",
                    "handler": function (response){
                        jQuery.ajax({
                            type:'post',
                            url:'payment_process.php',
                            data:"payment_id="+response.razorpay_payment_id,
                            success:function(result){
                                window.location.href="thank_you.php";
                            }
                        });
                    }
                };
                var rzp1 = new Razorpay(options);
                rzp1.open();
            }
        });
    }
</script>
```

payment_process.php

```
<?php
session_start();
include('db.php');
if(isset($_POST['amt']) && isset($_POST['name'])){
    $amt=$_POST['amt'];
    $name=$_POST['name'];
    $payment_status="pending";
    $added_on=date('Y-m-d h:i:s');
    mysqli_query($con,"insert into payment(name,amount,payment_status,added_on)
values('$name','$amt','$payment_status','$added_on')");
    $_SESSION['OID']=mysqli_insert_id($con);
}

if(isset($_POST['payment_id']) && isset($_SESSION['OID'])){
    $payment_id=$_POST['payment_id'];
    mysqli_query($con,"update payment set
payment_status='complete',payment_id='$payment_id' where
id='".$._SESSION['OID']."' ");
}
?>
```

db.php

```
<?php
$con=mysqli_connect('host','username','password','dbname');
?>
```

thankyou.php

```
<h1>Payment Complete</h1>
```

Woocommerce Payment Gateway

Step 3: Create the Main Plugin File

Inside your plugin directory, create a main PHP file (e.g., `custom-payment-gateway.php`) and add the following code:

```
php
<?php
/*
Plugin Name: Custom Payment Gateway
Description: Custom payment gateway integration for WooCommerce.
Version: 1.0
Author: Your Name
*/
// Add your custom payment gateway code here
?>
```

[Copy code](#)

 Reaene

Step 4: Add Payment Gateway Class

Inside your main plugin file, create a class for your payment gateway. This class should extend the `WC_Payment_Gateway` class provided by WooCommerce. Here's a simplified example:

```
class Custom_Payment_Gateway extends WC_Payment_Gateway {
    public function __construct() {
        $this->id = 'custom_payment';
        $this->icon = ''; // URL to an image for your gateway
        $this->method_title = 'Custom Payment Gateway';
        $this->method_description = 'Pay with Custom Payment Gateway';
        $this->has_fields = false;

        $this->init_settings();

        $this->title = $this->get_option('title');
```

```
$this->description = $this->get_option('description');

        add_action('woocommerce_update_options_payment_gateways_' . $this-
>id, array($this, 'process_admin_options'));
    }

public function init_form_fields() {
    $this->form_fields = array(
        'title' => array(
            'title' => 'Title',
            'type' => 'text',
            'description' => 'Title displayed to customers during
checkout.',
            'default' => 'Custom Payment'
        ),
        'description' => array(
            'title' => 'Description',
            'type' => 'textarea',
            'description' => 'Description displayed to customers during
checkout.',
            'default' => 'Pay securely with Custom Payment Gateway.'
        )
    );
}

public function process_payment($order_id) {
    // Handle payment processing here and return the result
}
}

function add_custom_payment_gateway($methods) {
    $methods[] = 'Custom_Payment_Gateway';
    return $methods;
}

add_filter('woocommerce_payment_gateways', 'add_custom_payment_gateway');
```

In this code:

- We create a class `Custom_Payment_Gateway` that extends `WC_Payment_Gateway`. Customize the class properties and methods as needed.
 - The `init_form_fields` method defines settings fields in WooCommerce admin.
 - The `process_payment` method handles payment processing logic.
 - `add_custom_payment_gateway` adds your custom gateway to WooCommerce.

Step 5: Activate the Plugin

Go to your WordPress admin dashboard, navigate to "Plugins," and activate your custom payment gateway plugin.

Step 6: Configure and Test

Custom plugin for payumoney

```
class Custom_PayUmoney_Gateway extends WC_Payment_Gateway {
    public function __construct() {
        $this->id = 'custom_payumoney';
        $this->icon = ''; // URL to an image for your gateway
        $this->method_title = 'Custom PayUmoney Gateway';
        $this->method_description = 'Pay with Custom PayUmoney Gateway';
        $this->has_fields = false;

        $this->init_settings();

        $this->title = $this->get_option('title');
        $this->description = $this->get_option('description');

        add_action('woocommerce_update_options_payment_gateways_' . $this->id, array($this, 'process_admin_options'));
    }

    public function init_form_fields() {
        $this->form_fields = array(
            'title' => array(
                'title' => 'Title',
                'desc' => 'The title of your payment method',
                'type' => 'text',
                'std' => ''
            )
        );
    }
}
```

```

        'type' => 'text',
        'description' => 'Title displayed to customers during
checkout.',

        'default' => 'Custom PayUmoney'
    ),
    'description' => array(
        'title' => 'Description',
        'type' => 'textarea',
        'description' => 'Description displayed to customers during
checkout.',

        'default' => 'Pay securely with Custom PayUmoney Gateway.'
),
    'merchant_key' => array(
        'title' => 'Merchant Key',
        'type' => 'text',
        'description' => 'Your PayUmoney merchant key.',
        'default' => ''
),
    'salt' => array(
        'title' => 'Salt',
        'type' => 'text',
        'description' => 'Your PayUmoney merchant salt.',
        'default' => ''
),
),
);
}

public function process_payment($order_id) {
    // Handle payment processing using the PayUmoney API here
}
}

function add_custom_payumoney_gateway($methods) {
    $methods[] = 'Custom_PayUmoney_Gateway';
    return $methods;
}

add_filter('woocommerce_payment_gateways', 'add_custom_payumoney_gateway');

```

Custom plugin for CCAvenue

```
class Custom_CCAvenue_Gateway extends WC_Payment_Gateway {  
    public function __construct() {  
        $this->id = 'custom_ccavenue';  
        $this->icon = ''; // URL to an image for your gateway  
        $this->method_title = 'Custom CCAvenue Gateway';  
        $this->method_description = 'Pay with Custom CCAvenue Gateway';  
        $this->has_fields = false;  
  
        $this->init_settings();  
  
        $this->title = $this->get_option('title');  
        $this->description = $this->get_option('description');  
  
        add_action('woocommerce_update_options_payment_gateways_' . $this->id, array($this, 'process_admin_options'));  
    }  
  
    public function init_form_fields() {  
        $this->form_fields = array(  
            'title' => array(  
                'title' => 'Title',  
                'type' => 'text',  
                'description' => 'Title displayed to customers during  
checkout.',  
                'default' => 'Custom CCAvenue'  
>,  
            'description' => array(  
                'title' => 'Description',  
                'type' => 'textarea',  
                'description' => 'Description displayed to customers during  
checkout.',  
                'default' => 'Pay securely with Custom CCAvenue Gateway.'  
>,  
            'merchant_id' => array(  
                'title' => 'Merchant ID',  
                'type' => 'text',  
                'description' => 'Your CCAvenue merchant ID.',  
                'default' => ''  
>,  
            'access_code' => array(  
                'title' => 'Access Code',  
                'type' => 'text',  
                'description' => 'Your CCAvenue access code.',  
                'default' => ''  
>);  
    }  
}
```

```
        ),
        'working_key' => array(
            'title' => 'Working Key',
            'type' => 'text',
            'description' => 'Your CCAvenue working key.',
            'default' => ''
        ),
    );
}

public function process_payment($order_id) {
    // Handle payment processing using the CCAvenue API here
}
}

function add_custom_ccavenue_gateway($methods) {
    $methods[] = 'Custom_CCAvenue_Gateway';
    return $methods;
}

add_filter('woocommerce_payment_gateways', 'add_custom_ccavenue_gateway');
```

Wordpress installation on AWS

1. **Create an AWS Account:** If you don't already have an AWS account, sign up for one at <https://aws.amazon.com/>. You will need to provide billing information, but you can use the AWS Free Tier to get started without incurring charges for the first 12 months.

2. **Launch an EC2 Instance:**

- Sign in to your AWS Management Console.
- Navigate to the EC2 Dashboard.
- Click the "Launch Instance" button to create a new virtual server.
- Choose an **Amazon Machine Image (AMI)** based on your requirements. You can use an Amazon Linux AMI, which is suitable for most purposes.
- Select an instance type, which determines the server's hardware resources. **The t2.micro instance is part of the Free Tier.**
- Configure instance details, such as the number of instances, network settings, and storage.
- Add storage for your instance. You can use the default settings or adjust the storage size as needed.
- Configure security groups to control incoming traffic to your instance. At a minimum, allow **SSH (port 22)** and **HTTP/HTTPS (ports 80 and 443)**.
- Review and launch the instance, and create a new key pair or use an existing one to securely connect to your instance.

3. **Connect to Your EC2 Instance:**

- Once the instance is running, **use SSH to connect to it. Use the private key** from the key pair you selected during instance launch.
- The command to connect will look like this:

```
css
```

 Copy code

```
ssh -i /path/to/your-key.pem ec2-user@your-instance-ip
```

4. Install LAMP Stack:

- After connecting to your EC2 instance, you can install the LAMP (Linux, Apache, MySQL, PHP) stack to host WordPress.
- Run the following commands to install Apache, MySQL, and PHP:

bash

 Copy code

```
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
sudo yum install mysql-server -y
sudo systemctl start mysqld
sudo systemctl enable mysqld
sudo yum install php php-mysql -y
```

5. Secure MySQL:

- Run the MySQL secure installation script to set a root password and secure your MySQL installation.

 Copy code

```
sudo mysql_secure_installation
```

6. Download and Configure WordPress:

- Download the latest WordPress release and extract it in the Apache web server's document root:

bash

 Copy code

```
cd /var/www/html
sudo wget https://wordpress.org/latest.tar.gz
sudo tar -xzf latest.tar.gz
```

6. Download and Configure WordPress:

- Download the latest WordPress release and extract it in the Apache web server's document root:

```
bash
```

 Copy code

```
cd /var/www/html  
sudo wget https://wordpress.org/latest.tar.gz  
sudo tar -xzf latest.tar.gz
```

7. Create a MySQL Database for WordPress:

- Log in to MySQL and create a new database and user for WordPress.

```
css
```

 Copy code

```
mysql -u root -p
```

Enter your MySQL root password, then run the following SQL commands:

```
sql
```

 Copy code

```
CREATE DATABASE wordpress;  
CREATE USER 'wordpressuser'@'localhost' IDENTIFIED BY 'password';  
GRANT ALL PRIVILEGES ON wordpress.* TO 'wordpressuser'@'localhost';  
FLUSH PRIVILEGES;  
EXIT;
```

 Regen

8. Configure WordPress:

- Rename the `wp-config-sample.php` file to `wp-config.php` and edit it to provide your database details:

```
arduino
```

 Copy code

```
cd /var/www/html/wordpress  
sudo mv wp-config-sample.php wp-config.php  
sudo nano wp-config.php
```

Update the database name, user, and password you created earlier.

9. Access WordPress Installation:

- Open a web browser and enter your instance's public IP address or domain name. You should see the WordPress installation wizard.
- Complete the setup by providing your site title, username, password, and email address.

10. Complete WordPress Setup:

- After the setup, you can log in to your WordPress admin dashboard and start customizing your site.

11. Additional Configuration:

- You may want to configure your domain name, set up SSL/TLS certificates, and enhance security. AWS offers services like Route 53 and ACM for domain management and SSL certificates.

12. Regular Backups and Updates:

- Ensure you regularly back up your WordPress site and keep both your WordPress and server software up to date to maintain security and performance.

This is a basic guide to installing WordPress on AWS. Depending on your specific requirements, you may need to configure additional services, such as an S3 bucket for media storage or CloudFront for content delivery. Always refer to the AWS documentation for the most up-to-date information and best practices.

 Reg

LAMP Setup

<https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-20-04>

Step 1 – Installing Apache and Updating the Firewall

The Apache web server is among the most popular web servers in the world. It's well documented, has an active community of users, and has been in wide use for much of the history of the web, which makes it a great choice for hosting a website.

Start by updating the package manager cache. If this is the first time you're using `sudo` within this session, you'll be prompted to provide your user's password to confirm you have the right privileges to manage system packages with `apt`.

```
$ sudo apt update
```

[Copy](#)

Then, install Apache with:

```
$ sudo apt install apache2
```

[Copy](#)

You'll also be prompted to confirm Apache's installation by pressing `Y`, then `ENTER`.

Once the installation is finished, you'll need to adjust your firewall settings to allow HTTP traffic. UFW has different application profiles that you can leverage for accomplishing that. To list all currently available UFW application profiles, you can run:

```
$ sudo ufw app list
```

[Copy](#)

You'll see output like this:

```
Output
Available applications:
Apache
Apache Full
Apache Secure
OpenSSH
```

Here's what each of these profiles mean:

- **Apache**: This profile opens only port `80` (normal, unencrypted web traffic).
- **Apache Full**: This profile opens both port `80` (normal, unencrypted web traffic) and port `443` (TLS/SSL encrypted traffic).
- **Apache Secure**: This profile opens only port `443` (TLS/SSL encrypted traffic).

For now, it's best to allow only connections on port `80`, since this is a fresh Apache installation and you still don't have a TLS/SSL certificate configured to allow for HTTPS traffic on your server.

To only allow traffic on port `80`, use the `Apache` profile:

```
$ sudo ufw allow in "Apache"
```

Copy

You can verify the change with:

```
$ sudo ufw status
```

Copy

Output

Status: active

To	Action	From
--	-----	-----
OpenSSH	ALLOW	Anywhere
Apache	ALLOW	Anywhere
OpenSSH (v6)	ALLOW	Anywhere (v6)
Apache (v6)	ALLOW	Anywhere (v6)

Traffic on port `80` is now allowed through the firewall.

You can do a spot check right away to verify that everything went as planned by visiting your server's public IP address in your web browser (see the note under the next heading to find out what your public IP address is if you do not have this information already):

```
http://your_server_ip
```

You'll see the default Ubuntu 20.04 Apache web page, which is there for informational and testing purposes. It should look something like this:



Apache2 Ubuntu Default Page

ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   '-- ports.conf
|-- mods-enabled
|   '-- *.load
|       '-- *.conf
|-- conf-enabled
|   '-- *.conf
|-- sites-enabled
|   '-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective `*-available/` counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf`. See their respective man pages for detailed information.
- The binary is called `apache2`. Due to the use of environment variables, in the default configuration, `apache2` needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

Document Roots

By default, Ubuntu does not allow access through the web browser to *any* file apart of those located in `/var/www`, `public_html` directories (when enabled) and `/usr/share` (for web applications). If your site is using a web document root located elsewhere (such as in `/srv`) you may need to whitelist your document root directory in `/etc/apache2/apache2.conf`.

The default Ubuntu document root is `/var/www/html`. You can make your own virtual hosts under `/var/www`. This is different to previous releases which provides better security out of the box.

How To Find your Server's Public IP Address

If you do not know what your server's public IP address is, there are a number of ways you can find it. Usually, this is the address you use to connect to your server through SSH.

There are a few different ways to do this from the command line. First, you could use the `iproute2` tools to get your IP address by typing this:

```
$ ip addr show eth0 | grep inet | awk '{ print $2; }' | sed 's/\.*$//'
```

Copy

This will give you two or three lines back. They are all correct addresses, but your computer may only be able to use one of them, so feel free to try each one.

An alternative method is to use the `curl` utility to contact an outside party to tell you how *it* sees your server. This is done by asking a specific server what your IP address is:

```
$ curl http://icanhazip.com
```

Copy

Regardless of the method you use to get your IP address, type it into your web browser's address bar to view the default Apache page.

Step 2 – Installing MySQL

Now that you have a web server up and running, you need to install the database system to be able to store and manage data for your site. MySQL is a popular database management system used within PHP environments.

Again, use `apt` to acquire and install this software:

```
$ sudo apt install mysql-server
```

Copy

When prompted, confirm installation by typing `y`, and then `ENTER`.

When the installation is finished, it's recommended that you run a security script that comes pre-installed with MySQL. This script will remove some insecure default settings and lock down access to your database system. Start the interactive script by running:

```
$ sudo mysql_secure_installation
```

Copy

This will ask if you want to configure the `VALIDATE PASSWORD PLUGIN`.

Note: Enabling this feature is something of a judgment call. If enabled, passwords which don't match the specified criteria will be rejected by MySQL with an error. It is safe to leave validation disabled, but you should always use strong, unique passwords for database credentials.

Answer **y** for yes, or anything else to continue without enabling.

VALIDATE PASSWORD PLUGIN can be used to test passwords and improve security. It checks the strength of password and allows the users to set only those passwords which are secure enough. Would you like to setup VALIDATE PASSWORD plugin?

Press **y|Y** for Yes, any other key for No:

If you answer "yes", you'll be asked to select a level of password validation. Keep in mind that if you enter **2** for the strongest level, you will receive errors when attempting to set any password which does not contain numbers, upper and lowercase letters, and special characters, or which is based on common dictionary words.

There are three levels of password validation policy:

LOW Length >= 8
MEDIUM Length >= 8, numeric, mixed case, and special characters
STRONG Length >= 8, numeric, mixed case, special characters and dictionary file

Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: **1**

Regardless of whether you chose to set up the **VALIDATE PASSWORD PLUGIN**, your server will next ask you to select and confirm a password for the MySQL **root** user. This is not to be confused with the **system root**. The **database root** user is an administrative user with full privileges over the database system. Even though the default authentication method for the MySQL root user dispenses the use of a password, **even when one is set**, you should define a strong password here as an additional safety measure. We'll talk about this in a moment.

If you enabled password validation, you'll be shown the password strength for the root password you just entered and your server will ask if you want to continue with that password. If you are happy with your current password, enter **y** for "yes" at the prompt:

Estimated strength of the password: **100**

Do you wish to continue with the password provided?(Press **y|Y** for Yes, any other key for No) : **y**

For the rest of the questions, press `Y` and hit the `ENTER` key at each prompt. This will remove some anonymous users and the test database, disable remote root logins, and load these new rules so that MySQL immediately respects the changes you have made.

When you're finished, test if you're able to log in to the MySQL console by typing:

```
$ sudo mysql
```

[Copy](#)

This will connect to the MySQL server as the administrative database user `root`, which is inferred by the use of `sudo` when running this command. You should see output like this:

Output

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 22
Server version: 8.0.19-0ubuntu5 (Ubuntu)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

To exit the MySQL console, type:

```
mysql> exit
```

[Copy](#)

Notice that you didn't need to provide a password to connect as the `root` user, even though you have defined one when running the `mysql_secure_installation` script. That is because the default authentication method for the administrative MySQL user is `unix_socket` instead of `password`. Even though this might look like a security concern at first, it makes the database server more secure because the only users allowed to log in as the `root` MySQL user are the system users with `sudo` privileges connecting from the console or through an application running with the same privileges. In practical terms, that means you won't be able to use the administrative database `root` user to connect from your PHP application. Setting a password for the `root` MySQL account works as a safeguard, in case the default authentication method is changed from `unix_socket` to `password`.

For increased security, it's best to have dedicated user accounts with less expansive privileges set up for every database, especially if you plan on having multiple databases hosted on your server.

Step 3 – Installing PHP

You have Apache installed to serve your content and MySQL installed to store and manage your data. PHP is the component of our setup that will process code to display dynamic content to the final user. In addition to the `php` package, you'll need `php-mysql`, a PHP module that allows PHP to communicate with MySQL-based databases. You'll also need `libapache2-mod-php` to enable Apache to handle PHP files. Core PHP packages will automatically be installed as dependencies.

To install these packages, run:

```
$ sudo apt install php libapache2-mod-php php-mysql
```

Copy

Once the installation is finished, you can run the following command to confirm your PHP version:

```
$ php -v
```

Copy

Output

```
PHP 7.4.3 (cli) (built: Jul 5 2021 15:13:35) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
    with Zend OPcache v7.4.3, Copyright (c), by Zend Technologies
```

At this point, your LAMP stack is fully operational, but before you can test your setup with a PHP script, it's best to set up a proper [Apache Virtual Host](#) to hold your website's files and folders. We'll do that in the next step.

Step 4 – Creating a Virtual Host for your Website

When using the Apache web server, you can create *virtual hosts* (similar to server blocks in Nginx) to encapsulate configuration details and host more than one domain from a single server. In this guide, we'll set up a domain called **your_domain**, but you should **replace this with your own domain name**.

Note: In case you are using DigitalOcean as DNS hosting provider, you can check our [product docs](#) for detailed instructions on how to set up a new domain name and point it to your server.

Apache on Ubuntu 20.04 has one server block enabled by default that is configured to serve documents from the `/var/www/html` directory. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying `/var/www/html`, we'll create a directory structure within `/var/www` for the **your_domain** site, leaving `/var/www/html` in place as the default directory to be served if a client request doesn't match any other sites.

Create the directory for **your_domain** as follows:

```
$ sudo mkdir /var/www/your_domain
```

Copy

Next, assign ownership of the directory with the `$USER` environment variable, which will reference your current system user:

```
$ sudo chown -R $USER:$USER /var/www/your_domain
```

Copy

Then, open a new configuration file in Apache's `sites-available` directory using your preferred command-line editor. Here, we'll use `nano`:

```
$ sudo nano /etc/apache2/sites-available/your_domain.conf
```

Copy

This will create a new blank file. Paste in the following bare-bones configuration:

```
/etc/apache2/sites-available/your_domain.conf
```

```
<VirtualHost *:80>
    ServerName your_domain
    ServerAlias www.your_domain
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/your_domain
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Save and close the file when you're done. If you're using `nano`, you can do that by pressing `CTRL+X`, then `Y` and `ENTER`.

With this `VirtualHost` configuration, we're telling Apache to serve `your_domain` using `/var/www/your_domain` as the web root directory. If you'd like to test Apache without a domain name, you can remove or comment out the options `ServerName` and `ServerAlias` by adding a `#` character in the beginning of each option's lines.

You can now use `a2ensite` to enable the new virtual host:

```
$ sudo a2ensite your_domain
```

Copy

You might want to disable the default website that comes installed with Apache. This is required if you're not using a custom domain name, because in this case Apache's default configuration would overwrite your virtual host. To disable Apache's default website, type:

```
$ sudo a2dissite 000-default
```

Copy

To make sure your configuration file doesn't contain syntax errors, run:

```
$ sudo apache2ctl configtest
```

Copy

Finally, reload Apache so these changes take effect:

```
$ sudo systemctl reload apache2
```

Copy

Your new website is now active, but the web root `/var/www/your_domain` is still empty. Create an `index.html` file in that location so that we can test that the virtual host works as expected:

```
$ nano /var/www/your_domain/index.html
```

Copy

Include the following content in this file:

```
/var/www/your_domain/index.html
```

```
<html>
  <head>
    <title>your_domain website</title>
  </head>
  <body>
    <h1>Hello World!</h1>

    <p>This is the landing page of <strong>your_domain</strong>.</p>
  </body>
</html>
```

Now go to your browser and access your server's domain name or IP address once again:

```
http://server_domain_or_IP
```

You'll see a page like this:

Hello World!

This is the landing page of **your_domain**.

If you see this page, it means your Apache virtual host is working as expected.

You can leave this file in place as a temporary landing page for your application until you set up an `index.php` file to replace it. Once you do that, remember to remove or rename the `index.html` file from your document root, as it would take precedence over an `index.php` file by default.

A Note About `DirectoryIndex` on Apache

With the default `DirectoryIndex` settings on Apache, a file named `index.html` will always take precedence over an `index.php` file. This is useful for setting up maintenance pages in PHP applications, by creating a temporary `index.html` file containing an informative message to visitors. Because this page will take precedence over the `index.php` page, it will then become the landing page for the application. Once maintenance is over, the `index.html` is renamed or removed from the document root, bringing back the regular application page.

In case you want to change this behavior, you'll need to edit the `/etc/apache2/mods-enabled/dir.conf` file and modify the order in which the `index.php` file is listed within the `DirectoryIndex` directive:

```
$ sudo nano /etc/apache2/mods-enabled/dir.conf
```

Copy

/etc/apache2/mods-enabled/dir.conf

```
<IfModule mod_dir.c>
    DirectoryIndex index.php index.html index.cgi index.pl index.xhtml index.htm
</IfModule>
```

After saving and closing the file, you'll need to reload Apache so the changes take effect:

```
$ sudo systemctl reload apache2
```

Copy

In the next step, we'll create a PHP script to test that PHP is correctly installed and configured on your server.

Step 5 – Testing PHP Processing on your Web Server

Now that you have a custom location to host your website's files and folders, we'll create a PHP test script to confirm that Apache is able to handle and process requests for PHP files.

Create a new file named `info.php` inside your custom web root folder:

```
$ nano /var/www/your_domain/info.php
```

Copy

This will open a blank file. Add the following text, which is valid PHP code, inside the file:

```
/var/www/your_domain/info.php
```

```
<?php  
phpinfo();
```

Copy

When you are finished, save and close the file.

To test this script, go to your web browser and access your server's domain name or IP address, followed by the script name, which in this case is `info.php`:

```
http://server_domain_or_IP/info.php
```

You'll see a page similar to this:

PHP Version 7.4.3	
System	Linux sassy-starfish 5.4.0-26-generic #30-Ubuntu SMP Mon Apr 20 16:58:30 UTC 2020 x86_64
Build Date	Mar 26 2020 20:24:23
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.4/apache2
Loaded Configuration File	/etc/php/7.4/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.4/apache2/conf.d
Additional .ini files parsed	/etc/php/7.4/apache2/conf.d/10-mysqld.ini, /etc/php/7.4/apache2/conf.d/10-opcache.ini, /etc/php/7.4/apache2/conf.d/10-pdo.ini, /etc/php/7.4/apache2/conf.d/20-calendar.ini, /etc/php/7.4/apache2/conf.d/20-ctype.ini, /etc/php/7.4/apache2/conf.d/20-exif.ini, /etc/php/7.4/apache2/conf.d/20-fil.ini, /etc/php/7.4/apache2/conf.d/20-gettext.ini, /etc/php/7.4/apache2/conf.d/20-iconv.ini, /etc/php/7.4/apache2/conf.d/20-json.ini, /etc/php/7.4/apache2/conf.d/20-mysqli.ini, /etc/php/7.4/apache2/conf.d/20-posix.ini, /etc/php/7.4/apache2/conf.d/20-readline.ini, /etc/php/7.4/apache2/conf.d/20-shmop.ini, /etc/php/7.4/apache2/conf.d/20-sockets.ini, /etc/php/7.4/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.4/apache2/conf.d/20-sysvshm.ini, /etc/php/7.4/apache2/conf.d/20-tokenizer.ini
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902,NTS
PHP Extension Build	API20190902,NTS
Debug Build	no

Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ttps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.4.0, Copyright (c) Zend Technologies
with Zend OPcache v7.4.3, Copyright (c), by Zend Technologies



This page provides information about your server from the perspective of PHP. It is useful for debugging and to ensure that your settings are being applied correctly.

If you can see this page in your browser, then your PHP installation is working as expected.

After checking the relevant information about your PHP server through that page, it's best to remove the file you created as it contains sensitive information about your PHP environment and your Ubuntu server. You can use `rm` to do so:

```
$ sudo rm /var/www/your_domain/info.php
```

Copy

You can always recreate this page if you need to access the information again later.

To create a new database, run the following command from your MySQL console:

```
mysql> CREATE DATABASE example_database;
```

Copy

Now you can create a new user and grant them full privileges on the custom database you've just created.

The following command creates a new user named `example_user`, using `mysql_native_password` as default authentication method. We're defining this user's password as `password`, but you should replace this value with a secure password of your own choosing.

```
mysql> CREATE USER 'example_user'@'%' IDENTIFIED WITH mysql_native_password BY 'password';
```

Copy

Now we need to give this user permission over the `example_database` database:

```
mysql> GRANT ALL ON example_database.* TO 'example_user'@'%';
```

Copy

This will give the `example_user` user full privileges over the `example_database` database, while preventing this user from creating or modifying other databases on your server.

Now exit the MySQL shell with:

```
mysql> exit
```

Copy

You can test if the new user has the proper permissions by logging in to the MySQL console again, this time using the custom user credentials:

```
$ mysql -u example_user -p
```

Copy

Notice the `-p` flag in this command, which will prompt you for the password used when creating the **example_user** user. After logging in to the MySQL console, confirm that you have access to the **example_database** database:

```
mysql> SHOW DATABASES;
```

Copy

This will give you the following output:

Output

Database
example_database
information_schema

Step 1 – Creating a MySQL Database and User for WordPress

The first step that we will take is a preparatory one. WordPress uses MySQL to manage and store site and user information. We have MySQL installed already, but we need to make a database and a user for WordPress to use.

To get started, log into the MySQL root (administrative) account by issuing this command (note that this is not the root user of your server):

```
$ mysql -u root -p
```

Copy

You will be prompted for the password you set for the MySQL root account when you installed the software.

Note: If you cannot access your MySQL database via root, as a `sudo` user you can update your root user's password by logging into the database like so:

```
$ sudo mysql -u root
```

Copy

Once you receive the MySQL prompt, you can update the root user's password. Here, replace `new_password` with a strong password of your choosing.

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'new_p' Copy
```

You may now type `EXIT;` and can log back into the database via password with the following command:

```
$ mysql -u root -p
```

Copy

Within the database, we can create an exclusive database for WordPress to control. You can call this whatever you would like, but we will be using the name `wordpress` in this guide. Create the database for WordPress by typing:

```
mysql> CREATE DATABASE wordpress DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Copy

Note: Every MySQL statement must end in a semi-colon (;). Check to make sure this is present if you are running into any issues.

Next, we are going to create a separate MySQL user account that we will use exclusively to operate our new database. Creating specific databases and accounts can support us from a management and security standpoint. We will use the name **wordpressuser** in this guide, but feel free to use whatever name is relevant for you.

We are going to create this account, set a password, and grant access to the database we created. We can do this by typing the following command. Remember to choose a strong password here for your database user where we have **password**:

```
mysql> CREATE USER 'wordpressuser'@'%' IDENTIFIED WITH mysql_native_password BY 'password'; Copy
```

Next, let the database know that our **wordpressuser** should have complete access to the database we set up:

```
mysql> GRANT ALL ON wordpress.* TO 'wordpressuser'@'%'; Copy
```

You now have a database and user account, each made specifically for WordPress. We need to flush the privileges so that the current instance of MySQL knows about the recent changes we've made:

```
mysql> FLUSH PRIVILEGES; Copy
```

Exit out of MySQL by typing:

```
mysql> EXIT; Copy
```

In the next step, we'll lay some foundations for WordPress plugins by downloading PHP extensions for our server.

Step 2 – Installing Additional PHP Extensions

When setting up our LAMP stack, we only required a very minimal set of extensions in order to get PHP to communicate with MySQL. WordPress and many of its plugins leverage additional PHP extensions.

We can download and install some of the most popular PHP extensions for use with WordPress by typing:

```
$ sudo apt update
$ sudo apt install php-curl php-gd php-mbstring php-xml php-xmlrpc php-soap php-intl php-zip Copy
```

This will lay the groundwork for installing additional plugins into our WordPress site.

Note: Each WordPress plugin has its own set of requirements. Some may require additional PHP packages to be installed. Check your plugin documentation to discover its PHP requirements. If they are available, they can be installed with `apt` as demonstrated above.

We will need to restart Apache to load these new extensions, we'll be doing more configurations on Apache in the next section, so you can wait until then, or restart now to complete the PHP extension process.

```
$ sudo systemctl restart apache2
```

Copy

Step 3 – Adjusting Apache’s Configuration to Allow for .htaccess Overrides and Rewrites

Next, we will be making a few minor adjustments to our Apache configuration. Based on the prerequisite tutorials, you should have a configuration file for your site in the `/etc/apache2/sites-available/` directory.

In this guide, we'll use `/etc/apache2/sites-available/wordpress.conf` as an example here, but you should substitute the path to your configuration file where appropriate. Additionally, we will use `/var/www/wordpress` as the root directory of our WordPress install. You should use the web root specified in your own configuration. If you followed our [LAMP tutorial](#), it may be your domain name instead of `wordpress` in both of these instances.

Note: It's possible you are using the `000-default.conf` default configuration (with `/var/www/html` as your web root). This is fine to use if you're only going to host one website on this server. If not, it's better to split the necessary configuration into logical chunks, one file per site.

With our paths identified, we can move onto working with `.htaccess` so that Apache can handle configuration changes on a per-directory basis.

Enabling .htaccess Overrides

Currently, the use of `.htaccess` files is disabled. WordPress and many WordPress plugins use these files extensively for in-directory tweaks to the web server's behavior.

Open the Apache configuration file for your website with a text editor like nano.

```
$ sudo nano /etc/apache2/sites-available/wordpress.conf
```

Copy

To allow `.htaccess` files, we need to set the `AllowOverride` directive within a `Directory` block pointing to our document root. Add the following block of text inside the `VirtualHost` block in your configuration file, making sure to use the correct web root directory:

```
/etc/apache2/sites-available/wordpress.conf
```

```
<Directory /var/www/wordpress/>
    AllowOverride All
</Directory>
```

When you are finished, save and close the file. In nano, you can do this by pressing `CTRL` and `X` together, then `Y`, then `ENTER`.

Enabling the Rewrite Module

Next, we can enable `mod_rewrite` so that we can utilize the WordPress permalink feature:

```
$ sudo a2enmod rewrite
```

Copy

This allows you to have more human-readable permalinks to your posts, like the following two examples:

```
http://example.com/2012/post-name/
http://example.com/2012/12/30/post-name
```

The `a2enmod` command calls a script that enables the specified module within the Apache configuration.

Enabling the Changes

Before we implement the changes we've made, check to make sure we haven't made any syntax errors by running the following test.

```
$ sudo apache2ctl configtest
```

Copy

You may receive output like the following:

Output

```
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1
Syntax OK
```

If you wish to suppress the top line, just add a `ServerName` directive to your main (global) Apache configuration file at `/etc/apache2/apache2.conf`. The `ServerName` can be your server's domain or IP address. This is just a message, however, and doesn't affect the functionality of your site. As long as the output contains `Syntax OK`, you are ready to continue.

Restart Apache to implement the changes. Make sure to restart now even if you have restarted earlier in this tutorial.

```
$ sudo systemctl restart apache2
```

[Copy](#)

Next, we will download and set up WordPress itself.

Step 4 – Downloading WordPress

Now that our server software is configured, we can download and set up WordPress. For security reasons in particular, it is always recommended to get the latest version of WordPress from their site.

Change into a writable directory (we recommend a temporary one like `/tmp`) and download the compressed release.

```
$ cd /tmp
$ curl -O https://wordpress.org/latest.tar.gz
```

[Copy](#)

Extract the compressed file to create the WordPress directory structure:

```
$ tar xzvf latest.tar.gz
```

[Copy](#)

We will be moving these files into our document root momentarily. Before we do, we can add a dummy `.htaccess` file so that this will be available for WordPress to use later.

Create the file by typing:

```
$ touch /tmp/wordpress/.htaccess
```

[Copy](#)

We'll also copy over the sample configuration file to the filename that WordPress reads:

```
$ cp /tmp/wordpress/wp-config-sample.php /tmp/wordpress/wp-config.php
```

[Copy](#)

We can also create the `upgrade` directory, so that WordPress won't run into permissions issues when trying to do this on its own following an update to its software:

```
$ mkdir /tmp/wordpress/wp-content/upgrade
```

[Copy](#)

Now, we can copy the entire contents of the directory into our document root. We are using a dot at the end of our source directory to indicate that everything within the directory should be copied, including hidden files (like the `.htaccess` file we created):

```
$ sudo cp -a /tmp/wordpress/. /var/www/wordpress
```

[Copy](#)

Ensure that you replace the `/var/www/wordpress` directory with the directory you have set up on your server.

Step 5 – Configuring the WordPress Directory

Before we do the web-based WordPress setup, we need to adjust some items in our WordPress directory.

Adjusting the Ownership and Permissions

An important step that we need to accomplish is setting up reasonable file permissions and ownership.

We'll start by giving ownership of all the files to the `www-data` user and group. This is the user that the Apache web server runs as, and Apache will need to be able to read and write WordPress files in order to serve the website and perform automatic updates.

Update the ownership with the `chown` command which allows you to modify file ownership. Be sure to point to your server's relevant directory.

```
$ sudo chown -R www-data:www-data /var/www/wordpress
```

[Copy](#)

Next we'll run two `find` commands to set the correct permissions on the WordPress directories and files:

```
$ sudo find /var/www/wordpress/ -type d -exec chmod 750 {} \;
$ sudo find /var/www/wordpress/ -type f -exec chmod 640 {} \;
```

[Copy](#)

These permissions should get you working effectively with WordPress, but note that some plugins and procedures may require additional tweaks.

Setting Up the WordPress Configuration File

Now, we need to make some changes to the main WordPress configuration file.

When we open the file, our first task will be to adjust some secret keys to provide a level of security for our installation. WordPress provides a secure generator for these values so that you do not have to try to come up with good values on your own. These are only used internally, so it won't hurt usability to have complex, secure values here.

To grab secure values from the WordPress secret key generator, type:

```
$ curl -s https://api.wordpress.org/secret-key/1.1/salt/
```

Copy

You will get back unique values that resemble output similar to the block below.

Warning! It is important that you request unique values each time. Do **NOT** copy the values below!

Output

```
define('AUTH_KEY',         '1jl/vqfs<XhdXoAPz9 DO NOT COPY THESE VALUES c_j{iwqD^<+c9.k<J@4H');  
define('SECURE_AUTH_KEY', 'E2N-h2]Dcvpt+aS/p7X DO NOT COPY THESE VALUES {Ka(f;rv?Pxf})CgLi-3');  
define('LOGGED_IN_KEY',   'W(50,{W^,OPB%PB<JF DO NOT COPY THESE VALUES 2;y&,2m%3]R6DUth[;88');  
define('NONCE_KEY',       'l1,4UC)7ua+8<!4VM+ DO NOT COPY THESE VALUES # DXF+[atzM7 o^-C7g');  
define('AUTH_SALT',        'koMrurzOA+|L_1G}kf DO NOT COPY THESE VALUES 07VC*Lj*1D&?3w!BT#-');  
define('SECURE_AUTH_SALT', 'p32*p,]z%LZ+pAu:VY DO NOT COPY THESE VALUES C-?y+K0DK_+F|0h{!_xY');  
define('LOGGED_IN_SALT',   'i^/G2W7!-1H20Q+t$3 DO NOT COPY THESE VALUES t6**bRVFSD[Hi])-qS`|');  
define('NONCE_SALT',       'Q6]U:K?j4L%Z}]h^q7 DO NOT COPY THESE VALUES 1% ^qUswIggn+6&xqHN8%');
```

These are configuration lines that we can paste directly in our configuration file to set secure keys.

These are configuration lines that we can paste directly in our configuration file to set secure keys.
Copy the output you received now.

Next, open the WordPress configuration file:

```
$ sudo nano /var/www/wordpress/wp-config.php
```

Copy

Find the section that contains the example values for those settings.

```
/var/www/wordpress/wp-config.php
```

```
...  
  
define('AUTH_KEY',         'put your unique phrase here');  
define('SECURE_AUTH_KEY',  'put your unique phrase here');  
define('LOGGED_IN_KEY',    'put your unique phrase here');  
define('NONCE_KEY',        'put your unique phrase here');  
define('AUTH_SALT',        'put your unique phrase here');  
define('SECURE_AUTH_SALT', 'put your unique phrase here');  
define('LOGGED_IN_SALT',   'put your unique phrase here');  
define('NONCE_SALT',       'put your unique phrase here');
```

Delete those lines and paste in the values you copied from the command line:

```
/var/www/wordpress/wp-config.php
```

```
...  
  
define('AUTH_KEY',         'VALUES COPIED FROM THE COMMAND LINE');  
define('SECURE_AUTH_KEY',  'VALUES COPIED FROM THE COMMAND LINE');  
define('LOGGED_IN_KEY',    'VALUES COPIED FROM THE COMMAND LINE');  
define('NONCE_KEY',        'VALUES COPIED FROM THE COMMAND LINE');  
define('AUTH_SALT',        'VALUES COPIED FROM THE COMMAND LINE');  
define('SECURE_AUTH_SALT', 'VALUES COPIED FROM THE COMMAND LINE');  
define('LOGGED_IN_SALT',   'VALUES COPIED FROM THE COMMAND LINE');  
define('NONCE_SALT',       'VALUES COPIED FROM THE COMMAND LINE');
```

Next, we are going to modify some of the database connection settings at the beginning of the file. You need to adjust the database name, the database user, and the associated password that you configured within MySQL.

The other change we need to make is to set the method that WordPress should use to write to the filesystem. Since we've given the web server permission to write where it needs to, we can explicitly set the filesystem method to "direct". Failure to set this with our current settings would result in WordPress prompting for FTP credentials when we perform some actions.

This setting can be added below the database connection settings, or anywhere else in the file:

```
/var/www/wordpress/wp-config.php

. . .

// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'wordpress' );

/** MySQL database username */
define( 'DB_USER', 'wordpressuser' );

/** MySQL database password */
define( 'DB_PASSWORD', 'password' );

/** MySQL hostname */
define( 'DB_HOST', 'localhost' );

/** Database Charset to use in creating database tables. */
define( 'DB_CHARSET', 'utf8' );

/** The Database Collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', '' );

. . .

define('FS_METHOD', 'direct');
```

Save and close the file when you are finished.

Step 6 – Completing the Installation Through the Web Interface

Now that the server configuration is complete, we can complete the installation through the web interface.

In your web browser, navigate to your server's domain name or public IP address:

```
https://server_domain_or_IP
```

Select the language you would like to use:



Next, you will come to the main setup page.

Select a name for your WordPress site and choose a username. It is recommended to choose something unique and avoid common usernames like "admin" for security purposes. A strong password is generated automatically. Save this password or select an alternative strong password.

Enter your email address and select whether you want to discourage search engines from indexing your site:



Welcome

Welcome to the famous five-minute WordPress Installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title	Example
Username	myuser
Password	Z0pkm0IG9vHZ7Gf&F Strong
Your Email	admin@example.com
Search Engine Visibility	<input type="checkbox"/> Discourage search engines from indexing this site It is up to search engines to honor this request.

[Install WordPress](#)

When you click ahead, you will be taken to a page that prompts you to log in:

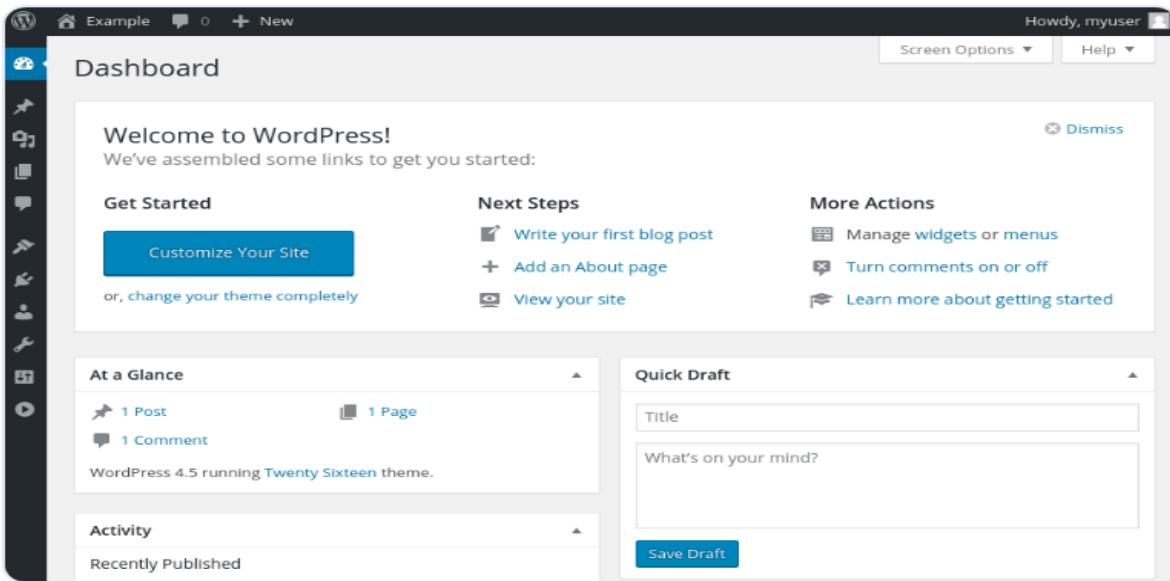
Success!

WordPress has been installed. Thank you, and enjoy!

Username	myuser
Password	<i>Your chosen password.</i>

[Log In](#)

Once you log in, you will be taken to the WordPress administration dashboard:



At this point, you can begin to design your WordPress website! If this is your first time using WordPress, explore the interface a bit to get acquainted with your new CMS.

Conclusion

Congratulations, WordPress is now installed and is ready to be used!

At this point you may want to start doing the following:

- Choose your permalinks setting for WordPress posts, which can be found in [Settings > Permalinks](#).
- Select a new theme in [Appearance > Themes](#).
- Install new plugins to increase your site's functionality under [Plugins > Add New](#).
- If you are going to collaborate with others, you may also wish to add additional users at this time under [Users > Add New](#).

You can find additional resources for alternate ways to install WordPress, learn how to install WordPress on different server distributions, automate your WordPress installations, and scale your WordPress sites by checking out our [WordPress Community tag](#).

Initial Server Setup with Ubuntu 20.04

Step 1 – Logging in as root

To log into your server, you will need to know your **server's public IP address**. You will also need the password or — if you installed an SSH key for authentication — the private key for the **root** user's account. If you have not already logged into your server, you may want to follow our guide on [how to Connect to Droplets with SSH](#), which covers this process in detail.

If you are not already connected to your server, log in now as the **root** user using the following command (substitute the highlighted portion of the command with your server's public IP address):

```
$ ssh root@your_server_ip
```

Copy

Accept the warning about host authenticity if it appears. If you are using password authentication, provide your **root** password to log in. If you are using an SSH key that is passphrase protected, you may be prompted to enter the passphrase the first time you use the key each session. If this is your first time logging into the server with a password, you may also be prompted to change the **root** password.

About root

The **root** user is the administrative user in a Linux environment that has very broad privileges. Because of the heightened privileges of the **root** account, you are *discouraged* from using it on a regular basis. This is because the **root** account is able to make very destructive changes, even by accident.

The next step is setting up a new user account with reduced privileges for day-to-day use. Later, we'll show you how to temporarily gain increased privileges for the times when you need them.

Step 2 – Creating a New User

Once you are logged in as **root**, you'll be able to add the new user account. In the future, we'll log in with this new account instead of **root**.

This example creates a new user called **sammy**, but you should replace that with a username that you like:

```
# adduser sammy
```

Copy

You will be asked a few questions, starting with the account password.

Enter a strong password and, optionally, fill in any of the additional information if you would like. This is not required and you can just hit **ENTER** in any field you wish to skip.

Step 3 – Granting Administrative Privileges

Now we have a new user account with regular account privileges. However, we may sometimes need to do administrative tasks.

To avoid having to log out of our normal user and log back in as the **root** account, we can set up what is known as *superuser* or **root** privileges for our normal account. This will allow our normal user to run commands with administrative privileges by putting the word `sudo` before the command.

To add these privileges to our new user, we need to add the user to the **sudo** group. By default, on Ubuntu 20.04, users who are members of the **sudo** group are allowed to use the `sudo` command.

As **root**, run this command to add your new user to the **sudo** group (substitute the highlighted username with your new user):

```
# usermod -aG sudo sammy
```

Copy

Now, when logged in as your regular user, you can type `sudo` before commands to run them with superuser privileges.

Step 4 – Setting Up a Basic Firewall

Ubuntu 20.04 servers can use the UFW firewall to make sure only connections to certain services are allowed. We can set up a basic firewall using this application.

Note: If your servers are running on DigitalOcean, you can optionally use [DigitalOcean Cloud Firewalls](#) instead of the UFW firewall. We recommend using only one firewall at a time to avoid conflicting rules that may be difficult to debug.

Applications can register their profiles with UFW upon installation. These profiles allow UFW to manage these applications by name. OpenSSH, the service allowing us to connect to our server now, has a profile registered with UFW.

You can see this by typing:

```
# ufw app list
```

Copy

Output

Available applications:
OpenSSH

We need to make sure that the firewall allows SSH connections so that we can log back in next time. We can allow these connections by typing:

```
# ufw allow OpenSSH
```

Copy

Afterwards, we can enable the firewall by typing:

```
# ufw enable
```

Copy

Type `y` and press `ENTER` to proceed. You can see that SSH connections are still allowed by typing:

```
# ufw status
```

Copy

Output

Status: active

To	Action	From
--	-----	-----
OpenSSH	ALLOW	Anywhere
OpenSSH (v6)	ALLOW	Anywhere (v6)

As **the firewall is currently blocking all connections except for SSH**, if you install and configure additional services, you will need to adjust the firewall settings to allow traffic in. You can learn some common UFW operations in our [UFW Essentials guide](#).

Step 5 – Enabling External Access for Your Regular User

Now that we have a regular user for daily use, we need to make sure we can SSH into the account directly.

Note: Until verifying that you can log in and use `sudo` with your new user, we recommend staying logged in as **root**. This way, if you have problems, you can troubleshoot and make any necessary changes as **root**. If you are using a DigitalOcean Droplet and experience problems with your **root** SSH connection, you can [regain access to Droplets using the Recovery Console](#).

The process for configuring SSH access for your new user depends on whether your server's **root** account uses a password or SSH keys for authentication.

If the root Account Uses Password Authentication

If you logged in to your **root** account *using a password*, then password authentication is *enabled* for SSH. You can SSH to your new user account by opening up a new terminal session and using SSH with your new username:

```
$ ssh sammy@your_server_ip
```

Copy

After entering your regular user's password, you will be logged in. Remember, if you need to run a command with administrative privileges, type `sudo` before it like this:

```
$ sudo command_to_run
```

Copy

You will be prompted for your regular user password when using `sudo` for the first time each session (and periodically afterwards).

To enhance your server's security, **we strongly recommend setting up SSH keys instead of using password authentication.** Follow our guide on [setting up SSH keys on Ubuntu 20.04](#) to learn how to configure key-based authentication.

If the root Account Uses SSH Key Authentication

If you logged in to your `root` account *using SSH keys*, then password authentication is *disabled* for SSH. You will need to add a copy of your local public key to the new user's `~/.ssh/authorized_keys` file to log in successfully.

Since your public key is already in the `root` account's `~/.ssh/authorized_keys` file on the server, we can copy that file and directory structure to our new user account in our existing session.

The simplest way to copy the files with the correct ownership and permissions is with the `rsync` command. This will copy the `root` user's `.ssh` directory, preserve the permissions, and modify the file owners, all in a single command. Make sure to change the highlighted portions of the command below to match your regular user's name:

Note: The `rsync` command treats sources and destinations that end with a trailing slash differently than those without a trailing slash. When using `rsync` below, be sure that the source directory (`~/.ssh`) **does not** include a trailing slash (check to make sure you are not using `~/ssh/`).

If you accidentally add a trailing slash to the command, `rsync` will copy the *contents* of the `root` account's `~/.ssh` directory to the `sudo` user's home directory instead of copying the entire `~/.ssh` directory structure. The files will be in the wrong location and SSH will not be able to find and use them.

```
# rsync --archive --chown=sammy:sammy ~/.ssh /home/sammy
```

Copy

Now, open up a new terminal session on your local machine, and use SSH with your new username:

```
$ ssh sammy@your_server_ip
```

Copy

You should be logged in to the new user account without using a password. Remember, if you need to run a command with administrative privileges, type `sudo` before it like this:

```
$ sudo command_to_run
```

Copy

Install wordpress on window iis server

Step 1: Download WordPress

1. Go to the official WordPress website (<https://wordpress.org/download/>) and download the latest version of WordPress in ZIP format.

Step 2: Extract the WordPress Files

1. Extract the contents of the ZIP file you downloaded to a directory on your Windows server where you want to host your WordPress site. This will be your website's root directory (e.g., `C:\inetpub\wwwroot\mywordpress`).

Step 3: Create a MySQL Database

1. You'll need a MySQL database to store WordPress data. Install MySQL on your Windows server if you haven't already.
2. Create a new MySQL database for WordPress. You can use a MySQL client like phpMyAdmin or the MySQL command line to create the database.
3. Create a MySQL user and grant it privileges on the WordPress database. Note down the database name, username, and password for later use.

Step 4: Configure PHP on IIS

1. Install PHP on your Windows server. You can use a package like XAMPP or install PHP manually.
2. Configure PHP to work with IIS by adding PHP as a FastCGI module. You can use the "Internet Information Services (IIS) Manager" to configure this.

Step 5: Configure WordPress

1. Rename the `wp-config-sample.php` file in your WordPress directory to `wp-config.php`.
2. Open `wp-config.php` and configure the database settings by entering the database name, username, and password you created earlier:

```
php
```

 Copy code

```
define('DB_NAME', 'your_database_name');
define('DB_USER', 'your_database_user');
define('DB_PASSWORD', 'your_database_password');
```

3. You may also want to define other constants like the authentication keys and salts. You can generate these keys by visiting the WordPress.org secret key generator (<https://api.wordpress.org/secret-key/1.1/salt/>).
4. Save your `wp-config.php` file.

Step 6: Configure IIS for WordPress

1. Open the "Internet Information Services (IIS) Manager."
2. In the Connections pane on the left, expand your server node and click on "Sites."
3. Right-click on "Default Web Site" or create a new site if needed, and select "Add Application."
4. Set an alias for your WordPress site (e.g., "mywordpress") and browse for the physical path where you extracted the WordPress files.
5. In the "Edit Application" dialog, go to the "Modules" section, and ensure that the "FastCgiModule" is installed and enabled.

Step 7: Access Your WordPress Site

1. Open a web browser and enter the URL to your WordPress site (e.g., `http://localhost/mywordpress`). Follow the WordPress installation instructions.
2. Complete the installation by providing your site title, username, password, and email address.
3. Once the installation is complete, you can log in to the WordPress admin dashboard and start customizing your site.

That's it! You've successfully installed WordPress on your Windows IIS server. You can now start building and managing your website using the WordPress platform.

CRUD Oops Vishal programming

```
<?php
class database{
    private $host;
    private $dbusername;
    private $dbpassword;
    private $dbname;

    protected function connect(){
        $this->host='localhost';
        $this->dbusername='wwwabcx_abcuser';
        $this->dbpassword='&;=mbh));6@{';
        $this->dbname='wwwabcx_oops_crud_vishal';
        $con=new mysqli($this->host,$this->dbusername,$this->dbpassword,$this-
>dbname);
        return $con;
    }
}

class query extends database{
    public function
getData($table,$field='*',$condition_arr='', $order_by_field='', $order_by_type='de
sc', $limit ''){
    $sql="select $field from $table ";
    if($condition_arr!=''){
        $sql.=' where ';
        $c=count($condition_arr);
        $i=1;
        foreach($condition_arr as $key=>$val){
            if($i==$c){
                $sql.="$key='$val'";
            }else{
                $sql.="$key='$val' and ";
            }
            $i++;
        }
    }
    if($order_by_field!=''){
        $sql.=" order by $order_by_field $order_by_type ";
    }

    if($limit!= ''){
        $sql.=" limit $limit ";
    }
}
```

```

$result=$this->connect()->query($sql);
if($result->num_rows>0){
    $arr=array();
    while($row=$result->fetch_assoc()){
        $arr[]=$row;
    }
    return $arr;
}else{
    return 0;
}

}

public function insertData($table,$condition_arr){
    if($condition_arr!=""){
        foreach($condition_arr as $key=>$val){
            $fieldArr[]=$key;
            $valueArr[]=$val;
        }
        $field=implode(", ",$fieldArr);
        $value=implode(", '",$valueArr);
        $value="'. $value .'";
        $sql="insert into $table($field) values($value) ";
        $result=$this->connect()->query($sql);
    }
}

public function deleteData($table,$condition_arr){
    if($condition_arr!=""){
        $sql="delete from $table where ";
        $c=count($condition_arr);
        $i=1;
        foreach($condition_arr as $key=>$val){
            if($i==$c){
                $sql.="$key='$val'";
            }else{
                $sql.="$key='$val' and ";
            }
            $i++;
        }
        $result=$this->connect()->query($sql);
    }
}

public function updateData($table,$condition_arr,$where_field,$where_value){
    if($condition_arr!=""){
        $sql="update $table set ";

```

```

$c=count($condition_arr);
$i=1;
foreach($condition_arr as $key=>$val){
    if($i==$c){
        $sql.="$key='$val'";
    }else{
        $sql.="$key='$val', ";
    }
    $i++;
}
$sql.=" where $where_field='$where_value' ";
$result=$this->connect()->query($sql);
}

public function get_safe_str($str){
    if($str!=''){
        return mysqli_real_escape_string($this->connect(),$str);
    }
}
?>

```

Database.php

```

1 <?php
2 class database{
3     private $host;
4     private $dbusername;
5     private $dbpassword;
6     private $dbname;
7
8     protected function connect(){
9         $this->host='localhost';
10        $this->dbusername='wwwabcx_abcuser';
11        $this->dbpassword='&;=mbh));6@{';
12        $this->dbname='wwwabcx_oops_crud_vishal';
13        $con=new mysqli($this->host,$this->dbusername,$this->dbpassword,$this
14             ->dbname);
15        return $con;
16    }
17 }

```

```
17
18 class query extends database{
19     public function getData($table,$field='*',$condition_arr='', $order_by_field
20 = '',$order_by_type='desc',$limit=''){
21         $sql="select $field from $table ";
22         if($condition_arr!=""){
23             $sql.=' where ';
24             $c=count($condition_arr);
25             $i=1;
26             foreach($condition_arr as $key=>$val){
27                 if($i==$c){
28                     $sql.="$key='$val'";
29                 }else{
30                     $sql.="$key='$val' and ";
31                 }
32                 $i++;
33             }
34         if($order_by_field!=""){
35             $sql.=" order by $order_by_field $order_by_type ";
36         }
37
38         if($limit!=""){
39             $sql.=" limit $limit ";
40         }
41         //die($sql);
42         $result=$this->connect()->query($sql);
43         if($result->num_rows>0){
44             $arr=array();
45             while($row=$result->fetch_assoc()){
46                 $arr[]=$row;
47             }
48             return $arr;
49         }else{
50             return 0;
51         }
52     }
53 }
```

```
54-
55-     public function insertData($table,$condition_arr){
56-         if($condition_arr!=""){
57-             foreach($condition_arr as $key=>$val){
58-                 $fieldArr[]=$key;
59-                 $valueArr[]=$val;
60-             }
61-             $field=implode(",",$fieldArr);
62-             $value=implode(",",$valueArr);
63-             $value="".$value."'";
64-             $sql="insert into $table($field) values($value) ";
65-             $result=$this->connect()->query($sql);
66-         }
67-
68-     public function deleteData($table,$condition_arr){
69-         if($condition_arr!=""){
70-             $sql="delete from $table where ";
71-             $c=count($condition_arr);
72-             $i=1;
73-             foreach($condition_arr as $key=>$val){
74-                 if($i==$c){
75-                     $sql.="$key='$val'";
76-                 }else{
77-                     $sql.="$key='$val' and ";
78-                 }
79-                 $i++;
80-             }
81-             $result=$this->connect()->query($sql);
82-         }
83-     }
84-
85-     public function updateData($table,$condition_arr,$where_field,$where_value){
86-         if($condition_arr!=""){
87-             $sql="update $table set ";
88-             $c=count($condition_arr);
89-             $i=1;
90-             foreach($condition_arr as $key=>$val){
91-                 if($i==$c){
92-                     $sql.="$key='$val'";
93-                 }else{
94-                     $sql.="$key='$val', ";
95-                 }
96-                 $i++;
97-             }
98-             $sql.=" where $where_field='$where_value' ";
99-             $result=$this->connect()->query($sql);
100-        }
101-    }
102-
103-    public function get_safe_str($str){
104-        if($str!=""){
105-            return mysqli_real_escape_string($this->connect(),$str);
106-        }
107-    }
108-}
109-
```

Users.php

```
1 <?php
2 include('database.php');
3 $obj=new query();
4
5 $name='';
6 $email='';
7 $mobile='';
8 $id='';
9
10 if(isset($_GET['id']) && $_GET['id']!=''){
11     $id=$obj->get_safe_str($_GET['id']);
12     $condition_arr=array('id'=>$id);
13     $result=$obj->getData('user','*', $condition_arr);
14     $name=$result['0']['name'];
15     $email=$result['0']['email'];
16     $mobile=$result['0']['mobile'];
17 }
18
19 if(isset($_POST['submit'])){
20     $name=$obj->get_safe_str($_POST['name']);
21     $email=$obj->get_safe_str($_POST['email']);
22     $mobile=$obj->get_safe_str($_POST['mobile']);
23
24     $condition_arr=array('name'=>$name,'email'=>$email,'mobile'=>$mobile);
25
26 if($id==''){
27     $obj->insertData('user',$condition_arr);
28 }else{
29     $obj->updateData('user',$condition_arr,'id',$id);
30 }
31
32 //header('location:users.php');
33 ?>
34 <script>
```

```

34      <script>
35      window.location.href='users.php';
36      </script>
37      <?php
38  }
39 ?>
40 <!doctype html>
41 <html lang="en-US">
42   <head>
43     <meta charset="UTF-8">
44     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
45     <title>Manage User - PHP Object Oriented Programming CRUD</title>
46     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css">
47     <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.3/css/all.css">
48     <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
49 <!--[if lt IE 9]>
50 <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js">
51 </script>
52 <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
53 <![endif]-->
54   <style>
55     .container{margin-top:100px;}
56   </style>
57 </head>
<body>
58
59   <div class="container">
60     <div class="card">
61       <div class="card-header"><i class="fa fa-fw fa-plus-circle"></i>
62         <strong>Add User</strong> <a href="users.php" class="float-right btn btn-dark btn-sm"><i class="fa fa-fw fa-globe"></i> Browse Users</a></div>
63     <div class="card-body">
64       <div class="col-sm-6">
65         <h5 class="card-title">Fields with <span class="text-danger">*</span> are mandatory!</h5>
66         <form method="post">
67           <div class="form-group">
68             <label>Name <span class="text-danger">*</span></label>
69             <input type="text" name="name" id="name" class="form-control" placeholder="Enter name" required value="<?php echo $name?>">
70           </div>
71           <div class="form-group">
72             <label>Email <span class="text-danger">*</span></label>
73             <input type="email" name="email" id="email" class="form-control" placeholder="Enter email" required value="<?php echo $email?>">
74           </div>
75           <div class="form-group">
76             <label>Mobile <span class="text-danger">*</span></label>
77             <input type="tel" class="tel form-control" name="mobile" id="mobile" placeholder="Enter mobile" required value="<?php echo $mobile?>">
78           </div>
79           <div class="form-group">
80             <button type="submit" name="submit" value="submit" id="submit" class="btn btn-primary"><i class="fa fa-fw fa-plus-circle"></i> Manage User</button>
81           </div>
82         </form>
83       </div>
84     </div>

```

Index.php

```
1  <?php
2  include('database.php');
3  $obj=new query();
4
5  if(isset($_GET['type']) && $_GET['type']=='delete'){
6      $id=$obj->get_safe_str($_GET['id']);
7      $condition_arr=array('id'=>$id);
8      $obj->deleteData('user',$condition_arr);
9  }
10
11 $result=$obj->getData('user','*','','id','desc');
12 ?>
13 <!doctype html>
14 <html lang="en-US">
15   <head>
16     <meta charset="UTF-8">
17     <meta name="viewport" content="width=device-width, initial-scale=1, shrink
18       -to-fit=no">
19     <title>User Listing - PHP Object Oriented Programming CRUD</title>
20     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap
21       /4.2.1/css/bootstrap.min.css">
22     <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.3
23       /css/all.css">
24     <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media
25       queries -->
26     <!--[if lt IE 9]>
27     <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"
28       ></script>
29     <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script
30       >
31     <![endif]-->
32     <style>
33       .container{margin-top:100px;}
34     </style>
35   </head>
36   <body>
37     <div class="container">
38       <div class="card">
39         <div class="card-header"><i class="fa fa-fw fa-globe"></i> <strong
40           >Browse User</strong> <a href="manage-users.php" class="float
41             -right btn btn-dark btn-sm"><i class="fa fa-fw fa-plus-circle"
42               ></i> Add Users</a></div>
43     </div>
```

```
34         ></i> Add Users</a></div>
35     </div>
36     <hr>
37     <div>
38         <table class="table table-striped table-bordered">
39             <thead>
40                 <tr class="bg-primary text-white">
41                     <th>Sr#</th>
42                     <th>Name</th>
43                     <th>Email</th>
44                     <th>Mobile</th>
45                     <th class="text-center">Action</th>
46                 </tr>
47             </thead>
48             <tbody>
49                 <?php
50                     if(isset($result['0'])){
51                         $id=1;
52                         foreach($result as $list){
53                             ?>
54                             <tr>
55                                 <td><?php echo $id?></td>
56                                 <td><?php echo $list['name']?></td>
57                                 <td><?php echo $list['email']?></td>
58                                 <td><?php echo $list['mobile']?></td>
59                                 <td align="center">
60                                     <a href="manage-users.php?id=<?php echo $list['id']?>" class="text-primary"><i class="fa fa-fw fa-edit"></i> Edit</a> |
61                                     <a href="?type=delete&id=<?php echo $list['id']?>" class="text-danger"><i class="fa fa-fw fa-trash"></i> Delete</a>
62                                 </td>
63                             </tr>
64                             <?php
65                             $id++;
66                             } } else {?>
67                             <tr>
68                                 <td colspan="6" align="center">No Records Found!</td>
69                             </tr>
70                         <?php } ?>
71                     </tbody>
72                 </table>
73             </div>
74             <!--.col-sm-12-->
```

Yahoo baba CRUD OOPs

```
1 <?php
2
3 v class Database{
4
5     private $db_host = "localhost";
6     private $db_user = "wwwabcx_abouser";
7     private $db_pass = "&;=mbh));6@{";
8     private $db_name = "wwwabcx_crud_yahoobaba";
9
10    private $mysqli = "";
11    private $result = array();
12    private $conn = false;
13
14 v     public function __construct(){
15     if(!$this->conn){
16         $this->mysqli = new mysqli($this->db_host,$this->db_user,$this
17             ->db_pass,$this->db_name);
18         $this->conn = true;
19         if($this->mysqli->connect_error){
20             array_push($this->result, $this->mysqli->connect_error);
21             return false;
22         }
23     }else{
24         return true;
25     }
26
27     // Function to insert into the database
28 v     public function insert($table,$params=array()){
29         // Check to see if the table exists
30 v         if($this->tableExists($table)){
31             // Seperate $params's Array KEYs and VALUEs and Convert them to String
32             // Value
33             $table_columns = implode(', ', array_keys($params));
34             $table_value = implode("', '", $params);
35
36             $sql = "INSERT INTO $table ($table_columns) VALUES ('$table_value')";
37             // Make the query to insert to the database
38             if($this->mysqli->query($sql)){
39                 array_push($this->result, $this->mysqli->insert_id);
40                 return true; // The data has been inserted
41             }else{
42                 array_push($this->result, $this->mysqli->error);
43                 return false; // The data has not been inserted
44             }
45         }else{
46             return false; // Table does not exist
47         }
48     }
49 }
```

```
50 // Function to update row in database
51 public function update($table,$params=array(),$where = null){
52     // Check to see if table exists
53     if($this->tableExists($table)){
54         // Create Array to hold all the columns to update
55         $args = array();
56         foreach ($params as $key => $value) {
57             $args[] = "$key = '$value'"; // Separate each column out with it's
58             corresponding value
59         }
60
61         $sql = "UPDATE $table SET " . implode(', ', $args);
62         if($where != null){
63             $sql .= " WHERE $where";
64         }
65         // Make query to database
66         if($this->mysqli->query($sql)){
67             array_push($this->result, $this->mysqli->affected_rows);
68             return true; // Update has been successful
69         }else{
70             array_push($this->result, $this->mysqli->error);
71             return false; // Update has not been successful
72         }
73     }else{
74         return false; // The table does not exist
75     }
76
77 //Function to delete table or row(s) from database
78 public function delete($table,$where = null){
79     // Check to see if table exists
80     if($this->tableExists($table)){
81         $sql = "DELETE FROM $table"; // Create query to delete rows
82         if($where != null){
83             $sql .= " WHERE $where";
84         }
85         // Submit query to database
86         if($this->mysqli->query($sql)){
87             array_push($this->result, $this->mysqli->affected_rows);
88             return true; // The query executed correctly
89         }else{
90             array_push($this->result, $this->mysqli->error);
91             return false; // The query did not execute correctly
92         }
93     }else{
94         return false; // The table does not exist
95     }
96 }
97 }
98 }
```

```
99 // Function to SELECT from the database
100 public function select($table, $rows="*", $join = null, $where = null, $order
101   = null, $limit=null){
102   // Check to see if the table exists
103   if($this->tableExists($table)){
104     // Create query from the variables passed to the function
105     $sql = "SELECT $rows FROM $table";
106     if($join != null){
107       $sql .= " JOIN $join";
108     }
109     if($where != null){
110       $sql .= " WHERE $where";
111     }
112     if($order != null){
113       $sql .= " ORDER BY $order";
114     }
115     if($limit != null){
116       if(isset($_GET['page'])){
117         $page = $_GET['page'];
118       }else{
119         $page = 1;
120       }
121       $start = ($page - 1) * $limit;
122       $sql .= " LIMIT $start,$limit";
123     }
124     $query = $this->mysqli->query($sql);
125
126     if($query){
127       $this->result = $query->fetch_all(MYSQLI_ASSOC);
128       return true; // Query was successful
129     }else{
130       array_push($this->result, $this->mysqli->error);
131       return false; // No rows were returned
132     }
133   }else{
134     return false; // Table does not exist
135   }
136 }
```

```
138 // FUNCTION to show Pagination
139 public function pagination($table,$join = null,$where = null,$limit=null){
140     // Check to see if table exists
141     if($this->tableExists($table)){
142         if($limit != null){
143             // select count() query for pagination
144             $sql = "SELECT COUNT(*) FROM $table";
145             if($join != null){
146                 $sql .= " JOIN $join";
147             }
148             if($where != null){
149                 $sql .= " WHERE $where";
150             }
151
152             $query = $this->mysqli->query($sql);
153
154             $total_record = $query->fetch_array();
155             $total_record = $total_record[0];
156
157             $total_page = ceil($total_record / $limit);
158
159             $url = basename($_SERVER['PHP_SELF']);
160             // Get the Page Number which is set in URL
161             if(isset($_GET['page'])){
162                 $page = $_GET['page'];
163             }else{
164                 $page = 1;
165             }
166             // show pagination
167             $output = "<ul class='pagination'>";
168     }
```

```
    if($page>1){
        $output .= "<li><a href='".$url?page=".($page-1)."'>Prev</a></li>";
    }

    if($total_record > $limit){
        for($i = 1; $i <= $total_page; $i++){
            if($i == $page){
                $cls = "class='active'";
            }else{
                $cls = "";
            }
            $output .= "<li><a $cls href='".$url?page=$i'>$i</a></li>";
        }
        if($total_page>$page){
            $output .= "<li><a href='".$url?page=".($page+1)."'>Next</a></li>";
        }
        $output .= "</ul>";

        echo $output;
    }else{
        return false; // If Limit is null
    }
}else{
    return false; // Table does not exist
}

198 public function sql($sql){
199     $query = $this->mysqli->query($sql);
200
201     if($query){
202         $this->result = $query->fetch_all(MYSQLI_ASSOC);
203         return true; // Query was successful
204     }else{
205         array_push($this->result, $this->mysqli->error);
206         return false; // No rows were returned
207     }
208 }
209
210 // Private function to check if table exists for use with queries
211 private function tableExists($table){
212     $sql = "SHOW TABLES FROM $this->db_name LIKE '$table'";
213     $tableInDb = $this->mysqli->query($sql);
214     if($tableInDb){
215         if($tableInDb->num_rows == 1){
216             return true; // The table exists
217         }else{
218             array_push($this->result,$table." does not exist in this database.");
219         }
220     }
221 }
222 }
```

```
223
224 // Public function to return the data to the user
225 public function getResult(){
226     $val = $this->result;
227     $this->result = array();
228     return $val;
229 }
230
231 // close connection
232 public function __destruct(){
233     if($this->conn){
234         if($this->mysqli->close()){
235             $this->conn = false;
236             return true;
237         }
238     }else{
239         return false;
240     }
241 }
242
243 } //Class Close
244
245
246 ?>
```

CRUD Using Oops (Chat GPT)

```
<?php

// Database configuration
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

// Create database connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Book class
class Book {
    private $id;
    private $title;
    private $author;

    public function __construct($title, $author) {
        $this->title = $title;
        $this->author = $author;
    }

    public function getId() {
        return $this->id;
    }

    public function getTitle() {
        return $this->title;
    }
}
```

```
}

public function getAuthor() {
    return $this->author;
}

public function save() {
    global $conn;
    $stmt = $conn->prepare("INSERT INTO books (title, author) VALUES (?, ?)");
    $stmt->bind_param("ss", $this->title, $this->author);
    $stmt->execute();
    $this->id = $stmt->insert_id;
    $stmt->close();
}

public function update() {
    global $conn;
    $stmt = $conn->prepare("UPDATE books SET title = ?, author = ? WHERE id = ?");
    $stmt->bind_param("ssi", $this->title, $this->author, $this->id);
    $stmt->execute();
    $stmt->close();
}

public function delete() {
    global $conn;
    $stmt = $conn->prepare("DELETE FROM books WHERE id = ?");
    $stmt->bind_param("i", $this->id);
    $stmt->execute();
    $stmt->close();
}
}
```

```
// Example usage:  
// Create a new book  
$newBook = new Book("Sample Book", "John Doe");  
$newBook->save();  
echo "New book created with ID: " . $newBook->getId() . "<br>";  
  
// Update the book  
$newBook->setTitle("Updated Book");  
$newBook->setAuthor("Jane Smith");  
$newBook->update();  
echo "Book updated<br>";  
  
// Delete the book  
$newBook->delete();  
echo "Book deleted<br>";  
  
// Close database connection  
$conn->close();  
?>
```

CRED Operation Wordpress

Step 1: Create a new MySQL table (you can do this via phpMyAdmin or any MySQL client):

Step 2: Create a new folder for your plugin. For this example, we'll name it "crud-plugin."

Step 3: Create the main plugin file (crud-plugin.php) inside the "crud-plugin" folder:

```
<?php  
/*  
Plugin Name: CRUD Plugin  
Description: A simple plugin for CRUD operations.  
Version: 1.0  
Author: Your Name  
*/  
  
// Create the database table on plugin activation  
function crud_plugin_create_table() {  
    global $wpdb;  
    $table_name = $wpdb->prefix . 'custom_table';  
  
$sql = "CREATE TABLE $table_name (  
    id INT NOT NULL AUTO_INCREMENT,  
    name VARCHAR(255) NOT NULL,  
    email VARCHAR(255) NOT NULL,  
    phone VARCHAR(20),  
PRIMARY KEY (id )";
```

```
require_once(ABSPATH . 'wp-admin/includes/upgrade.php');

dbDelta($sql);

}

register_activation_hook(__FILE__, 'crud_plugin_create_table');
```

Step 4: Create a file to handle the CRUD operations (crud-operations.php) inside the "crud-plugin" folder:

```
<?php

// Add new record

function crud_plugin_add_record($name, $email, $phone) {

global $wpdb;

$table_name = $wpdb->prefix . 'custom_table';

$data = array(
    'name' => $name,
    'email' => $email,
    'phone' => $phone,
);

$wpdb->insert($table_name, $data);

}

// Update record by ID

function crud_plugin_update_record($id, $name, $email, $phone) {

global $wpdb;

$table_name = $wpdb->prefix . 'custom_table';
```

```

$data = array(
    'name' => $name,
    'email' => $email,
    'phone' => $phone,
);

$where = array('id' => $id);

$wpdb->update($table_name, $data, $where);

}

```

```

// Delete record by ID

function crud_plugin_delete_record($id) {

    global $wpdb;

    $table_name = $wpdb->prefix . 'custom_table';

    $where = array('id' => $id);

    $wpdb->delete($table_name, $where);

}

```

Step 5: Create a form in your main plugin file (crud-plugin.php) to handle user input:

```

<?php

// Include the file with CRUD operations functions

require_once(plugin_dir_path(__FILE__) . 'crud-operations.php');

// Handle form submission

if (isset($_POST['submit'])) {

    $name = sanitize_text_field($_POST['name']);

```

```
$email = sanitize_email($_POST['email']);

$phone = sanitize_text_field($_POST['phone']);

// Add new record

crud_plugin_add_record($name, $email, $phone);

}

?>

<!-- Display the form -->

<form method="post">

<label for="name">Name:</label> <input type="text" name="name" required>

<label for="email">Email:</label> <input type="email" name="email" required>

<label for="phone">Phone:</label> <input type="text" name="phone">

<input type="submit" name="submit" value="Submit">

</form>
```

CRUD application wordpress plugin refrence chatGPT

```
<?php

/**
 * Plugin Name: Book Manager
 * Description: A simple CRUD application for managing books.
 * Version: 1.0.0
 * Author: Your Name
 */

// Activation hook

register_activation_hook(__FILE__, 'book_manager_activate');

function book_manager_activate() {
    // Create the database table for books
    global $wpdb;
    $table_name = $wpdb->prefix . 'books';
    $charset_collate = $wpdb->get_charset_collate();

    $sql = "CREATE TABLE $table_name (
        id mediumint(9) NOT NULL AUTO_INCREMENT,
        title varchar(255) NOT NULL,
        author varchar(255) NOT NULL,
        publication_date date NOT NULL,
        PRIMARY KEY (id)
    ) $charset_collate;";
```

```
require_once(ABSPATH . 'wp-admin/includes/upgrade.php');

dbDelta($sql);

}

// Deactivation hook

register_deactivation_hook(__FILE__, 'book_manager_deactivate');

function book_manager_deactivate() {

// Remove the database table for books

global $wpdb;

$table_name = $wpdb->prefix . 'books';

$wpdb->query("DROP TABLE IF EXISTS $table_name");

}

// Admin menu hook

add_action('admin_menu', 'book_manager_menu');

function book_manager_menu() {

add_menu_page(
'Book Manager',
'Book Manager',
'manage_options',
'book-manager',
```

```
'book_manager_list_books',
'dashicons-book-alt',
30
);
}

function book_manager_list_books() {
    // Display the list of books
    global $wpdb;
    $table_name = $wpdb->prefix . 'books';
    $books = $wpdb->get_results("SELECT * FROM $table_name");

    echo '<div class="wrap">';
    echo '<h1>Book Manager</h1>';
    echo '<a href="?page=book-manager&action=add" class="page-title-action">Add New Book</a>';
    echo '<table class="wp-list-table widefat fixed striped">';
    echo '<thead><tr><th>Title</th><th>Author</th><th>Publication Date</th><th>Actions</th></tr></thead>';
    echo '<tbody>';
    foreach ($books as $book) {
        echo '<tr>';
        echo '<td>' . $book->title . '</td>';
        echo '<td>' . $book->author . '</td>';
        echo '<td>' . $book->publication_date . '</td>';
        echo '<td>';
        echo '<a href="?page=book-manager&action=edit&id=' . $book->id . '">Edit</a> | ';
    }
}
```

```
echo '<a href="?page=book-manager&action=delete&id=' . $book->id . '">Delete</a>';

echo '</td>';

echo '</tr>';

}

echo '</tbody>';

echo '</table>';

echo '</div>';

}
```

```
// Process actions

add_action('admin_init', 'book_manager_process_actions');

function book_manager_process_actions() {

if (isset($_GET['page']) && $_GET['page'] === 'book-manager') {

if (isset($_GET['action'])) {

$action = $_GET['action'];

switch ($action) {

case 'add':

book_manager_add_book();

break;

case 'edit':

if (isset($_GET['id'])) {

book_manager_edit_book($_GET['id']);

}


```

```
        break;

    case 'delete':

        if (isset($_GET['id'])) {

            book_manager_delete_book($_GET['id']);

        }

        break;

    }

}

}
```

```
function book_manager_add_book() {  
    // Add a new book  
  
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
  
        global $wpdb;  
  
        $table_name = $wpdb->prefix . 'books';  
  
        $title = sanitize_text_field($_POST['title']);  
        $author = sanitize_text_field($_POST['author']);  
        $publication_date = sanitize_text_field($_POST['publication_date']);  
  
        $wpdb->insert(  
            $table_name,  
            array(  
                'title' => $title,
```

```
'author' => $author,  
'publication_date' => $publication_date,  
)  
);  
  
  
wp_redirect(admin_url('admin.php?page=book-manager'));  
exit;  
}  
else {  
// Display add book form  
  
echo '<div class="wrap">';  
  
echo '<h1>Add New Book</h1>';  
  
echo '<form method="post" action="">';  
  
echo '<label>Title:</label>';  
  
echo '<input type="text" name="title" required><br>';  
  
echo '<label>Author:</label>';  
  
echo '<input type="text" name="author" required><br>';  
  
echo '<label>Publication Date:</label>';  
  
echo '<input type="date" name="publication_date" required><br>';  
  
echo '<input type="submit" class="button button-primary" value="Add Book">';  
  
echo '</form>';  
  
echo '</div>';  
}  
}  
  
  
function book_manager_edit_book($id) {
```

```
// Edit an existing book

if ($_SERVER['REQUEST_METHOD'] === 'POST') {

    global $wpdb;

    $table_name = $wpdb->prefix . 'books';

    $title = sanitize_text_field($_POST['title']);

    $author = sanitize_text_field($_POST['author']);

    $publication_date = sanitize_text_field($_POST['publication_date']);

    $wpdb->update(
        $table_name,
        array(
            'title' => $title,
            'author' => $author,
            'publication_date' => $publication_date,
        ),
        array('id' => $id)
    );

    wp_redirect(admin_url('admin.php?page=book-manager'));
    exit;
} else {

    // Display edit book form

    global $wpdb;

    $table_name = $wpdb->prefix . 'books';
```

```
$book = $wpdb->get_row("SELECT * FROM $table_name WHERE id = $id");

if ($book) {

    echo '<div class="wrap">';

    echo '<h1>Edit Book</h1>';

    echo '<form method="post" action="">';

    echo '<label>Title:</label>';

    echo '<input type="text" name="title" value="" . $book->title . "" required><br>';

    echo '<label>Author:</label>';

    echo '<input type="text" name="author" value="" . $book->author . "" required><br>';

    echo '<label>Publication Date:</label>';

    echo '<input type="date" name="publication_date" value="" . $book->publication_date . "" required><br>';

    echo '<input type="submit" class="button button-primary" value="Update Book">';

    echo '</form>';

    echo '</div>';

} else {

    wp_redirect(admin_url('admin.php?page=book-manager'));

    exit;

}

}

}

function book_manager_delete_book($id) {

// Delete a book

global $wpdb;
```

```
$table_name = $wpdb->prefix . 'books';

$wpdb->delete($table_name, array('id' => $id));

wp_redirect(admin_url('admin.php?page=book-manager'));

exit;

}

// Enqueue styles and scripts

add_action('admin_enqueue_scripts', 'book_manager_enqueue_scripts');

function book_manager_enqueue_scripts($hook) {

if ($hook === 'toplevel_page_book-manager') {

wp_enqueue_style('book-manager-styles', plugins_url('css/styles.css', __FILE__));

}

}
```

Plugin Name: CRUD Book Plugin

```
<?php

/*
Plugin Name: CRUD Book Plugin
Description: Simple CRUD operations for managing books.
Version: 1.0
Author: Your Name
*/
// Create custom table on plugin activation
function crud_book_plugin_activation() {
    global $wpdb;
    $table_name = $wpdb->prefix . 'books';
    $charset_collate = $wpdb->get_charset_collate();
    $sql = "CREATE TABLE $table_name (
        id INT NOT NULL AUTO_INCREMENT,
        title VARCHAR(255) NOT NULL,
        author VARCHAR(255) NOT NULL,
        PRIMARY KEY (id)
    ) $charset_collate;";
    require_once(ABSPATH . 'wp-admin/includes/upgrade.php');
    dbDelta( $sql );
}
register_activation_hook( __FILE__, 'crud_book_plugin_activation' );
```

```
// Insert book data into the custom table

function crud_book_plugin_insert_book($title, $author) {

    global $wpdb;

    $table_name = $wpdb->prefix . 'books';

    $wpdb->insert(
        $table_name,
        array(
            'title' => $title,
            'author' => $author,
        )
    );
}
```

```
// Update book data in the custom table

function crud_book_plugin_update_book($id, $title, $author) {

    global $wpdb;

    $table_name = $wpdb->prefix . 'books';

    $wpdb->update(
        $table_name,
        array(
            'title' => $title,
            'author' => $author,
        ),
        array('id' => $id)
    );
}
```

```
}

// Delete book data from the custom table

function crud_book_plugin_delete_book($id) {

    global $wpdb;

    $table_name = $wpdb->prefix . 'books';

    $wpdb->delete(
        $table_name,
        array('id' => $id)
    );
}
```

```
// Retrieve all books from the custom table

function crud_book_plugin_get_books() {

    global $wpdb;

    $table_name = $wpdb->prefix . 'books';

    return $wpdb->get_results("SELECT * FROM $table_name", ARRAY_A);
}
```

Step 4: Now, let's create a simple front-end form to interact with our plugin and manage books. Add the following shortcode to the 'crud_book_plugin.php' file:

```
// Shortcode to display the book management form

function crud_book_plugin_shortcode() {

    ob_start();

    ?>

    <div class="crud-book-form">
```

```

<h3>Add a New Book</h3>

<form method="post">

    <input type="text" name="title" placeholder="Title" required>
    <input type="text" name="author" placeholder="Author" required>
    <button type="submit">Add Book</button>
</form>

<h3>Existing Books</h3>

<ul>
<?php
$books = crud_book_plugin_get_books();

foreach ($books as $book) {
    echo '<li>' . $book['title'] . ' - ' . $book['author'] . ' [<a href="?action=edit&id=' . $book['id'] . '">Edit</a> | <a href="?action=delete&id=' . $book['id'] . '">Delete</a>]</li>';
}
?>
</ul>
</div>

<?php
return ob_get_clean();
}

add_shortcode('crud_book_plugin', 'crud_book_plugin_shortcode');

// Handle form submissions

function crud_book_plugin_handle_form() {
if (isset($_POST['title']) && isset($_POST['author'])) {

```

```
$title = sanitize_text_field($_POST['title']);

$author = sanitize_text_field($_POST['author']);

crud_book_plugin_insert_book($title, $author);

}

}

add_action('init', 'crud_book_plugin_handle_form');

// Handle delete and edit actions

function crud_book_plugin_handle_actions() {

if (isset($_GET['action']) && isset($_GET['id'])) {

$id = intval($_GET['id']);

if ($_GET['action'] === 'delete') {

crud_book_plugin_delete_book($id);

} elseif ($_GET['action'] === 'edit') {

// For editing, you can implement a separate form and handle the update operation.

}

}

}

add_action('init', 'crud_book_plugin_handle_actions');
```

Plugin Name: CRUD Operations

```
<?php
/*
Plugin Name: CRUD Operations
Plugin URI: https://www.davidangulo.xyz/portfolio/
Description: A simple plugin that allows you to perform Create (INSERT), Read (SELECT), Update and Delete operations.
Version: 1.0.0
Author: David Angulo
Author URI: https://www.davidangulo.xyz/
License: GPL2
*/
register_activation_hook(__FILE__, 'crudOperationsTable');

function crudOperationsTable() {
    global $wpdb;
    $charset_collate = $wpdb->get_charset_collate();
    $table_name = $wpdb->prefix . 'userstable';
    $sql = "CREATE TABLE `{$table_name}` (
        `user_id` int(11) NOT NULL AUTO_INCREMENT,
        `name` varchar(220) DEFAULT NULL,
        `email` varchar(220) DEFAULT NULL,
        PRIMARY KEY(`user_id`)
    ) ENGINE=MyISAM DEFAULT CHARSET=latin1;
    ";

    if ($wpdb->get_var("SHOW TABLES LIKE '{$table_name}'") != $table_name) {
        require_once(ABSPATH . 'wp-admin/includes/upgrade.php');
        dbDelta($sql);
    }
}

add_action('admin_menu', 'addAdminPageContent');
function addAdminPageContent() {
    add_menu_page('CRUD', 'CRUD', 'manage_options' ,__FILE__, 'crudAdminPage',
    'dashicons-wordpress');
}

function crudAdminPage() {
    global $wpdb;
    $table_name = $wpdb->prefix . 'userstable';
```

```

if (isset($_POST['newsubmit'])) {
    $name = $_POST['newname'];
    $email = $_POST['newemail'];
    $wpdb->query("INSERT INTO $table_name(name,email) VALUES('$name','$email')");
    echo "<script>location.replace('admin.php?page=crud.php');</script>";
}

if (isset($_POST['uptsubmit'])) {
    $id = $_POST['uptid'];
    $name = $_POST['uptname'];
    $email = $_POST['uptemail'];
    $wpdb->query("UPDATE $table_name SET name='$name',email='$email' WHERE user_id='$id'");
    echo "<script>location.replace('admin.php?page=crud.php');</script>";
}

if (isset($_GET['del'])) {
    $del_id = $_GET['del'];
    $wpdb->query("DELETE FROM $table_name WHERE user_id='$del_id'");
    echo "<script>location.replace('admin.php?page=crud.php');</script>";
}

?>
<div class="wrap">
    <h2>CRUD Operations</h2>
    <table class="wp-list-table widefat striped">
        <thead>
            <tr>
                <th width="25%">User ID</th>
                <th width="25%">Name</th>
                <th width="25%">Email Address</th>
                <th width="25%">Actions</th>
            </tr>
        </thead>
        <tbody>
            <form action="" method="post">
                <tr>
                    <td><input type="text" value="AUTO_GENERATED" disabled></td>
                    <td><input type="text" id="newname" name="newname"></td>
                    <td><input type="text" id="newemail" name="newemail"></td>
                    <td><button id="newsubmit" name="newsubmit" type="submit">INSERT</button></td>
                </tr>
            </form>
        </tbody>
    </table>
</div>

```

```

<?php

$result = $wpdb->get_results("SELECT * FROM $table_name");
foreach ($result as $print) {
    echo "
        <tr>
            <td width='25%'>$print->user_id</td>
            <td width='25%'>$print->name</td>
            <td width='25%'>$print->email</td>
            <td width='25%'><a href='admin.php?page=crud.php&upt=$print-
$user_id'><button type='button'>UPDATE</button></a> <a
href='admin.php?page=crud.php&del=$print->user_id'><button
type='button'>DELETE</button></a></td>
        </tr>
    ";
}
?>
</tbody>
</table>
<br>
<br>

<?php
if (isset($_GET['upt'])) {
    $upt_id = $_GET['upt'];
    $result = $wpdb->get_results("SELECT * FROM $table_name WHERE
user_id='$upt_id'");
    foreach($result as $print) {
        $name = $print->name;
        $email = $print->email;
    }
    echo "
        <table class='wp-list-table widefat striped'>
            <thead>
                <tr>
                    <th width='25%'>User ID</th>
                    <th width='25%'>Name</th>
                    <th width='25%'>Email Address</th>
                    <th width='25%'>Actions</th>
                </tr>
            </thead>
            <tbody>
                <form action=' ' method='post'>
                    <tr>

```

```
        <td width='25%'>$print->user_id <input type='hidden' id='uptid'
name='uptid' value='$print->user_id'></td>
        <td width='25%'><input type='text' id='uptname' name='uptname'
value='$print->name'></td>
        <td width='25%'><input type='text' id='uptemail' name='uptemail'
value='$print->email'></td>
        <td width='25%'><button id='uptsubmit' name='uptsubmit'
type='submit'>UPDATE</button> <a href='admin.php?page=crud.php'><button
type='button'>CANCEL</button></a></td>
    </tr>
</form>
</tbody>
</table>";
}
?></div>
<?php}
```

Plugin Name: Custom EMS

```
<?php
/*
 * Plugin Name: Custom EMS
 * Description: My plugin to explain the crud functionality.
 * Version: 1.0
 * Author: Diwakar Academy
 * Plugin URI: https://diwakaracademy.com/how-to-create-crud-operations-plugin-in-wordpress/
 * Author URI: https://diwakaracademy.com/
 */

register_activation_hook(__FILE__, 'table_creator');
function table_creator()
{
    global $wpdb;
    $charset_collate = $wpdb->get_charset_collate();
    $table_name = $wpdb->prefix . 'ems';
    $sql = "DROP TABLE IF EXISTS $table_name;
            CREATE TABLE $table_name(
                id mediumint(11) NOT NULL AUTO_INCREMENT,
                emp_id varchar(50) NOT NULL,
                emp_name varchar (250) NOT NULL,
                emp_email varchar (250) NOT NULL,
                emp_dept varchar (250) NOT NULL,
                PRIMARY KEY id(id)
            )$charset_collate;";
    require_once(ABSPATH . 'wp-admin/includes/upgrade.php');
    dbDelta($sql);
}

add_action('admin_menu', 'da_display_esm_menu');
function da_display_esm_menu()
{
    add_menu_page('EMS', 'EMS', 'manage_options', 'emp-list',
    'da_ems_list_callback');
    add_submenu_page('emp-list', 'Employee List', 'Employee List',
    'manage_options', 'emp-list', 'da_ems_list_callback');
    add_submenu_page('emp-list', 'Add Employee', 'Add Employee',
    'manage_options', 'add-emp', 'da_ems_add_callback');
    add_submenu_page(null, 'Update Employee', 'Update Employee',
    'manage_options', 'update-emp', 'da_emp_update_call');
}
```

```

        add_submenu_page(null, 'Delete Employee', 'Delete Employee',
'manage_options', 'delete-emp', 'da_emp_delete_call');
        add_submenu_page('emp-list', 'Employee List Shortcode', 'Employee List
_shortcode', 'edit_others_posts', 'emp-shotcode', 'da_emp_shortcode_call');

}

function da_emps_add_callback()
{
    global $wpdb;
    $table_name = $wpdb->prefix . 'ems';
    $msg = '';
    if (isset($_REQUEST['submit'])) {

        $wpdb->insert("$table_name", [
            'emp_id' => $_REQUEST['emp_id'],
            'emp_name' => $_REQUEST['emp_name'],
            'emp_email' => $_REQUEST['emp_email'],
            'emp_dept' => $_REQUEST['emp_dept']
        ]);

        if ($wpdb->insert_id > 0) {
            $msg = "Saved Successfully";
        } else {
            $msg = "Failed to save data";
        }
    }
?>

<h4 id="msg"><?php echo $msg; ?></h4>

<form method="post">
<p><label>EMP ID</label> <input type="text" name="emp_id" placeholder="Enter
ID" required> </p>
<p><label>Name</label> <input type="text" name="emp_name"
placeholder="Enter Name" required> </p>
<p><label>Email</label><input type="email" name="emp_email"
placeholder="Enter Email" required> </p>
<p>
    <label>Department</label> <input type="text" name="emp_dept"
placeholder="Enter Department" required> </p>
<p><button type="submit" name="submit">Submit</button></p>
</form>
<?php }

```

```

function da_emp_shortcode_call()
{ ?>
    <p> <label>Shortcode</label> <input type="text" value="[employee_list]"></p>
<?php }

// [employee_list]
add_shortcode('employee_list', 'da_emps_list_callback');
function da_emps_list_callback()
{
    global $wpdb;
    $table_name = $wpdb->prefix . 'ems';
    $employee_list = $wpdb->get_results($wpdb->prepare("select * FROM
$table_name", ""), ARRAY_A);
    if (count($employee_list) > 0): ?>
        <div style="margin-top: 40px;">
            <table border="1" cellpadding="10">
                <tr>
                    <th>S.No.</th>
                    <th>EMP ID</th>
                    <th>Name</th>
                    <th>Email</th>
                    <th>Department</th>
                    <?php if (is_admin()): ?>
                        <th>Action</th>
                    <?php endif; ?>
                </tr>
                <?php $i = 1;
                foreach ($employee_list as $index => $employee): ?>
                    <tr>
                        <td><?php echo $i++; ?></td>
                        <td><?php echo $employee['emp_id']; ?></td>
                        <td><?php echo $employee['emp_name']; ?></td>
                        <td><?php echo $employee['emp_email']; ?></td>
                        <td><?php echo $employee['emp_dept']; ?></td>
                        <?php if (is_admin()): ?>
                            <td>
                                <a href="admin.php?page=update-emp&id=<?php echo
$employee['id']; ?>">Edit</a>
                                <a href="admin.php?page=delete-emp&id=<?php echo
$employee['id']; ?>">Delete</a>
                            </td>
                        <?php endif; ?>
                    </tr>
    
```

```

        <?php endforeach; ?>
    </table>

</div>
<?php else:echo "<h2>Employee Record Not Found</h2>";endif;
}

function da_emp_update_call()
{
    global $wpdb;
    $table_name = $wpdb->prefix . 'ems';
    $msg = '';
    $id = isset($_REQUEST['id']) ? intval($_REQUEST['id']) : "";
    if (isset($_REQUEST['update'])) {
        if (!empty($id)) {
            $wpdb->update("$table_name", ["emp_id" => $_REQUEST['emp_id'],
'emp_name' => $_REQUEST['emp_name'], 'emp_email' => $_REQUEST['emp_email'],
'emp_dept' => $_REQUEST['emp_dept']], ["id" => $id]);
            $msg = 'Data updated';
        }
    }
    $employee_details = $wpdb->get_row($wpdb->prepare("SELECT * FROM $table_name
where id = %d", $id), ARRAY_A); ?>
<h4><?php echo $msg; ?></h4>
<form method="post">
    <p>
        <label>EMP ID</label>
        <input type="text" name="emp_id" placeholder="Enter ID" value="<?php
echo $employee_details['emp_id']; ?>" required>
    </p>

    <p>
        <label>Name</label>
        <input type="text" name="emp_name" placeholder="Enter Name"
value="<?php echo $employee_details['emp_name']; ?>" required>
    </p>
    <p>
        <label>Email</label>
        <input type="email" name="emp_email" placeholder="Enter Email"
value="<?php echo $employee_details['emp_email']; ?>" required>
    </p>

```

```

<p>
    <label>Department</label>
    <input type="text" name="emp_dept" placeholder="Enter Department"
           value="php echo $employee_details['emp_dept']; ?" required>
</p>
<p>
    <button type="submit" name="update">Update</button>
</p>
</form>
<?php }>

function da_emp_delete_call()
{
    global $wpdb;
    $table_name = $wpdb->prefix . 'ems';
    $id = isset($_REQUEST['id']) ? intval($_REQUEST['id']) : "";
    if (isset($_REQUEST['delete'])) {
        if ($_REQUEST['conf'] == 'yes') {
            $row_exits = $wpdb->get_row($wpdb->prepare("SELECT * FROM $table_name
WHERE id = %d", $id), ARRAY_A);
            if (count($row_exits) > 0) {
                $wpdb->delete("$table_name", array('id' => $id));
            }
        } ?>
        <script>
            location.href = "<?php echo site_url(); ?>/wp-
admin/admin.php?page=emp-list";
        </script>
    } ?>
    <?php } ?>
    <form method="post">
        <p>
            <label>Are you sure want delete?</label><br>
            <input type="radio" name="conf" value="yes">Yes
            <input type="radio" name="conf" value="no" checked>No
        </p>
        <p>
            <button type="submit" name="delete">Delete</button>
            <input type="hidden" name="id" value="php echo $id; ?&gt;"&gt;
        &lt;/p&gt;
    &lt;/form&gt;
&lt;?php }&gt;
</pre

```

Plugin Name: Custom EMS

```
<?php
/*
 * Plugin Name: Custom EMS
 * Description: My plugin to explain the frontend crud functionality.
 * Version: 1.0
 * Author: Diwakar Academy
 * Plugin URI: hhttps://diwakaracademy.com/how-to-create-crud-operations-from-front-end-in-wordpress/
 * Author URI: https://diwakaracademy.com/
 */

register_activation_hook(__FILE__, 'table_creator');
function table_creator()
{
    global $wpdb;
    $charset_collate = $wpdb->get_charset_collate();
    $table_name = $wpdb->prefix . 'ems';
    $sql = "DROP TABLE IF EXISTS $table_name;
            CREATE TABLE $table_name(
                id mediumint(11) NOT NULL AUTO_INCREMENT,
                emp_id varchar(50) NOT NULL,
                emp_name varchar (250) NOT NULL,
                emp_email varchar (250) NOT NULL,
                emp_dept varchar (250) NOT NULL,
                PRIMARY KEY id(id)
            )$charset_collate;";
    require_once(ABSPATH . 'wp-admin/includes/upgrade.php');
    dbDelta($sql);
}

//[frontend_crud]
add_shortcode('frontend_crud','da_ems_frontend_crud_callback');
function da_ems_frontend_crud_callback(){
    global $wpdb;
    $table_name = $wpdb->prefix.'ems';
    $msg = '';
    if(@$_REQUEST['action'] == 'submit'){

        $wpdb->insert("$table_name", [
            'emp_id' => sanitize_text_field($_REQUEST['emp_id']),
            'emp_name'=> sanitize_text_field($_REQUEST['emp_name']),
            'emp_email'=> sanitize_email($_REQUEST['emp_email']),
            'emp_dept'=> sanitize_text_field($_REQUEST['emp_dept']),
        ]);
    }
}
```

```

    ]);

    if($wpdb->insert_id > 0){
        $msg = "Saved Successfully";
    }else{
        $msg = "Failed to save data";
    }
}

if(@$_REQUEST['action'] == 'update-emp' && @$_REQUEST['id']){
    $id = @$_REQUEST['id'];

    if(@$_REQUEST['emp_id'] && @$_REQUEST['emp_name'] &&
    @$_REQUEST['emp_email'] && @$_REQUEST['emp_dept']){
        $update = $wpdb->update("$table_name", [
            'emp_id' => sanitize_text_field($_REQUEST['emp_id']),
            'emp_name' => sanitize_text_field($_REQUEST['emp_name']),
            'emp_email' => sanitize_email($_REQUEST['emp_email']),
            'emp_dept' => sanitize_text_field($_REQUEST['emp_dept']),
            ['id' => $id]);
    }

    if($update){
        $msg = "Data Updated <a
href='".get_page_link(get_the_ID())."'>Add Employee</a>";
    }
}

$employee = $wpdb->get_row($wpdb->prepare("select * from $table_name
where id = %d", $id), ARRAY_A);

$emp_id = $employee['emp_id'];
$emp_name = $employee['emp_name'];
$emp_email = $employee['emp_email'];
$emp_dept = $employee['emp_dept'];
}

if(@$_REQUEST['action'] == 'delete-emp' && @$_REQUEST['id']){
    $id = @$_REQUEST['id'];

    if($id){
        $row_exists = $wpdb->get_row($wpdb->prepare("select * from $table_name
where id = %d", $id), ARRAY_A);
        if(count($row_exists) > 0){
            $wpdb->delete("$table_name", array('id'=>$id));
        }
    }
}

```

```

        }
    ?>
        <script>
            location.href=<?php echo get_the_permalink(); ?>;
        </script>
    <?php
}
?>

<div class="form_container">
<h4><?php echo @$msg; ?></h4>
<form method="post">
    <p>
        <label>EMP ID</label>
        <input type="text" name="emp_id" value=<?php echo @$emp_id; ?>" placeholder="Enter ID" required>
    </p>
    <p>
        <label>Name</label>
        <input type="text" name="emp_name" value=<?php echo @$emp_name; ?>" placeholder="Enter Name" required>
    </p>
    <p>
        <label>Email</label>
        <input type="email" name="emp_email" value=<?php echo @$emp_email; ?>" placeholder="Enter Email" required>
    </p>
    <p>
        <label>Department</label>
        <input type="text" name="emp_dept" value=<?php echo @$emp_dept; ?>" placeholder="Enter Department" required>
    </p>

    <p>
        <button type="submit" name="action" value=<?php echo (@$_REQUEST['action'] == 'update-emp')?'update-emp':'submit'; ?>><?php echo (@$_REQUEST['action'] == 'update-emp')?'Update':'Submit'; ?>></button>
    </p>
</form>

</div>
<?php

$employee_list = $wpdb->get_results("SELECT * FROM $table_name", ARRAY_A);
$i = 1;

```

```

if($employee_list > 0 ){

    <div style="margin-top: 40px">
        <table border="1" cellpadding="10" style="font-size:14px;">
            <tr>
                <th>S. No.</th>
                <th>EMP ID</th>
                <th>Name</th>
                <th>Email</th>
                <th>Dept.</th>
                <th>Action</th>
            </tr>
            <?php foreach ($employee_list as $index => $employee):>

                ?>
                <tr>
                    <td><?php echo $i++; ?></td>
                    <td><?php echo $employee['emp_id']; ?></td>
                    <td><?php echo $employee['emp_name']; ?></td>
                    <td><?php echo $employee['emp_email']; ?></td>
                    <td><?php echo $employee['emp_dept']; ?></td>
                    <td>
                        <a href="?action=update-emp&id=<?php echo $employee['id']; ?>">Update</a>
                        <a href="?action=delete-emp&id=<?php echo $employee['id']; ?>" onclick="return confirm('Are you sure to remove this record?')">Delete</a>
                    </td>
                </tr>
            <?php endforeach; ?>
        </table>
    </div>
    <?php }>
}

```

CRUD for Wordpress theme

1.In your WordPress theme directory, create a new folder for your CRUD application. Let's call it "book-manager".

2.Create the main file, book-manager.php, inside the "book-manager" folder, and add the following code:

```
<?php
// Template Name: Book Manager

get_header();

// Check if the user is logged in and has the necessary capability
if (is_user_logged_in() && current_user_can('manage_options')) {
    // Handle form submissions
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        // Process CRUD operations
        if (isset($_POST['action'])) {
            $action = $_POST['action'];

            switch ($action) {
                case 'add':
                    // Handle the add operation
                    book_manager_add_book();
                    break;
                case 'edit':
                    // Handle the edit operation
                    if (isset($_POST['book_id'])) {
                        $book_id = intval($_POST['book_id']);
                        book_manager_edit_book($book_id);
                    }
                    break;
                case 'delete':
                    // Handle the delete operation
                    if (isset($_POST['book_id'])) {
                        $book_id = intval($_POST['book_id']);
                        book_manager_delete_book($book_id);
                    }
                    break;
            }
        }
    }
}

// Display the book manager form
```

```
book_manager_display_form();
} else {
    echo '<p>You do not have sufficient permissions to access this page.</p>';
}

get_footer();

// CRUD operations

function book_manager_add_book() {
    // Perform the add operation
    // Retrieve and sanitize form data
    $title = sanitize_text_field($_POST['title']);
    $author = sanitize_text_field($_POST['author']);
    $publication_date = sanitize_text_field($_POST['publication_date']);

    // Add your code to handle the add operation
    // Example:
    // $new_book_id = wp_insert_post(array(
    //     'post_title' => $title,
    //     'post_content' => $author,
    //     'post_date' => $publication_date,
    //     'post_type' => 'book',
    // ));

    // Redirect to the book manager page after the operation
    wp_redirect(get_permalink());
    exit;
}

function book_manager_edit_book($book_id) {
    // Perform the edit operation
    // Retrieve and sanitize form data
    $title = sanitize_text_field($_POST['title']);
    $author = sanitize_text_field($_POST['author']);
    $publication_date = sanitize_text_field($_POST['publication_date']);

    // Add your code to handle the edit operation
    // Example:
    // $updated = wp_update_post(array(
    //     'ID' => $book_id,
    //     'post_title' => $title,
    //     'post_content' => $author,
    //     'post_date' => $publication_date,
    // ));
}
```

```
// Redirect to the book manager page after the operation
wp_redirect(get_permalink());
exit;
}

function book_manager_delete_book($book_id) {
    // Perform the delete operation
    // Add your code to handle the delete operation
    // Example:
    // $deleted = wp_delete_post($book_id, true);

    // Redirect to the book manager page after the operation
    wp_redirect(get_permalink());
    exit;
}

// Display form

function book_manager_display_form() {
    // Display the book manager form
    ?>
    <div class="book-manager-form">
        <h2>Book Manager</h2>
        <form method="post" action="">
            <input type="hidden" name="action" value="add">
            <label for="title">Title:</label>
            <input type="text" name="title" required>
            <label for="author">Author:</label>
            <input type="text" name="author" required>
            <label for="publication_date">Publication Date:</label>
            <input type="date" name="publication_date" required>
            <input type="submit" value="Add Book">
        </form>
        <?php
        // Retrieve books from the database and display them in a table
        $books = get_posts(array(
            'post_type' => 'book',
            'posts_per_page' => -1,
        ));

        if ($books) {
            echo '<table>';
            echo '<tr><th>Title</th><th>Author</th><th>Publication
Date</th><th>Actions</th></tr>';
            foreach ($books as $book) {
                echo '<tr>';
                echo '<td>' . $book['title'] . '</td>';
                echo '<td>' . $book['author'] . '</td>';
                echo '<td>' . $book['date'] . '</td>';
                echo '<td>';
```

```

foreach ($books as $book) {
    $book_id = $book->ID;
    $title = $book->post_title;
    $author = $book->post_content;
    $publication_date = $book->post_date;

    echo '<tr>';
    echo '<td>' . $title . '</td>';
    echo '<td>' . $author . '</td>';
    echo '<td>' . $publication_date . '</td>';
    echo '<td>';
    echo '<form method="post" action="">';
    echo '<input type="hidden" name="action" value="edit">';
    echo '<input type="hidden" name="book_id" value="' . $book_id . '"';
    echo '<">';
    echo '<input type="submit" value="Edit">';
    echo '</form>';
    echo '<form method="post" action="">';
    echo '<input type="hidden" name="action" value="delete">';
    echo '<input type="hidden" name="book_id" value="' . $book_id . '"';
    echo '<">';
    echo '<input type="submit" value="Delete">';
    echo '</form>';
    echo '</td>';
    echo '</tr>';
}

echo '</table>';
} else {
    echo '<p>No books found.</p>';
}
?>
</div>
<?php
}

```

Create a [new page template file](#), `template-book-manager.php`, in your theme directory, and add the following code:

```

<?php
/*
 * Template Name: Book Manager
 */

```

```
get_header();

if (have_posts()) {
    while (have_posts()) {
        the_post();
        the_content();
    }
}

get_footer();
```

Custom end point

References Video

<https://www.youtube.com/watch?v=ckpd90TyBwY&t=32s>

```
wp-content > plugins > wp-learn-form-submissions-api > wp-learn-form-submissions-api.php

1  <?php
2  /**
3   * Plugin Name: WP Learn Form Submissions API
4   * Version: 0.0.1
5   */
6
7  register_activation_hook( __FILE__, 'wp_learn_setup_table' );
8
9  function wp_learn_setup_table() {
10    global $wpdb;
11    $table_name = $wpdb->prefix . 'form_submissions';
12
13    $sql = "CREATE TABLE $table_name (
14      id mediumint(9) NOT NULL AUTO_INCREMENT,
15      name varchar (100) NOT NULL,
16      email varchar (100) NOT NULL,
17      PRIMARY KEY  (id)
18    )";
19
20    require_once(ABSPATH . 'wp-admin/includes/upgrade.php' );
21    dbDelta( $sql );  I
22  }

23
24  add_action( 'rest_api_init', 'wp_learn_register_routes' );
25  function wp_learn_register_routes() {
26    register_rest_route(
27      'wp-learn-form-submissions-api/v1',
28      '/form-submissions',
29      array(
30        'methods' => 'GET',
31        'callback' => 'wp_learn_rest_get_form_submissions',
32        'permission_callback' => '__return_true'
33      )
34    );
35  }
36
```

```
38     register_rest_route(
39         'wp-learn-form-submissions-api/v1',
40         '/form-submission/(?P<id>[\d]+)',
41         array(
42             'methods'          => 'GET',
43             'callback'         => 'wp_learn_rest_get_form_submission',
44             'permission_callback' => '__return_true',
45         )
46     );
47
48     register_rest_route(
49         'wp-learn-form-submissions-api/v1',
50         '/form-submissions',
51         array(
52             'methods'          => 'POST',
53             'callback'         => 'wp_learn_rest_create_form_submissions',
54             'permission_callback' => '__return_true',
55         )
56     );
57
58
59
60 }
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
```

```
59
60     function wp_learn_rest_get_form_submissions() {
61         global $wpdb;
62         $table_name = $wpdb->prefix . 'form_submissions';
63
64         $results = $wpdb->get_results( "SELECT * FROM $table_name" );
65         return $results;
66     }
67
68     function wp_learn_rest_get_form_submission( $request ) {
69         $id = $request['id'];
70         global $wpdb;
71         $table_name = $wpdb->prefix . 'form_submissions';
72
73         $results = $wpdb->get_results( "SELECT * FROM $table_name WHERE id = $id" );
74         return $results[0];
75     }
76 }
```

```
79
80
81 function wp_learn_rest_create_form_submissions( $request ) {
82     global $wpdb;
83     $table_name = $wpdb->prefix . 'form_submissions';
84
85     $rows = $wpdb->insert(
86         $table_name,
87         array(
88             'name'  => $request['name'],
89             'email' => $request['email'],
90         )
91     );
92     return 'Form submission added';
93 }
```

WordPress HTTP API

Official documentation

<https://developer.wordpress.org/plugins/http-api/>

GETting data from an API

[Top ↑](#)

[GitHub](#) provides an excellent API that does not require app registration for many public aspects, so to demonstrate some of these methods, examples will target the GitHub API.

GETting data is made incredibly simple in WordPress through the [wp_remote_get\(\)](#) function. This function takes the following two arguments:

1. \$url – Resource to retrieve data from. This must be in a standard HTTP format
2. \$args – OPTIONAL – You may pass an array of arguments in here to alter behavior and headers, such as cookies, follow redirects, etc.

The following defaults are assumed, though they can be changed via the \$args parameter:

- method – GET
- timeout – 5 – How long to wait before giving up
- redirection – 5 – How many times to follow redirects.
- httpversion – 1.0
- blocking – true – Should the rest of the page wait to finish loading until this operation is complete?
- headers – array()
- body – null
- cookies – array()

```
$response = wp_remote_get( 'https://api.github.com/users/blobaugh' );
```

Output:

```
Array(
  [headers] => Array(
    [server] => nginx
    [date] => Fri, 05 Oct 2012 04:43:50 GMT
    [content-type] => application/json; charset=utf-8
    [connection] => close
    [status] => 200 OK
    [vary] => Accept
    [x-ratelimit-remaining] => 4988
    [content-length] => 594
    [last-modified] => Fri, 05 Oct 2012 04:39:58 GMT
    [etag] => "5d5e6f7a09462d6a2b473fb616a26d2a"
    [x-github-media-type] => github.beta
    [cache-control] => public, s-maxage=60, max-age=60
    [x-content-type-options] => nosniff
    [x-ratelimit-limit] => 5000
  )
  [body] => {"type": "User", "login": "blobaugh", "gravatar_id": "f25f324a4
  [response] => Array(
    [preserved_text 5237511b45884ac6db1ff9d7e407f225 /] => 200
    [message] => OK
  )
  [cookies] => Array()
  [filename] =>
)
```

GET the body you always wanted

[Top ↑](#)

Just the body can be retrieved using `wp_remote_retrieve_body()`. This function takes just one parameter, the response from any of the other `wp_remote_X` functions where retrieve is not the next value.

[Copy](#)

```
$response = wp_remote_get( 'https://api.github.com/users/blobaugh' );
$body     = wp_remote_retrieve_body( $response );
```

Still using the GitHub resource from the previous example, `$body` will be

[Copy](#)

```
{"type": "User", "login": "blobaugh", "public_repos": 31, "gravatar_id": "f25f3}
```

GET the response code

[Top ↑](#)

You may want to check the response code to ensure your retrieval was successful. This can be done via the `wp_remote_retrieve_response_code()` function:

[Copy](#)

```
$response = wp_remote_get( 'https://api.github.com/users/blobaugh' );
$http_code = wp_remote_retrieve_response_code( $response );
```

If successful `$http_code` will contain 200.

GET a specific header

[Top ↑](#)

If your desire is to retrieve a specific header, say last-modified, you can do so with `wp_remote_retrieve_header()`. This function takes two parameters

1. `$response` – The response from the get call
2. `$header` – Name of the header to retrieve

To retrieve the last-modified header

[Copy](#)

```
$response      = wp_remote_get( 'https://api.github.com/users/blobaugh'
$last_modified = wp_remote_retrieve_header( $response, 'last-modified' )
```

GET using basic authentication

[Top ↑](#)

APIs that are secured more provide one or more of many different types of authentication. A common, though not highly secure, authentication method is HTTP Basic Authentication. It can be used in WordPress by passing 'Authorization' to the second parameter of the `wp_remote_get()` function, as well as the other HTTP method functions.

```
$args = array(
    'headers' => array(
        'Authorization' => 'Basic ' . base64_encode( YOUR_USERNAME . ':' .
YOUR_PASSWORD )
    )
);
wp_remote_get( $url, $args );
```

```
1 $response = wp_remote_post( $api_url, array(
2     'headers' => array(
3         'Content-Type' => 'application/json',
4         'Authorization' => 'Bearer ' . $api_key
5     ),
6     'body'      => json_encode( $body, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_SLASHES )
7 ) );
```

POSTing data to an API

[Top ↑](#)

The same helper methods ([wp_remote_retrieve_body\(\)](#), etc) are available for all of the HTTP method calls, and utilized in the same fashion.

POSTing data is done using the [wp_remote_post\(\)](#) function, and takes exactly the same parameters as [wp_remote_get\(\)](#). It should be noted here that you are required to pass in ALL of the elements in the array for the second parameter. The Codex provides the default acceptable values. You only need to care right now about the data you are sending so the other values will be defaulted.

Lets assume we are submitting a contact form with the following fields: name, email, subject, comment. To setup the body we do the following:

```
$body = array(  
    'name'      => 'Jane Smith',  
    'email'     => 'some@email.com',  
    'subject'   => 'Checkout this API stuff',  
    'comment'   => 'I just read a great tutorial. You gotta check it out!'  
)
```

[Copy](#)

Now we need to set up the rest of the values that will be passed to the second parameter of [wp_remote_post\(\)](#).

```
$args = array(
    'body'          => $body,
    'timeout'       => '5',
    'redirection'  => '5',
    'httpversion'   => '1.0',
    'blocking'      => true,
    'headers'       => array(),
    'cookies'       => array(),
);
```

Then of course to make the call

[Copy](#)

```
$response = wp_remote_post( 'http://your-contact-form.com', $args );
```

HEADing off bandwidth usage

[Top ↑](#)

It can be pretty important, and sometimes required by the API, to check a resource status using HEAD before retrieving it. On high traffic APIs, GET is often limited to a number of requests per minute or hour. There is no need to even attempt a GET request unless the HEAD request shows that the data on the API has been updated.

As mentioned previously, HEAD contains data on whether or not the data has been updated, if the data should be cached, when to expire the cached copy, and sometimes a rate limit on requests to the API.

Going back to the GitHub example, here are few headers to watch out for. Most of these headers are standard, but you should always check the API docs to ensure you understand which headers are named what, and their purpose.

- **x-ratelimit-limit** – Number of requests allowed in a time period
- **x-ratelimit-remaining** – Number of remaining available requests in time period
- **content-length** – How large the content is in bytes. Can be useful to warn the user if the content is fairly large
- **last-modified** – When the resource was last modified. Highly useful to caching tools
- **cache-control** – How should the client handle caching

The following will check the HEAD value of my GitHub user account:

[Copy](#)

```
$response = wp_remote_head( 'https://api.github.com/users/blobaugh' );
```

\$response should look similar to

[Copy](#)

[Collapse code](#)

```
Array(
    [headers] => Array(
        (
            [server] => nginx
            [date] => Fri, 05 Oct 2012 05:21:26 GMT
            [content-type] => application/json; charset=utf-8
            [connection] => close
            [status] => 200 OK
            [vary] => Accept
            [x-ratelimit-remaining] => 4982
            [content-length] => 594
            [last-modified] => Fri, 05 Oct 2012 04:39:58 GMT
            [etag] => "5d5e6f7a09462d6a2b473fb616a26d2a"
            [x-github-media-type] => github.beta
            [cache-control] => public, s-maxage=60, max-age=60
            [x-content-type-options] => nosniff
            [x-ratelimit-limit] => 5000
        )
        [body] =>
        [response] => Array(
            (
                [preserved_text 39a8515bd2dce2aa06ee8a2a6656b1de /] => 200
                [message] => OK
            )
        [cookies] => Array(
    )
)
```

Make any sort of request

[Top ↑](#)

If you need to make a request using an HTTP method that is not supported by any of the above functions do not panic. The great people developing WordPress already thought of that and lovingly provided `wp_remote_request()`. This function takes the same two parameters as `wp_remote_get()`, and allows you to specify the HTTP method as well. What data you need to pass along is up to your method.

To send a `DELETE` method example you may have something similar to the following:

```
$args      = array(
    'method' => 'DELETE',
);
$response = wp_remote_request( 'http://some-api.com/object/to/delete', $args );
```

Cache an object (Set a transient)

[Top ↑](#)

Caching an object is done with the `set_transient()` function. This function takes the following three parameters:

1. `$transient` – Name of the transient for future reference
2. `$value` – Value of the transient
3. `$expiration` – How many seconds from saving the transient until it expires

An example of caching the GitHub user information response from above for one hour would be

[Copy](#)

```
$response = wp_remote_get( 'https://api.github.com/users/blobaugh' );
set_transient( 'prefix_github_userinfo', $response, 60 * 60 );
```

› Get a cached object (Get a transient)

[Top ↑](#)

Getting a cached object is quite a bit more complex than setting a transient. You need to request the transient, but then you also need to check to see if that transient has expired and if so fetch updated data. Usually the `set_transient()` call is made inside of the `get_transient()` call. Here is an example of getting the transient data for the GitHub user profile:

[Copy](#)

```
$github_userinfo = get_transient( 'prefix_github_userinfo' );
if ( false === $github_userinfo ) {
    // Transient expired, refresh the data
    $response = wp_remote_get( 'https://api.github.com/users/blobaugh' );
    set_transient( 'prefix_github_userinfo', $response, HOUR_IN_SECONDS )
}
// Use $github_userinfo as you will
```

Delete a cached object (Delete a transient)

[Top ↑](#)

Deleting a cached object is the easiest of all the transient functions, simply pass it a parameter of the name of the transient and you are done.

To remove the Github user info:

[Copy](#)

```
delete_transient( 'blobaugh_github_userinfo' );
```

There are basically two parts to the equation: the **HTTP request** and the **HTTP response**, or transaction. Both the request and the response are very similar in structure, they both have four parts:

Headers

Headers contain various bits of information about the request or the response.

HTTP 1.1 defines 46 types of headers but only one is required (only for requests), the "Host" header. Take a look at the screenshot from my Chrome developer tools that shows some of the headers sent along with a request to the main Kinsta blog:

Request Headers

```
:host: kinsta.com
:method: GET
:path: /blog/
:scheme: https
:version: HTTP/1.1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
accept-encoding: gzip, deflate, sdch
accept-language: en-US,en;q=0.8,hu;q=0.6
cache-control: max-age=0
```

Body

The body usually contains data about the requested resource. If you send a GET request to the main Kinsta blog you should receive the HTML required to render the page (the resource) in the body content.

About Restful APIs

Restful APIs, or the REST methodology aims to provide a simple and standard way to interact with an application (here you can learn more about the [basics of the WordPress REST API](#)). It is often used in conjunction with HTTP to create a very understandable system of interactions. It is based on **paths** and **HTTP verbs**.

HTTP Verbs are the same as the method names we saw earlier, the most common ones are: GET, POST, PUT, DELETE. I think PUT is the only ambiguous one here, think of it as an update command. When using these verbs together with paths we can construct a meaningful system:

`GET /post/1/` would be used to retrieve the post with the ID of 1. `DELETE /post/1/` would be used to delete the same post. We could also use `PUT /post/1/` to update it, supplying relevant information in the request body and headers.

I'm sure you can see that just by appending an HTTP version to our codes above we actually have the initial line of an HTTP transaction, which is just one reason that this system is so powerful.

Using The WordPress HTTP API

Armed with all that knowledge we can easily take in how the WordPress HTTP API works. The four methods used to make requests and intercept the responses are:

- `wp_remote_get()`
- `wp_remote_post()`
- `wp_remote_head()`
- `wp_remote_request()`

The first two functions are self-explanatory, they use the GET and POST methods respectively in the request. The third function uses the HEAD method, something we haven't talked about yet. This method is used to retrieve only the headers of a response. This can save a **lot** of overhead if we just need some metadata about a resource. The final function is a generic one, you can specify which method you would like to use within the function's parameters.

There are five additional functions we can use to retrieve specific parts of the response. These are basically shotcuts to navigate the mass of data we receive:

- `wp_remote_retrieve_body()`
- `wp_remote_retrieve_header()`
- `wp_remote_retrieve_headers()`
- `wp_remote_retrieve_response_code()`
- `wp_remote_retrieve_response_message()`

Step 1: Encode Consumer Key And Secret

Once you create an application you should have a consumer key and secret at hand. To make things easier, let's create constants that hold this information for us.

```
define( 'TWITTER_CONSUMER_KEY', '12disnir382jeqwdasd23wdasi' );
define( 'TWITTER_CONSUMER_SECRET', '23wdajskduhtrq2c32cuq9r8uhuf' )
```

The three steps of creating an encoded version of these are laid out in the docs:

- URL encode the consumer key and the consumer secret
- Concatenate them with a colon
- Base64 encode the whole string

In PHP this will be pretty easy to do, here goes!

```
$key = urlencode( TWITTER_CONSUMER_KEY );
$secret = urlencode( TWITTER_CONSUMER_SECRET );
$concatenated = $key . ':' . $secret;
$encoded = base64_encode( $concatenated );
```

Step 2: Getting a Bearer Token

Instead of using your actual password, you send Twitter your encoded string (which uses your API credentials) and you receive a temporary pass which is valid for a set amount of time. To do this we'll be making an HTTP request, here's what Twitter has to say:

- The request must be a HTTP POST request.
- The request must include an Authorization header with the value of Basic .
- The request must include a Content-Type header with the value of application/x-www-form-urlencoded; charset=UTF-8.
- The body of the request must be grant_type=client_credentials.

Let's start with the basics. We know we need a POST request so we'll be using `wp_remote_post()`. The function takes two parameters; the first one is the URL, the second holds optional arguments. The URL will be

`https://api.twitter.com/oauth2/token`, we'll use the second parameter to deal with all the other requirements.

```
$args = array(
    'headers' => array(
        'Authorization' => 'Basic ' . $encoded,
        'Content-Type' => 'application/x-www-form-urlencoded; charset=UTF-8'
    ),
    'body' => 'grant_type=client_credentials'
);
$response = wp_remote_post( 'https://api.twitter.com/oauth2/token', $args );
```

```
$body = wp_remote_retrieve_body( $response );
$body = json_decode( $body, true );
$access_token = $body['access_token'];
```

Step 3: Use The Bearer Token

The last step is simply to use this bearer token in all other API calls. We need to add it as an "Authorization" header with the value: `Bearer [bearer_token]`. Let's do a simple API call that will retrieve a user's latest tweets using the [user_timeline](#) path.

```
$url =
'https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=danielpatrick&count=3';
$args = array(
    'headers' => array(
        'Authorization' => 'Bearer ' . $access_token,
    ),
);
$response = wp_remote_get( $url, $args );
$tweets = json_decode( wp_remote_retrieve_body($response), true )
```

Fetching data from an API in a WordPress website can be accomplished using various methods. One of the common ways is by using the built-in WordPress HTTP API, or by utilizing plugins or custom code. Here's a general idea of how you might go about it:

Using WordPress HTTP API:

WordPress has its own HTTP API that allows you to make requests to external APIs. You can use this API to fetch data from a remote server. Here's an example of how you might do it:

```
php
Copy code

function fetch_data_from_api() {
    $api_url = 'https://api.example.com/data'; // Replace this with your API
    $response = wp_remote_get($api_url);

    if (is_array($response)) {
        $body = $response['body']; // Contains the API response as a string
        // Process $body as needed
    }
}
```

for adding data to an API server in WordPress using Basic Authentication or JWT token, you'll need to include the authentication details in your HTTP request headers. Here's an example using Basic Authentication and JWT token with the WordPress HTTP API:

```
function add_data_with_basic_auth_to_api() {
    $api_url = 'https://api.example.com/add'; // Replace this with your API
    $data = array(
        'key1' => 'value1',
        'key2' => 'value2'
        // Add more key-value pairs as needed for your API
    );

    $args = array(
        'headers' => array(
            'Authorization' => 'Basic ' . base64_encode('username:password')
        ),
        'body' => $data,
    );

    $response = wp_remote_post($api_url, $args);

    if (is_wp_error($response)) {
        // Handle error
    } else {
        // Handle the response
    }
}
```

By JWT

```
$api_url = 'https://api.example.com/add'; // Replace this with your API
$data = array(
    'key1' => 'value1',
    'key2' => 'value2'
    // Add more key-value pairs as needed for your API
);

$token = 'your_jwt_token_here';

$args = array(
    'headers' => array(
        'Authorization' => 'Bearer ' . $token
    ),
    'body' => $data,
);

$response = wp_remote_post($api_url, $args);

if (is_wp_error($response)) {
    // Handle error
} else {
    // Handle the response
}
}
```

Using Basic Authentication for Data Deletion:

php

 Copy code

```
function delete_data_with_basic_auth_from_api() {
    $api_url = 'https://api.example.com/delete'; // Replace this with your API URL
    $args = array(
        'headers' => array(
            'Authorization' => 'Basic ' . base64_encode('username:password')
        ),
        'method' => 'DELETE',
    );

    $response = wp_remote_request($api_url, $args);

    if (is_wp_error($response)) {
        // Handle error
    } else {
        // Handle the response
    }
}
```

Using JWT Token for Data Deletion:

php

 Copy code

```
function delete_data_with_jwt_from_api() {
    $api_url = 'https://api.example.com/delete'; // Replace this with your API URL
    $token = 'your_jwt_token_here';

    $args = array(
        'headers' => array(
            'Authorization' => 'Bearer ' . $token
        ),
        'method' => 'DELETE',
    );

    $response = wp_remote_request($api_url, $args);

    if (is_wp_error($response)) {
        // Handle error
    } else {
        // Handle the response
    }
}
```

Using Basic Authentication for Data Update:

php

 Copy code

```
function update_data_with_basic_auth_to_api() {
    $api_url = 'https://api.example.com/update'; // Replace this with your API URL
    $data = array(
        'key1' => 'new_value1',
        'key2' => 'new_value2'
        // Update key-value pairs as needed for your API
    );

    $args = array(
        'headers' => array(
            'Authorization' => 'Basic ' . base64_encode('username:password')
        ),
        'body' => $data,
        'method' => 'PUT', // or 'PATCH' depending on the API
    );

    $response = wp_remote_request($api_url, $args);

    if (is_wp_error($response)) {
        // Handle error
    } else {
        // Handle the response
    }
}
```

 Regenerate

```
function update_data_with_jwt_to_api() {
    $api_url = 'https://api.example.com/update'; // Replace this with your
    $data = array(
        'key1' => 'new_value1',
        'key2' => 'new_value2'
        // Update key-value pairs as needed for your API
    );
}

$token = 'your_jwt_token_here';

$args = array(
    'headers' => array(
        'Authorization' => 'Bearer ' . $token
    ),
    'body' => $data,
    'method' => 'PUT', // or 'PATCH' depending on the API
);

$response = wp_remote_request($api_url, $args);

if (is_wp_error($response)) {
    // Handle error
} else {
    // Handle the response
}
}
```

⟳ Re

Fetch API in to interact with a RESTful API and want to add new data (perform a POST request),

```
// Example data to be added
const newData = {
  name: 'John Doe',
  email: 'john@example.com',
  // Additional fields as needed
};

// API endpoint URL where you want to add the data
const apiUrl = 'https://api.example.com/data';

// Fetch request options
const requestOptions = {
  method: 'POST', // HTTP method for adding new data
  headers: {
    'Content-Type': 'application/json', // Specify the content type
    // Include any additional headers for authentication or other requirements
  },
  body: JSON.stringify(newData), // Convert data to JSON format
};

// Perform the POST request to add new data
fetch(apiUrl, requestOptions)
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok.');
    }
    return response.json();
})
  .then(data => {
    // Handle the response data after successfully adding the new data
    console.log('New data added:', data);
})
  .catch(error => {
    // Handle errors during the POST request
    console.error('There was an error adding new data:', error);
});
```

Wordpress Calling REST API

<https://www.youtube.com/watch?v=c8kBOWPKpyU>

```
wmk-tutor > wp-content > themes > astra > functions.php > display_weather
272 function display_weather() {
273
274     // Replace YOUR_API_KEY with your actual API key
275     $api_key = '8c23d6a317399d8a024419a3d28c04f0';
276
277     // Replace YOUR_CITY_NAME with the name of the city you want the weather for
278     $city = 'varanasi';
279
280     // Replace YOUR_COUNTRY_CODE with the ISO country code of the country the city is located in
281     $country_code = 'In';
282
283     // Create the API URL using the API key, city, and country code
284     $api_url = "https://api.openweathermap.org/data/2.5/weather?q={$city},{$country_code}&units=metric&appid={$api_key}";
285
286     // Get the JSON response from the API
287     $response = wp_remote_get($api_url);
288
289     // If the API request was successful, parse the JSON response and display the weather
290     if (is_array($response) && !is_wp_error($response)) {
291
292         $body = wp_remote_retrieve_body($response);
293         $data = json_decode($body);
294
295         // echo "<pre>";
296         // print_r($data);
297
298         if ($data) {
299             $temperature = round($data->main->temp);
300             $description = ucwords($data->weather[0]->description);
301             $icon = "https://openweathermap.org/img/w/{$data->weather[0]->icon}.png";
302             $cityname = $data->name ;
303             echo $cityname . " ";
304             $humidity = $data->main->humidity;
305             $Country = $data->sys->country;
306
307             echo "Toady Humudity is :" . $humidity . " ";
308
309             echo "<b>Country Name :</b>" . $Country;
310             echo "<div class='weather'>
311                 <img src='{$icon}' alt='{$description}' />
312                 <span>{$temperature}&deg;C, {$description}</span>
313             </div>";
314         }
315     }
316 }
317
318 add_shortcode('weather', 'display_weather');
319
```

JWT Authentication.

First install JWT Authentication plugin for REST API.

Second Configure plugin

```
> wp wp-config.php
 * Change these to different unique phrases! You can generate these using
 * the {@link https://api.wordpress.org/secret-key/1.1/salt/ WordPress.org secret-key service}.
 *
 * You can change these at any point in time to invalidate all existing cookies.
 * This will force all users to have to log in again.
 *
 * @since 2.6.0
 */
define( 'AUTH_KEY',         '3PvK-bQ0NH%qQ.[4eCA$16s#J~! :NWiB08Fnp^a{>Z{$93&$d.p=*]6ZtP&wYtP' );
define( 'SECURE_AUTH_KEY',  'zl@A)hiLPF2F<t~:=[ ]1u6x7uJ98hT)G!1=VL,-1qXYr$A./EnoBkPIL/5SX#j_C' );
define( 'LOGGED_IN_KEY',    'vy_2h&RempRC;l&ly#vZq)[-SEFe@OXPIpv3}fOrX^y@EvTP|e43$uTM=nT3eBNv' );
define( 'NONCE_KEY',        'L&VCF`YJfNNB34[lRVjEi_td=ZB^EvA1}QNXUZE8u/+}+e7.v]%)KIbaA/V#uBv>2' );
define( 'AUTH_SALT',        '|^$>_b&nj)VuZ[+DxZj58()l:&#G1TU=$0zi/<wrxz:+.}3>KGbd+6<_k+TV`7-' );
define( 'SECURE_AUTH_SALT', 'd(+49W/x8Lp.Hr4]LSz+M^9?p6;g1'y)kI6/6H(L8JbfE)8oom=BgYomTXQ[UQP%' );
define( 'LOGGED_IN_SALT',   'EbUstW(X`1}`A29qHzv=f0{eVIo(lo(l`n?VC1VmB2Fk2<XZ%8>`K_f3-dF/?*JT' );
define( 'NONCE_SALT',       '?ZV!ia(!jzi5A6V6GBR%Hx2Ua=N(5z 6*ZuaF,6Ws$liWna<o(pKVbY8/m[MlX1n' );

define('JWT_AUTH_SECRET_KEY', '-[<bE4_wI-%m*KBcKxA5C<:cI2*a,&*44CSd[c-PO>6&6qj)OJ3P,E5)Y=[0i|gq');
define('JWT_AUTH_CORS_ENABLE', true);

/**#@*/
```

Token Generation through Postman API Tool

The screenshot shows the Postman application interface. At the top, it displays a POST request to "localhost/wp-headless/server/wp-json/jwt-auth/v1/token". Below the URL, the method is set to POST and the target URL is "localhost/wp-headless/server/wp-json/jwt-auth/v1/token". The "Send" button is highlighted in blue. The "Body" tab is active, showing a JSON payload in the editor:

```
1
2   ...
3     "username": "admin",
4     "password": "admin123"
```

POST http://localhost/wp-hr

No Environment

http://localhost/wp-headless/server/wp-json/jwt-auth/v1/token

Save  

Send

Params Auth Headers (9) Body * Pre-req. Tests Settings Cookies Beautify

raw JSON

Body

200 OK 2.10 s 920 B Save Response

Pretty Raw Preview Visualize JSON

```

1
2   "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJpc3Mi0iJodHRw0i8vbG9jYWxob3N0L3dwLWh1YWRsZXNzL3NlcnZlciIsImhdCI6MTY3MjQ2NzQ4C
CwibmJmIjoxNjcyNDY3NDg4LCJleHAIoje2NzMwNzIy0DgsImRhdGEiOnsidXNlcI6eyJpZCI6IjEifD
19.j2VR64ok3yWM1PFe7T6Hu4BQyxR2EIIZcojmCR6FDDg",
3   "user_email": "admin@test.com",
4   "user_nicename": "admin",

```

localhost:3000/login

Login



admin@test.com

•••••••

Inspector Console Debugger Network Style Editor Performance Memory Storage Errors Warnings Logs Info Debug CSS XHR Filter Output

```

response
Object { data: {..}, status: 200, statusText: "OK", headers: {..}, config: {..}, request: XMLHttpRequest }
  config: Object { timeout: 0, xsrfCookieName: "XSRF-TOKEN", xsrfHeaderName: "X-XSRF-TOKEN", ... }
  data: Object { token:
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3Mi0iJodHRw0i8vbG9jYWxob3N0L3dwLWh1YWRsZXNzL3NlcnZlciIsImhdCI6MTY3NzY40TUzNCwibmJmIjoxNjcg5NT
    M0LCJleHAIoje2NzgyOTQzMzQsImRhdGEiOnsidXNlcI6eyJpZCI6IjEifX19.15ei8yutD1q2w2Aj8WirowCqp1At2LluEoQnplU3R0s", user_email: "admin@test.com",
    user_nicename: "admin", ... }
  token:
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3Mi0iJodHRw0i8vbG9jYWxob3N0L3dwLWh1YWRsZXNzL3NlcnZlciIsImhdCI6MTY3NzY40TUzNCwibmJmIjoxNjcg5N
    jgSNTM0LCJleHAIoje2NzgyOTQzMzQsImRhdGEiOnsidXNlcI6eyJpZCI6IjEifX19.15ei8yutD1q2w2Aj8WirowCqp1At2LluEoQnplU3R0s"
  user_display_name: "admin"
  user_email: "admin@test.com"
  user_nicename: "admin"
  <prototype>: Object { ... }
  headers: Object { "Content-Length": "220", "Content-Type": "application/json; charset=UTF-8" } 1ms 1ms https://localhost:3000/login

```

How to Add Custom Endpoints to WordPress API

<https://artisansweb.net/category/wordpress/>

<https://artisansweb.net/how-to-add-custom-endpoints-to-wordpress-api/>

Authorization Using WordPress REST API

WordPress REST API requires you to follow the Basic Auth flow. In the Basic Auth, you have to keep the unique token in the Authorization header while sending an API request. Starting WordPress 5.6, a new feature ‘Application Passwords’ is introduced in the system. This application password(with username) can be used to generate a token for the Authorization header.

The Application Passwords are available to all users on sites served over SSL/HTTPS. If for some reason you are not using SSL, you make it available using the below filter.

```
add_filter( 'wp_is_application_passwords_available', '__return_true' );
```

You will find the option for Application Passwords on the Users->Profile page. Generate the password by simply entering the Application Name.

The screenshot shows the 'Application Passwords' page in the WordPress admin. A new password has been generated: 'Ho9c 9vGs AOBG nXbO FPpr W5vO'. This password is highlighted with a red box. Below the password, a note says 'Be sure to save this in a safe location. You will not be able to retrieve it.' A table lists existing application passwords:

Name	Created	Last Used	Last IP	Revoke
Password for WordPress API	December 31, 2020	—	—	<button>Revoke</button>
WordPress App for JS	December 23, 2020	December 30, 2020	127.0.0.1	<button>Revoke</button>
Name	Created	Last Used	Last IP	Revoke

In the screenshot, you notice the spaces in the password. Application passwords can be used with or without spaces. If included, spaces will just be stripped out before the password is hashed and verified on the WordPress end.

Now, you are ready with the password. Next, to create an Auth token you have to create a Base64 encoded version of your username and application password. Let's say your username is 'admin' and the password is 'Ho9c 9vGs AOBG nXbO FPpr W5vO'. Use the following statement which gives you a final valid token.

```
<?php  
$username = 'admin';  
$application_password = 'Ho9c 9vGs A0BG nXb0 FPpr W5v0';  
  
echo base64_encode($username. ':' . $application_password);
```

The above statement returns a token value as

YWRtaW46SG85YyA5dkdzIEFPQkcgblhiMCBGUHByIFc1dk8=. This token you need to send in the Authorization header while calling WordPress REST API.

Login Using WordPress REST API

We are ready with the Basic Auth token value. Now, let's build a custom endpoint for a login system. In the below code, we write an API endpoint that receives user credentials and checks if the details are correct or not. You need to add this code to your themes `functions.php` file.

```
add_action(  
    'rest_api_init',  
    function () {  
        register_rest_route(  
            'api',  
            'login',  
            array(  
                'methods' => 'POST',  
                'callback' => 'login',  
            )  
        );  
    }  
);
```

WordPress provides an action `rest_api_init` to build the custom endpoints. Here I am using the `register_rest_route()` function which produces the above API endpoint as `YOUR_SITE_URL/wp-json/api/login`.

In our code 'api' is the namespace, 'login' is the route, a method is 'POST' and the callback function is 'login'. The callback method will have actual logic.

In order to write a logic for login flow, the POST parameters required are email and password which should be sent from the frontend along with the Authorization header. While posting this data, you need to send it in JSON format. For instance, from the VSCode using [Rest Client Extension](#) I send the POST request as shown in the screenshot below.

```
Send Request
POST http://localhost/wp/wp-json/api/login
Authorization: Basic YWRtaW46SG85YyA5dkdzIEFPQkcgblhiMCBGUHByIFc1dk8=
Content-Type: application/json

{
  "email": "admin@artisansweb.net",
  "password": "admin?123"
}
```

Here I have passed the token created in the previous step as the Authorization header value.

Add the below code for the `login()` method in the `functions.php` file.

```
<?php
function login( WP_REST_Request $request ) {
    $arr_request = json_decode( $request->get_body() );

    if ( ! empty( $arr_request->email ) && ! empty( $arr_request->password ) )
    {
        // this returns the user ID and other info from the user name.
        $user = get_user_by( 'email', $arr_request->email );

        if ( ! $user ) {
            // if the user name doesn't exist.
            return [
                'status' => 'error',
                'message' => 'Wrong email address.',
            ];
        }

        // check the user's login with their password.
        if ( ! wp_check_password( $arr_request->password, $user->user_pass,
$user->ID ) ) {
            // if the password is incorrect for the specified user.
            return [
                'status' => 'error',
                'message' => 'Wrong password.' ,
            ];
        }
    }
}
```

```

        'message' => 'Wrong password.',
    ];
}

return [
    'status' => 'success',
    'message' => 'User credentials are correct.',
];
} else {
    return [
        'status' => 'error',
        'message' => 'Email and password are required.',
    ];
}
}

?>

function login( WP_REST_Request $request ) {
    $arr_request = json_decode( $request->get_body() );

    if ( ! empty( $arr_request->email ) && ! empty( $arr_request->password ) ) {
        // this returns the user ID and other info from the user name.
        $user = get_user_by( 'email', $arr_request->email );

        if ( ! $user ) {
            // if the user name doesn't exist.
            return [
                'status' => 'error',
                'message' => 'Wrong email address.',
            ];
        }

        // check the user's login with their password.
        if ( ! wp_check_password( $arr_request->password, $user->user_pass, $user->user_id ) ) {
            // if the password is incorrect for the specified user.
            return [
                'status' => 'error',
                'message' => 'Wrong password.',
            ];
        }

        return [
            'status' => 'success',
            'message' => 'User credentials are correct.',
        ];
    } else {
        return [
            'status' => 'error',
            'message' => 'Email and password are required.',
        ];
    }
}

```

Note: If you received an error of “No route was found matching the URL and request method”, you need to update your permalink.

Upon receiving the ‘success’ value for the ‘status’ key, you can log the user in the frontend application.

Build an Endpoint for a GET Request

We have seen how to build custom endpoints for POST requests. Now, let’s look into the GET request using WordPress REST API. For this, I will write an API that deletes a user. From the front end, you should pass the id of a user as a GET parameter.

```
add_action(
    'rest_api_init',
    function () {
        register_rest_route(
            'api',
            'delete_user/(?P<id>\d+)',
            array(
                'methods' => 'GET',
                'callback' => 'delete_user',
            )
        );
    }
);
```

This code generates an API endpoint as *YOUR_SITE_URL/wp-json/api/delete_user/id*. To this endpoint, instead of id, you should pass the actual id of a user. I am taking this example for demonstration only.

When you delete the user, all posts created by the user also get deleted. Optionally, you can also reassign these posts. Read more about it in the [documentation](#).

The callback method `delete_user` will have the following code.

```
function delete_user( $data ) {
    // delete the user
    require_once(ABSPATH.'wp-admin/includes/user.php' );
    if ( wp_delete_user($data['id']) ) {
        return [
            'status' => 'success',
            'message' => 'User deleted successfully.',
        ];
    }

    return [
        'status' => 'error',
        'message' => 'It seems you passed the wrong user id.',
    ];
}
```

Custom Endpoint Tanuj para

<https://www.youtube.com/watch?v=-30MK1NDHrA&t=27s>

```
<?php
/*
 * Plugin Name:      IWS Custom Code
 * Description:     Add custom code
 * Author:          Tanuj G. Patra
 * Version:         1.0.0
 * Requires at least: 5.2
 * Requires PHP:    5.6
 */

remove_filter('the_excerpt', 'wpautop');
remove_filter('the_content', 'wpautop');

function iwsGetFeaturedImgSrc($obj, $fieldName, $request)
{
    if ($obj['featured_media']) {
        $img = wp_get_attachment_image_src($obj['featured_media'], 'full');
        return $img[0];
    }

    return false;
}

function iwsCreatePostWithFeaturedImg($request) {
    $params = $request->get_params();
    $title = sanitize_text_field($params['title']);
    $content = wp_kses_post($params['content']);

    $post = [
        'post_title' => $title,
        'post_content' => $content,
        'post_status' => 'publish',
        'post_type' => 'post'
    ];

    $postId = wp_insert_post($post);

    // Image upload
    if (isset($_FILES['featured_img']) && !empty($_FILES)) {
```

```

        $img = $_FILES['featured_img'];
        $img_name = str_replace(' ', '-', $title.'.'.explode('.',
$img['name'])[$img['name']]);
        $file = wp_upload_bits($img_name, null,
file_get_contents($img['tmp_name']));
        $fileType = wp_check_filetype($file['file'], null);

        $attachmentData = [
            'post_mime_type' => $fileType['type'],
            'post_title' => $img_name,
            'post_content' => '',
            'post_status' => 'inherit'
        ];
        $attachmentId = wp_insert_attachment($attachmentData, $file['file'],
$postId);

        set_post_thumbnail($postId, $attachmentId);
    }

    return [
        'status' => 'success',
        'post' => $postId
    ];
}

function iwsRegisterUser($request) {
    $params = $request->get_params();
    $username = sanitize_text_field($params['username']);
    $email = wp_kses_post($params['email']);
    $password = wp_kses_post($params['password']);

    $user = wp_create_user($username, $password, $email);

    if (is_wp_error($user)) {
        return ['status' => 'REGISTER_FAILED', 'error', $user-
>get_error_message()];
    }

    return [
        'user' => $user,
        'status' => 201,
    ];
}

add_action('rest_api_init', function () {

```

```
register_rest_field('post', 'featured_src', [
    'get_callback' => 'iwsGetFeaturedImgSrc',
]);

register_rest_route('wp/v2', 'create-post', [
    'methods' => "POST",
    'callback' => 'iwsCreatePostWithFeaturedImg'
]);

register_rest_route('wp/v2', 'register', [
    'methods' => "POST",
    'callback' => 'iwsRegisterUser'
]);
});
```

Wordpress official Custom Endpoint Method

<https://developer.wordpress.org/rest-api/requests/>

```
// Register our individual books endpoint.  
function prefix_register_book_route() {  
    register_rest_route( 'my-namespace/v1', '/books/(?P<id>\d+)', array(  
        // Supported methods for this endpoint. WP_REST_Server::READABLE  
        translates to GET.  
        'methods' => WP_REST_Server::READABLE,  
        // Register the callback for the endpoint.  
        'callback' => 'prefix_get_book',  
    ) );  
}  
  
add_action( 'rest_api_init', 'prefix_register_book_route' );  
  
/**  
 * Our registered endpoint callback. Notice how we are passing in $request as  
an argument.  
 * By default, the WP_REST_Server will pass in the matched request object to  
our callback.  
 *  
 * @param WP_REST_Request $request The current matched request object.  
 */  
function prefix_get_book( $request ) {  
    // Here we are accessing the path variable 'id' from the $request.  
    $book = prefix_get_the_book( $request['id'] );  
    return rest_ensure_response( $book );  
}  
  
// A simple function that grabs a book title from our books by ID.  
function prefix_get_the_book( $id ) {  
    $books = array(  
        'Design Patterns',  
        'Clean Code',  
        'Refactoring',  
        'Structure and Interpretation of Computer Programs',  
    );  
  
    $book = '';  
    if ( isset( $books[ $id ] ) ) {  
        // Grab the matching book.  
        $book = $books[ $id ];
```

```

    } else {
        // Error handling.
        return new WP_Error( 'rest_not_found', esc_html__( 'The book does not
exist', 'my-text-domain' ), array( 'status' => 404 ) );
    }

    return $book;
}

```

Query parameters exist in the query string portion of a URI. The query string portion of a URI in `https://ourawesomesite.com/wp-json/my-namespace/v1/books?per_page=2&genre=fiction` is `?per_page=2&genre=fiction`. The query string is started by the '?' character, the different values within the query string are separated by the '&' character. We specified two parameters in our query string; `per_page` and `genre`. In our endpoint we would want to grab only two books from the fiction genre. We could access those values in a callback like this: `$request['per_page']`, and `$request['genre']` (assuming $request is the name of the argument we are using)`. If you are familiar with PHP you have probably used query parameters in your web applications.

```

// Register our individual books endpoint.
function prefix_register_book_route() {
    register_rest_route( 'my-namespace/v1', '/books/(?P<id>\d+)', array(
        // Supported methods for this endpoint. WP_REST_Server::READABLE
        // translates to GET.
        'methods' => WP_REST_Server::READABLE,
        // Register the callback for the endpoint.
        'callback' => 'prefix_get_book',
    ) );
}

add_action( 'rest_api_init', 'prefix_register_book_route' );

/**
 * Our registered endpoint callback. Notice how we are passing in $request as
an argument.
 * By default, the WP_REST_Server will pass in the matched request object to
our callback.
 *
 * @param WP_REST_Request $request The current matched request object.

```

```

/*
function prefix_get_book( $request ) {
    // Here we are accessing the path variable 'id' from the $request.
    $book = prefix_get_the_book( $request['id'] );
    return rest_ensure_response( $book );
}

// A simple function that grabs a book title from our books by ID.
function prefix_get_the_book( $id ) {
    $books = array(
        'Design Patterns',
        'Clean Code',
        'Refactoring',
        'Structure and Interpretation of Computer Programs',
    );

    $book = '';
    if ( isset( $books[ $id ] ) ) {
        // Grab the matching book.
        $book = $books[ $id ];
    } else {
        // Error handling.
        return new WP_Error( 'rest_not_found', esc_html__( 'The book does not exist', 'my-text-domain' ), array( 'status' => 404 ) );
    }

    return $book;
}

```

Custom end point

<https://developer.wordpress.org/rest-api/extending-the-rest-api/adding-custom-endpoints/>

```

<?php
add_action( 'rest_api_init', function () {
    register_rest_route( 'myplugin/v1', '/author/(\?P<id>\d+)', array(
        'methods' => 'GET',
        'callback' => 'my_awesome_func',
    ) );
} );

```

Creating Endpoints

<https://developer.wordpress.org/rest-api/extending-the-rest-api/routes-and-endpoints/>

<https://developer.wordpress.org/rest-api/reference/categories/>

<https://developer.wordpress.org/rest-api/reference/pages/>

<https://developer.wordpress.org/rest-api/reference/posts/>

<https://developer.wordpress.org/rest-api/reference/application-passwords/>

If we wanted to create an endpoint that would return the phrase “Hello World, this is the WordPress REST API” when it receives a GET request, we would first need to register the route for that endpoint. To register routes you should use the `register_rest_route()` function. It needs to be called on the `rest_api_init` action hook. `register_rest_route()` handles all of the mapping for routes to endpoints. Let’s try to create a “Hello World, this is the WordPress REST API” route.

```
/**  
 * This is our callback function that embeds our phrase in a WP_REST_Response  
 */  
function prefix_get_endpoint_phrase() {  
    // rest_ensure_response() wraps the data we want to return into a  
    // WP_REST_Response, and ensures it will be properly returned.  
    return rest_ensure_response( 'Hello World, this is the WordPress REST  
    API' );  
}  
  
/**  
 * This function is where we register our routes for our example endpoint.  
 */  
function prefix_register_example_routes() {  
    // register_rest_route() handles more arguments but we are going to stick  
    // to the basics for now.  
    register_rest_route( 'hello-world/v1', '/phrase', array(  
        // By using this constant we ensure that when the WP_REST_Server  
        // changes our readable endpoints will work as intended.  
        'methods' => WP_REST_Server::READABLE,  
        // Here we register our callback. The callback is fired when this  
        // endpoint is matched by the WP_REST_Server class.  
        'callback' => 'prefix_get_endpoint_phrase',  
    ) );  
}  
  
add_action( 'rest_api_init', 'prefix_register_example_routes' );
```

HTTP Methods

HTTP methods are sometimes referred to as HTTP verbs. They are simply just different ways to communicate via HTTP. The main ones used by the WordPress REST API are:

- **GET** should be used for retrieving data from the API.
- **POST** should be used for creating new resources (i.e users, posts, taxonomies).
- **PUT** should be used for updating resources.
- **DELETE** should be used for deleting resources.
- **OPTIONS** should be used to provide context about our resources.

Vishal API Creation

[/domains/programmingwithvishal.com/public_html/api/index.php](http://domains/programmingwithvishal.com/public_html/api/index.php)

```
1 <?php
2 include('db.php');
3 $sql="select * from user";
4 $res=mysqli_query($con,$sql);
5 $count=mysqli_num_rows($res);
6 header('Content-Type:application/json');
7
8 if($count>0){
9     while($row=mysqli_fetch_assoc($res)){
10         $arr[]=$row;
11     }
12     echo json_encode(['status'=>'true','data'=>$arr,'result'=>'found']);
13 }else{
14     echo json_encode(['status'=>'true','data'=>'No data found','result'=>'not']);
15 }
16 ?>
```

```
index.php ×

index.php
1 <?php
2 $url="https://programmingwithvishal.com/api/";
3 $ch=curl_init();
4 curl_setopt($ch, CURLOPT_URL,$url);
5 curl_setopt($ch,CURLOPT_RETURNTRANSFER,true);
6 $result=curl_exec($ch);
7 curl_close($ch);
8 $result=json_decode($result,true);
9 if(isset($result['status'])){
10     if($result['status']==true){
11         if(isset($result['result'])){
12             if($result['result']==true){
13                 ?
14                 <table>
```

```
14 | | | | | <table>
15 | | | | | | <tr>
16 | | | | | | | <td>ID</td>
17 | | | | | | | <td>Name</td>
18 | | | | | | | <td>Email</td>
19 | | | | | </tr>
20 | | | | | <?php
21 | | | | | | foreach($result['data'] as $list){
22 | | | | | | | echo "<tr>
23 | | | | | | | | <td>".$list['id']."'".</td>
24 | | | | | | | | <td>".$list['name']."'".</td>
25 | | | | | | | | <td>".$list['email']."'".</td>
26 | | | | | | </tr>" ;
27 | | | | | }
28 | | | | | ?>
29 | | | | </table>
```

```
29 | | | | | </table>
30 | | | | | <?php
31 | | | | | }else{
32 | | | | | | echo $result['data'];
33 | | | | | }
34 | | | | |
35 | | | | }else{
36 | | | | | echo $result['data'];
37 | | | | }
38 | | | }else{
39 | | | | echo "API not working";
40 | | | }
41 | | | ?>
```

/domains/programmingwithvishal.com/public_html/api/index.php

```
1 <?php
2 include('db.php');
3 header('Content-Type:application/json');
4 if(isset($_GET['key'])){
5     $key=mysqli_real_escape_string($con,$_GET['key']);
6     $checkRes=mysqli_query($con,"select * from api_token where token
7         ='$key'");
8     if(mysqli_num_rows($checkRes)>0){
9         $checkRow=mysqli_fetch_assoc($checkRes);
10        if($checkRow['status']==1){
11            if($checkRow['hit_count']>=$checkRow['hit_limit']){
12                echo json_encode(['status'=>'false','data'=>'API hit limit
13                    exceed']);
14                die();
15            }else{
16                mysqli_query($con,"update api_token set hit_count=hit_count
17                    +1 where token='$key'");
18            }
19
20            $sql="select * from user";
21            $res=mysqli_query($con,$sql);
22            $count=mysqli_num_rows($res);
23
24            if($count>0){
25                while($row=mysqli_fetch_assoc($res)){
26                    $arr[]=$row;
27                }
28                echo json_encode(['status'=>'true','data'=>$arr,'result'
29                    =>'found']);
30            }else{
31                echo json_encode(['status'=>'true','data'=>'No data found'
32                    , 'result'=>'not']);
33            }
34        }else{
35            echo json_encode(['status'=>'false','data'=>'API key
36                    deactivated']);
37        }
38    }else{
39        echo json_encode(['status'=>'false','data'=>'Please provide
40                    valid api key']);
41    }
42}
43?>
```

+ Options

	id	token	hit_limit	hit_count	status
<input type="checkbox"/>  Edit  Copy  Delete	1	ahdhsasdnasbwldkgldk	5	5	1

```

index.php
1
2     s://programmingwithvishal.com/api/index.php?key=ahdhsasdinasbwldkgldk";
3     nit();
4     t($ch, CURLOPT_URL,$url);
5     t($ch,CURLOPT_RETURNTRANSFER,true);
6     rl_exec($ch);
7     ($ch);
8     on_decode($result,true);
9     result['status']){
10    sult['status']==true){
11        (isset($result['result'])){
12            if($result['result']==true){
13                ?>
14                <table>

```

Vishal Complete API CRED with Token Auth

+ Options								
		id	token		hit_limit	hit_count	status	
<input type="checkbox"/>		Edit		Copy		Delete	1	ahdhsasdinasbwldkgldk
<input type="checkbox"/>		Check all	With selected:			Edit		Copy
						Delete		Export

	user	<input type="checkbox"/> Show all	Number of rows: <input type="button" value="25"/>	Filter rows: <input type="text" value="Search this table"/>
+ Options				
		id	name	email

API token.php

```
1  <?php
2  include('db.php');
3  header('Content-Type:application/json');
4  if(isset($_GET['token'])){
5      $token=mysqli_real_escape_string($con,$_GET['token']);
6      $checkTokenRes=mysqli_query($con,"select * from api_token where tok
7      if(mysqli_num_rows($checkTokenRes)>0){
8          $checkTokenRow=mysqli_fetch_assoc($checkTokenRes);
9          if($checkTokenRow['status']==1){
10              |
11          }else{
12              $status='true';
13              $data="API token deactivated";
14              $code='3';
15          }
16      }else{
17          $status='true';
18          $data="Please provide valid API token";
19          $code='2';
20      }
}
```

API insert.php

```
1  <?php
2  include('token.php');
3  if(!isset($status)){
4      if(isset($_POST['name']) && isset($_POST['email'])){
5          $name=mysqli_real_escape_string($con,$_POST['name']);
6          $email=mysqli_real_escape_string($con,$_POST['email']);
7          mysqli_query($con,"insert into user(name,email) values('$name',
8          $status='true';
9          $data="Data inserted";
10         $code='8';
11     }else{
12         $status='true';
13         $data="Provide valid column count";
14         $code='9';
15     }
16 }
17 echo json_encode(["status"=>$status,"data"=>$data,"code"=>$code]);
18 ?>
```

API delete.php

```
1 <?php
2 include('token.php');
3 if(!isset($status)){
4 if(isset($_POST['id']) && $_POST['id']>0){
5     $id=mysqli_real_escape_string($con,$_POST['id']);
6     mysqli_query($con,"delete from user where id='$id'");
7     $status='true';
8     $data="Data deleted";
9     $code='7';
10 }else{
11     $status='true';
12     $data="Provide id";
13     $code='6';
14 }
15 }
16 echo json_encode(["status"=>$status,"data"=>$data,"code"=>$code]);
```

Consumer delete.php

```
* delete.php
2 if(isset($_GET['id']) && $_GET['id']>0){
3     $url="https://programmingwithvishal.com/api/delete.php?token=ahdhsa
4     $ch=curl_init();
5     $arr['id']=$_GET['id'];
6     curl_setopt($ch, CURLOPT_URL,$url);
7     curl_setopt($ch,CURLOPT_RETURNTRANSFER,true);
8     curl_setopt($ch,CURLOPT_POSTFIELDS,$arr);
9     $result=curl_exec($ch);
10    curl_close($ch);
11    $result=json_decode($result,true);
12    if(isset($result['status']) && !isset($result['code']) && $result['c
13        header('location:index.php');
14        die();
15    }else{
16        echo $result['data'];
17    }
18 }else{
19     header('location:index.php');
20     die();
21 }
```

API db.php

```
<?php
$con=mysqli_connect('localhost','username','password','db');
?>
```

API token.php

```
<?php
include('db.php');
header('Content-Type:application/json');
if(isset($_GET['token'])){
    $token=mysqli_real_escape_string($con,$_GET['token']);
    $checkTokenRes=mysqli_query($con,"select * from api_token where
token='$token'");
    if(mysqli_num_rows($checkTokenRes)>0){
        $checkTokenRow=mysqli_fetch_assoc($checkTokenRes);
        if($checkTokenRow['status']==1){

            }else{
                $status='true';
                $data="API token deactivated";
                $code='3';
            }
        }else{
            $status='true';
            $data="Please provide valid API token";
            $code='2';
        }
    }else{
        $status='true';
        $data="Please provide API token";
        $code='1';
    }
?>
```

API index.php

```
<?php
include('token.php');
if(!isset($status)){
    $sql="select * from user ";
    if(isset($_GET['id']) && $_GET['id']>0){
        $id=mysqli_real_escape_string($con,$_GET['id']);
        $sql.=" where id='".$id."'";
    }
    $sqlRes=mysqli_query($con,$sql);
    if(mysqli_num_rows($sqlRes)>0){
        $data=[];
        while($row=mysqli_fetch_assoc($sqlRes)){
            $data[]=$row;
        }
        $status='true';
        $code='5';
    }else{
        $status='true';
        $data="Data not found";
        $code='4';
    }
}
echo json_encode(["status"=>$status,"data"=>$data,"code"=>$code]);
?>
```

API delete.php

```
<?php
include('token.php');
if(!isset($status)){
    if(isset($_POST['id']) && $_POST['id']>0){
        $id=mysqli_real_escape_string($con,$_POST['id']);
        mysqli_query($con,"delete from user where id='".$id "'");
        $status='true';
        $data="Data deleted";
        $code='7';
    }else{
        $status='true';
        $data="Provide id";
        $code='6';
    }
}
echo json_encode(["status"=>$status,"data"=>$data,"code"=>$code]);
?>
```

```

API_manage_form.php
<?php
include('token.php');
if(!isset($status)){
    if(isset($_POST['name']) && isset($_POST['email'])){
        $name=mysqli_real_escape_string($con,$_POST['name']);
        $email=mysqli_real_escape_string($con,$_POST['email']);

        if(isset($_GET['id']) && $_GET['id']>0){
            $id=mysqli_real_escape_string($con,$_GET['id']);
            mysqli_query($con,"update user set name='$name',
email='$email' where id='$id'");
            $data="Data updated";
            $code='10';
        }else{
            mysqli_query($con,"insert into user(name,email)
values('$name','$email')");
            $data="Data inserted";
            $code='8';
        }
        $status='true';
    }else{
        $status='true';
        $data="Provide valid column count";
        $code='9';
    }
}
echo json_encode(["status"=>$status,"data"=>$data,"code"=>$code]);
?>

```

API Responce Code

1=>Please provide API token
 2=>Please provide valid API token
 3=>API token deactivated
 4=>Data not found
 5=>Data Return
 6=>Provide id
 7=>Data deleted
 8=>Data Inserted
 9=>Provide valid column count
 10=>Data updated

Consumer form.php

```
<?php
$msg="";
$name="";
$email="";
if(isset($_POST['submit'])){
    $arr['name']=$_POST['name'];
    $arr['email']=$_POST['email'];
    $id="";
    if(isset($_GET['id']) && $_GET['id']>0){
        $id=$_GET['id'];
    }
    $url="api_url/manage_form.php?token=ahdhsasdinasbwldkgl&id=".$id;
    $ch=curl_init();
    curl_setopt($ch, CURLOPT_URL,$url);
    curl_setopt($ch,CURLOPT_RETURNTRANSFER,true);
    curl_setopt($ch,CURLOPT_POSTFIELDS,$arr);
    $result=curl_exec($ch);
    curl_close($ch);
    $result=json_decode($result,true);
    $msg=$result['data'];
}

if(isset($_GET['id']) && $_GET['id']>0){
    $url="https://programmingwithvishal.com/api/index.php?token=ahdhsasdinasbw
dldkgl&id=".$_GET['id'];
    $ch=curl_init();
    curl_setopt($ch, CURLOPT_URL,$url);
    curl_setopt($ch,CURLOPT_RETURNTRANSFER,true);
    $result=curl_exec($ch);
    curl_close($ch);
    $result=json_decode($result,true);
    if(isset($result['status']) && isset($result['code']) &&
$result['code']==5){
        $name=$result['data'][0]['name'];
        $email=$result['data'][0]['email'];
    }else{
        header('location:index.php');
        die();
    }
}

?>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>API Crud Operation</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
        href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
        <script
            src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
        <script
            src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
        <link rel="stylesheet"
        href="https://fonts.googleapis.com/icon?family=Material+Icons">
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
        <link rel="stylesheet" href="css/style.css">
</head>
<body>
<div class="container-xl">
    <div class="table-responsive">
        <div class="table-wrapper">
            <div class="table-title">
                <div class="row">
                    <div class="col-sm-5">
                        <h2>Manage User</h2>
                    </div>
                    <div class="col-sm-7">
                        <a href="index.php" class="btn btn-secondary"><i
                            class="material-icons arrow_back">#xe5c4;</i>
                        <span>Back</span></a>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>

<div class="row" id="form_bg">
    <?php echo $msg?>
    <form method="post">
        <div class="col-sm-4">
            <div class="form-group">
```

```

        <input type="name" class="form-control" id="name"
name="name" placeholder="Enter name" required value="<?php echo $name?>">
    </div>
</div>
<div class="col-sm-4">
    <div class="form-group">
        <input type="email" class="form-control" id="nombre"
name="email" placeholder="Enter email" required value="<?php echo $email?>">
    </div>
</div>
<div class="col-sm-1">
    <div class="form-group">
        <input type="submit" value="Submit" class="btn btn-info
btn-block" name="submit">
    </div>
</div>
</form>
</div>
</div>
</body>
</html>

```

Manage User ← Back

Manage User ← Back

Consumer index.php

User			Add New User
#	Name	Email	Action
3	Bhaavit	bhaavit@gmail.com	 
4	Vishal	vishal@gmail.com	 
5	Amit	amit@gmail.com	 

```
<?php
$url="api_url/index.php?token=ahdhsasdinasbwldkgl dk";
$ch=curl_init();
curl_setopt($ch, CURLOPT_URL,$url);
curl_setopt($ch,CURLOPT_RETURNTRANSFER,true);
$result=curl_exec($ch);
curl_close($ch);
$result=json_decode($result,true);
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>API Crud Operation</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
        <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
        <link rel="stylesheet"
href="https://fonts.googleapis.com/icon?family=Material+Icons">
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
        <link rel="stylesheet" href="css/style.css">
<script>
$(document).ready(function(){
    $('[data-toggle="tooltip"]').tooltip();
});
</script>
</head>
<body>
```

```
<div class="container-xl">
    <div class="table-responsive">
        <div class="table-wrapper">
            <div class="table-title">
                <div class="row">
                    <div class="col-sm-5">
                        <h2>User</h2>
                    </div>
                    <div class="col-sm-7">
                        <a href="form.php" class="btn btn-secondary"><i
class="material-icons">+</i> <span>Add New
User</span></a>
                    </div>
                </div>
            <?php
                if(isset($result['status']) && isset($result['code']) &&
$result['code']==5){
                    ?>
                    <table class="table table-striped table-hover">
                        <thead>
                            <tr>
                                <th>#</th>
                                <th>Name</th>
                                <th>Email</th>
                                <th>Action</th>
                            </tr>
                        </thead>
                        <tbody>
                            <?php
                                foreach($result['data'] as $list){
                                    ?>
                                    <tr>
                                        <td><?php echo $list['id']?></td>
                                        <td><?php echo $list['name']?></td>
                                        <td><?php echo $list['email']?></td>
                                        <td>
                                            <a href="form.php?id=<?php echo
$list['id']?>" class="edit" title="Edit"><i class="material-icons
colorize">+</i></a>
                                            <a href="delete.php?id=<?php echo
$list['id']?>" class="delete" title="Delete" ><i class="material-
icons">-</i></a>
                                        </td>
                                    </tr>
                                }
                            </tbody>
                        </table>
                    <?php
                }
            </?php
        }
    </div>
</div>
```

```

                <?php
            }
        ?>
        </tbody>
    </table>
    <?php
}else{
    echo $result['data'];
}
?>
</div>
</div>
</div>
</body>
</html>

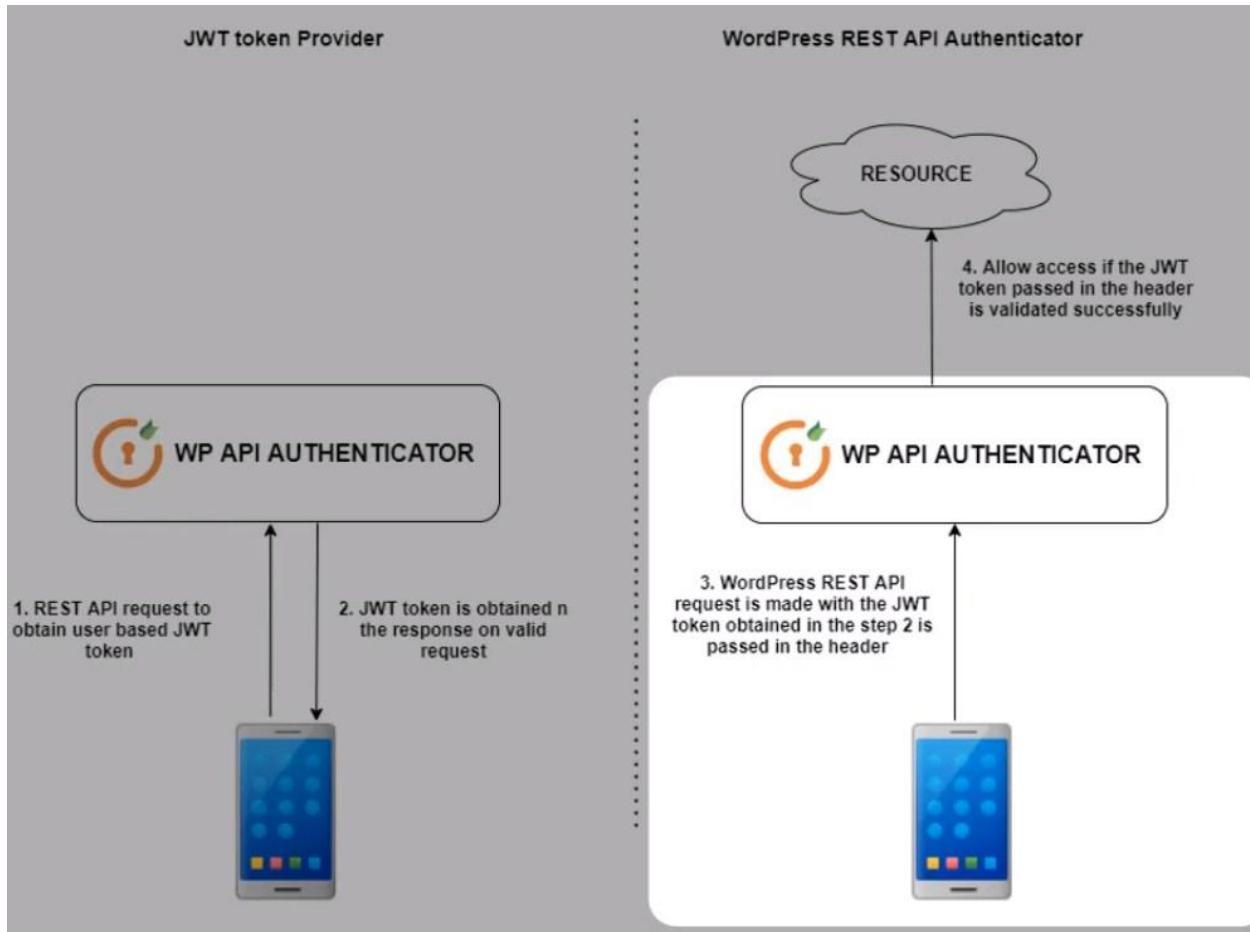
```

Consumer delete.php

```

<?php
if(isset($_GET['id']) && $_GET['id']>0){
    $url="api_url/delete.php?token=ahdhsasdnasbwldkgldk";
    $ch=curl_init();
    $arr['id']=$_GET['id'];
    curl_setopt($ch, CURLOPT_URL,$url);
    curl_setopt($ch,CURLOPT_RETURNTRANSFER,true);
    curl_setopt($ch,CURLOPT_POSTFIELDS,$arr);
    $result=curl_exec($ch);
    curl_close($ch);
    $result=json_decode($result,true);
    if(isset($result['status']) && !isset($result['code'])) &&
$result['code']==7){
        header('location:index.php');
        die();
    }else{
        echo $result['data'];
    }
}else{
    header('location:index.php');
    die();
}
?>

```



Rest API basic authentication in WordPress

In WordPress, you can use the HTTP API to make requests to a REST API that requires basic authentication. You can use the `wp_remote_request()` function with the appropriate authentication headers. For instance:

```
$url = 'https://your-api-endpoint.com/wp-json/your-plugin/v1/data';
$username = 'your_username';
$password = 'your_password';

$response = wp_remote_request( $url, array(
    'headers' => array(
        'Authorization' => 'Basic ' . base64_encode( $username . ':' . $password )
    ),
) );
// Process $response as needed
```

```
'Authorization' => 'Basic ' . base64_encode( $username . ':' . $password ),
```

Jwt token authentication in rest API in WordPress

WordPress supports JWT (JSON Web Tokens) for authentication in its REST API. You can use plugins like "JWT Authentication for WP REST API" or code snippets to implement JWT token authentication.

First, install the plugin and follow its instructions to set it up. Then, for making authenticated requests to the REST API using JWT, you typically acquire a token by authenticating via a specific endpoint. Once you have the token, you include it in the headers of subsequent requests.

Here's a general example of making an authenticated request using a JWT token:

```
$token = 'YOUR_JWT_TOKEN_HERE';
$url = 'https://your-api-endpoint.com/wp-json/your-plugin/v1/data';

$response = wp_remote_request( $url, array(
    'headers' => array(
        'Authorization' => 'Bearer ' . $token,
    ),
) );
// Process $response as needed
```

Basic authentication in PHP involves sending an HTTP request with the appropriate headers that include a username and password. Here's an example using PHP's cURL, a popular way to make HTTP requests:

```
$url = 'https://api.example.com/data';
$username = 'your_username';
$password = 'your_password';

$ch = curl_init($url);

// Set up basic authentication
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
curl_setopt($ch, CURLOPT_USERPWD, "$username:$password");

// Other options
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true); // Set to receive the response as a string

$response = curl_exec($ch);

// Check for errors
if($response === false){
    echo 'Curl error: ' . curl_error($ch);
} else {
    // Process $response as needed
    echo $response;
}

curl_close($ch);
```

To fetch all posts from a WordPress site using the REST API, you can make a GET request to the `wp/v2/posts` endpoint. Here's an example using PHP's cURL:

```
$url = 'https://your-wordpress-site.com/wp-json/wp/v2/posts';

$ch = curl_init($url);

curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($ch);

if($response === false){
    echo 'Curl error: ' . curl_error($ch);
} else {
    // Process $response if needed
    echo $response;
}

curl_close($ch);
```

By wordpress

```
$response = wp_remote_get('https://your-wordpress-site.com/wp-json/wp/v2/post');

if ( is_wp_error( $response ) ) {
    echo 'Error occurred: ' . $response->get_error_message();
} else {
    $posts = json_decode( wp_remote_retrieve_body( $response ), true );
    // Handle the retrieved posts
    print_r($posts);
}
```

By Jquery method, To fetch all posts from a WordPress site using the REST API,

javascript

 Copy code

```
$.ajax({
    url: 'https://your-wordpress-site.com/wp-json/wp/v2/posts',
    type: 'GET',
    dataType: 'json',
    success: function(posts) {
        // Process the retrieved posts
        console.log(posts);
        // Example: Loop through each post and do something
        posts.forEach(function(post) {
            console.log(post.title.rendered); // Output the title of each post
        });
    },
    error: function(error) {
        console.log('Error:', error);
    }
});
```

How to add new post in wordpress by rest api

Adding a new post in WordPress using the REST API involves sending a POST request to the `wp/v2/posts` endpoint with the necessary post data. You'll need to include specific headers, authentication, and the post content in your request.

```
$url = 'https://your-wordpress-site.com/wp-json/wp/v2/posts';
$token = 'YOUR_JWT_TOKEN_HERE'; // Include your authentication token
$post_data = array(
    'title' => 'Your Post Title',
    'content' => 'Your post content goes here.',
    'status' => 'publish' // Set the status as needed (draft, pending, publish)
);

$headers = array(
    'Content-Type: application/json',
    'Authorization: Bearer ' . $token,
);

$ch = curl_init($url);

curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($post_data));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($ch);

if($response === false){
    echo 'Curl error: ' . curl_error($ch);
} else {
    // Process $response if needed
    echo $response;
}

curl_close($ch);
```

How to add new post in wordpress by rest api by jquery ajax

```
// Post data to be added
var newPost = {
    title: 'New Post Title',
    content: 'Content of the new post.',
    status: 'publish' // Set the status as needed (draft, pending, publish,
};

// Token for authentication (if required)
var token = 'YOUR_JWT_TOKEN_HERE';

$.ajax({
    url: 'https://your-wordpress-site.com/wp-json/wp/v2/posts',
    type: 'POST',
    data: JSON.stringify(newPost),
    contentType: 'application/json',
    beforeSend: function(xhr) {
        xhr.setRequestHeader('Authorization', 'Bearer ' + token);
    },
    success: function(response) {
        console.log('New post added:', response);
        // Do something with the response
    },
    error: function(error) {
        console.log('Error:', error);
    }
});
```

Reg

By wordpress

Another way

```
$token = 'YOUR_TOKEN_HERE'; // The token for authentication
$url = 'https://target-site.com/wp-json/wp/v2/posts'; // URL of the target WordPress site

$new_post = array(
    'title' => 'New Post Title',
    'content' => 'Content of the new post.',
    'status' => 'publish'
);

$request = wp_remote_request(
    $url,
    array(
        'method' => 'POST',
        'headers' => array(
            'Authorization' => 'Bearer ' . $token,
            'Content-Type' => 'application/json'
        ),
        'body' => wp_json_encode($new_post)
    )
);

if (is_wp_error($request)) {
    echo 'Error occurred: ' . $request->get_error_message();
} else {
    $response_code = wp_remote_retrieve_response_code($request);
    $response_body = wp_remote_retrieve_body($request);

    if ($response_code === 201) {
        echo 'New post added successfully.';
    } else {
        echo 'Failed to add a new post. Response: ' . $response_body;
    }
}
```

Another way

```
$url = 'https://target-site.com/wp-json/wp/v2/posts'; // URL of the target
$username = 'your_username';
$password = 'your_password';

$new_post = array(
    'title' => 'New Post Title',
    'content' => 'Content of the new post.',
    'status' => 'publish'
);

$credentials = base64_encode( $username . ':' . $password );

$args = array(
    'headers' => array(
        'Authorization' => 'Basic ' . $credentials,
        'Content-Type' => 'application/json'
    ),
    'body' => wp_json_encode($new_post),
);

$response = wp_remote_post( $url, $args );

if ( is_wp_error( $response ) ) {
    echo 'Error occurred: ' . $response->get_error_message();
} else {
    $response_code = wp_remote_retrieve_response_code($response);
    if ( $response_code === 201 ) {
        echo 'New post added successfully.';
    } else {
        echo 'Failed to add a new post. Response code: ' . $response_code;
    }
}
```

To delete a specific post in WordPress using the REST API

```
$url = 'https://your-wordpress-site.com/wp-json/wp/v2/posts/POST_ID';
$token = 'YOUR_JWT_TOKEN_HERE'; // Include your authentication token

$headers = array(
    'Authorization: Bearer ' . $token,
);

$ch = curl_init($url);

curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($ch);

if($response === false){
    echo 'Curl error: ' . curl_error($ch);
} else {
    // Process $response if needed
    echo $response;
}

curl_close($ch);
```

To delete specific post in wp by rest api by jquery ajax

```
<button id="deletePostBtn">Delete Post</button>
<div id="responseMessage"></div>
```

```
ent).ready(function() {
$('#deletePostBtn').click(function() {
var postId = 123; // Replace with the ID of the post you want to delete
var token = 'YOUR_JWT_TOKEN_HERE'; // Include your authentication token

$.ajax({
  type: 'DELETE',
  url: 'https://your-wordpress-site.com/wp-json/wp/v2/posts/' + postId,
  beforeSend: function(xhr) {
    xhr.setRequestHeader('Authorization', 'Bearer ' + token);
  },
  success: function(response) {
    $('#responseMessage').text('Post deleted successfully: ' + response);
    // Handle the response after post deletion
  },
  error: function(error) {
    $('#responseMessage').text('Error deleting post: ' + error);
    // Handle errors during post deletion
  }
});
});
```

To update specific post in wp by rest api

```
$url = 'https://your-wordpress-site.com/wp-json/wp/v2/posts/POST_ID';
$token = 'YOUR_JWT_TOKEN_HERE'; // Include your authentication token

$post_data = array(
    'title' => 'Updated Post Title',
    'content' => 'Updated post content goes here.',
);

$headers = array(
    'Content-Type: application/json',
    'Authorization: Bearer ' . $token,
);

$ch = curl_init($url);

curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST"); // or "PUT" depending on yo
curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($post_data));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($ch);

if($response === false){
    echo 'Curl error: ' . curl_error($ch);
} else {
    // Process $response if needed
    echo $response;
}

curl_close($ch);
```

To update specific post in wp by rest api by jquery

```
<button id="updatePostBtn">Update Post</button>
<div id="responseMessage"></div>
```

```
ent).ready(function() {
    $('#updatePostBtn').click(function() {
        var postId = 123; // Replace with the ID of the post you want to update
        var token = 'YOUR_JWT_TOKEN_HERE'; // Include your authentication token

        var updatedPost = {
            title: 'Updated Post Title',
            content: 'Updated post content goes here.',
        };

        $.ajax({
            type: 'POST', // or 'PUT' depending on your server configuration
            url: 'https://your-wordpress-site.com/wp-json/wp/v2/posts/' + postId,
            beforeSend: function(xhr) {
                xhr.setRequestHeader('Authorization', 'Bearer ' + token);
            },
            data: JSON.stringify(updatedPost),
            contentType: 'application/json',
            success: function(response) {
                $('#responseMessage').text('Post updated successfully: ' + response);
                // Handle the response after post update
            },
            error: function(error) {
                $('#responseMessage').text('Error updating post: ' + error);
                // Handle errors during post update
            }
        });
    });
});
```

Reg

How to fetch all user of wordpress in through rest api by js

javascript

 Copy code

```
fetch('https://your-wordpress-site.com/wp-json/wp/v2/users')
  .then(response => response.json())
  .then(users => {
    // Handle the retrieved user data
    console.log(users);
  })
  .catch(error => {
    // Handle any errors during the request
    console.error('Error fetching users:', error);
  });
});
```

By Jquery

```
$.ajax({
  url: 'https://your-wordpress-site.com/wp-json/wp/v2/users',
  type: 'GET',
  dataType: 'json',
  success: function(users) {
    // Handle the retrieved user data
    console.log(users);
  },
  error: function(error) {
    // Handle any errors during the request
    console.error('Error fetching users:', error);
  }
});
```

By php curl

```
php Copy code

$url = 'https://your-wordpress-site.com/wp-json/wp/v2/users';

$ch = curl_init($url);

curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($ch);

if($response === false){
    echo 'Curl error: ' . curl_error($ch);
} else {
    // Process the retrieved user data
    $users = json_decode($response, true);
    var_dump($users);
}

curl_close($ch);
```

How to create user in wp through rest api by php curl

```
$url = 'https://your-wordpress-site.com/wp-json/wp/v2/users';
$token = 'YOUR_JWT_TOKEN_HERE'; // Include your authentication token

$new_user_data = array(
    'username' => 'newuser',
    'email' => 'newuser@example.com',
    'password' => 'password123',
    'roles' => ['subscriber'] // Set user roles as needed
);

$ch = curl_init($url);

curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($new_user_data));
curl_setopt($ch, CURLOPT_HTTPHEADER, array(
    'Content-Type: application/json',
    'Authorization: Bearer ' . $token,
));
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($ch);

if($response === false){
    echo 'Curl error: ' . curl_error($ch);
} else {
    // Process the response after creating the user
    var_dump(json_decode($response, true));
}

curl_close($ch);
```

By Jquery

```
<button id="createUserBtn">Create User</button>
<div id="responseMessage"></div>

$(document).ready(function() {
    $('#createUserBtn').click(function() {
        var token = 'YOUR_JWT_TOKEN_HERE'; // Include your authentication token here

        var newUser = {
            username: 'newuser',
            email: 'newuser@example.com',
            password: 'password123',
            roles: ['subscriber'] // Set user roles as needed
        };

        $.ajax({
            type: 'POST',
            url: 'https://your-wordpress-site.com/wp-json/wp/v2/users',
            data: JSON.stringify(newUser),
            contentType: 'application/json',
            beforeSend: function(xhr) {
                xhr.setRequestHeader('Authorization', 'Bearer ' + token);
            },
            success: function(response) {
                $('#responseMessage').text('User created successfully. User ID: ' + response.id);
                // Handle the response after creating the user
            },
            error: function(error) {
                $('#responseMessage').text('Error creating user: ' + error.message);
                // Handle errors during user creation
            }
        });
    });
});
```

```
function(response) {
    $('#responseMessage').text('User created successfully. User ID: ' + response.id);
}
```

Complete Rest API

https://tutorialsclass.com/php-rest-api-file_get_contents/

<https://dummy.restapiexample.com/>

PHP: REST API – file_get_contents()

There are two popular methods to fetch REST API data in PHP. Either you can use PHP built-in function `file_get_contents()` or use CURL commands.

Working with JSON data in API

XML & JSON are two mostly used API data format. In this chapter, we will use an API that works with JSON data.

Most of the time, it is easy to read or get data from REST API URL. Most applications create dedicated URLs (also called endpoints or routes) to provide access to their data for other applications.

As most APIs use HTTP GET method to read API data, you can simply hit URL in the browser to view data for testing. This is because when you HIT any URL in the browser, it sends GET request and access data. You can search for dummy or sample REST API URLs online for practice.

SAMPLE JSON data format in APIs

We will use Dummy REST API Example website to work with live API data in our examples. This website provides a number of routes or URLs to read, create, update, delete data using API. We will use the following to read employees data.

GET Employees data using API URL: <http://dummy.restapiexample.com/>

Above URL provides employee's profile data in JSON format something similar to the following:

```
[{"id": "110", "employee_name": "Robin", "employee_salary": "54000", "employee_age": "65"},  
 {"id": "112", "employee_name": "Deepak", "employee_salary": "12350", "employee_age": "29"},  
 {"id": "114", "employee_name": "Ankit", "employee_salary": "90000", "employee_age": "18"}]
```

Code to read API data using PHP file_get_contents() function

PHP inbuilt `file_get_contents()` function is used to read a file into a string. This can read any file or API data using URLs and store data in a variable. This function is the easiest method to read any data by passing API URL.

In the following PHP program, we will use a sample API URL that provides employee's data and displays them.

```
<?php

$api_url = 'https://dummy.restapiexample.com/api/v1/employees';

// Read JSON file
$json_data = file_get_contents($api_url);

// Decode JSON data into PHP array
$response_data = json_decode($json_data);

// All user data exists in 'data' object
$user_data = $response_data->data;

// Cut long data into small & select only first 10 records
$user_data = array_slice($user_data, 0, 9);

// Print data if need to debug
//print_r($user_data);

// Traverse array and display user data
foreach ($user_data as $user) {
    echo "name: ".$user->employee_name;
    echo "<br />";
    echo "name: ".$user->employee_age;
    echo "<br /> <br />";
}

?>
```

Code Explanation

1. Pass API URL in `file_get_contents()` to fetch data
2. Decode JSON data into PHP array
3. Select only first 10 records by slicing array. Because given feed data is large, this may slow down your browser to show them all.
4. Traverse user data array and display the required information.

Using the PHP `file_get_contents()` function, we can read files or API data but can not perform write, update, delete operations. In the next chapters, we will use cURL methods to perform all those operations using API data.

PHP : REST API – GET data using cURL

In this tutorial, we will learn how to GET API data using cURL functions.

What is cURL?

cURL (stands for client URL) is a command-line tool to transfer data between servers. It supports various protocols such as HTTP, HTTPS, FTP, FTPS, IMAP etc. Most programming languages support cURL commands using some libraries or extension.

cURL is one of the popular methods to interact with REST API data. PHP use cURL library (module) to provide access curl functions within PHP. By default, most PHP installations come with cURL support is enabled.

How to use cURL in PHP

The basic process of PHP cURL can be divided into four parts.

- Initialize a cURL session using `$curl_handle = curl_init();`
- Pass URL option in the cURL session `curl_setopt($curl_handle, CURLOPT_URL, $url);`
- Execute cURL session & store data: `$api_data = curl_exec($handle);`
- Close cURL session: `curl_close($curl_handle);`

Sample REST API data in JSON

We will use Dummy REST API Example website to work with live API data in our examples. This website provides a number of routes or URLs to read, create, update, delete employee's data using API. We will use following to get employees data.

GET Employees data using API URL: <https://dummy.restapiexample.com/api/v1/employees>

Above URL provides employee's profile data in JSON format something similar to following:

```
[{"id": "110", "employee_name": "Robin", "employee_salary": "54000", "employee_age": "65"}, {"id": "112", "employee_name": "Deepak", "employee_salary": "12350", "employee_age": "29"}, {"id": "114", "employee_name": "Ankit", "employee_salary": "90000", "employee_age": "18"}]
```

PHP program to GET REST API data using cURL

In the following PHP program, we will use a sample PHP CURL function to get employee's data and displays them.

```
<?php
// Initiate curl session in a variable (resource)
$curl_handle = curl_init();

$url = "https://dummy.restapiexample.com/api/v1/employees";

// Set the curl URL option
curl_setopt($curl_handle, CURLOPT_URL, $url);

// This option will return data as a string instead of direct output
curl_setopt($curl_handle, CURLOPT_RETURNTRANSFER, true);

// Execute curl & store data in a variable
$curl_data = curl_exec($curl_handle);

curl_close($curl_handle);

// Decode JSON into PHP array
$response_data = json_decode($curl_data);
// Print all data if needed
// print_r($response_data);
// die();

// All user data exists in 'data' object
$user_data = $response_data->data;

// Extract only first 5 user data (or 5 array elements)
$user_data = array_slice($user_data, 0, 4);

// Traverse array and print employee data
foreach ($user_data as $user) {
    echo "name: ".$user->employee_name;
    echo "<br />";
}

?>
```

Code Explanation

First, we need to initiate a curl and pass your API URL. Then execute CURL & store received data. As API URL returns JSON data, we need to decode it using json_decode. Now you have all data as an array which you can traverse using foreach and display accordingly.

PHP : REST API – POST data using cURL

In this tutorial, we will learn how to POST API data using cURL functions. An HTTP POST is often used to create new content.

If you are not familiar with basic cURL process and functions, read the previous tutorial: PHP: REST API – GET data using cURL

SAMPLE REST API data to send via POST Method

We will use Dummy REST API Example website to work with HTTP POST method and create a new employee record.

API URL to create a new Employee: <https://dummy.restapiexample.com/api/v1/create>

To create a new employee, we need to POST data in following JSON format. We will use cURL to send this data.

```
[{"id": "110", "employee_name": "Deepak Kumar", "employee_salary": "74000", "employee_age": "42"}]
```

PHP program to POST REST API data using cURL

In the following PHP program, we will send new user details in given JSON format. This data will be passed to the given URL using HTTP POST method in cURL and create a new user.

```
<?php

// User data to send using HTTP POST method in curl
$data = array('name'=>'New User 123', 'salary'=>'65000', 'age' => '33');

// Data should be passed as json format
$data_json = json_encode($data);

// API URL to send data
$url = 'https://dummy.restapiexample.com/api/v1/employees';

// curl initiate
$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, $url);

curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: application/json'));
```

```
// SET Method as a POST
curl_setopt($ch, CURLOPT_POST, 1);

// Pass user data in POST command
curl_setopt($ch, CURLOPT_POSTFIELDS,$data_json);

curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Execute curl and assign returned data
$response = curl_exec($ch);

// Close curl
curl_close($ch);

// See response if data is posted successfully or any error
print_r ($response);

?>
```

Code Explanation

First, we have prepared data for a new user in JSON format. Then we passed those user data to given API URL via HTTP POST Method. After that, we executed CURL & stored received data. This will create a new user with data send as JSON.

If a similar user name exists in the API, it will throw an error for duplicate records. Therefore, change user data in JSON and try again.

After successful user creation, you can find a new user at the end of the API data. You can read employee data on the following URL.

API URL to GET Employees data: <http://dummy.restapiexample.com/api/v1/employees>

In the next chapters, we will use cURL methods to perform an update and delete operations using API data.

PHP : REST API – PUT (Update) data using cURL

In this tutorial, we will learn how to use HTTP PUT to update REST API data using cURL functions.

If you are not familiar with basic cURL process and functions, read the previous tutorial: [PHP: REST API – GET data using cURL](#)

SAMPLE REST API data to send via PUT Method

We will use [Dummy REST API Example](#) website to work with HTTP PUT method and employee record.

API URL to create a new Employee: <https://dummy.restapiexample.com/api/v1/update/10>

To update an employee, we need to pass the id of that employee in URL with updated details in following JSON format. We will use cURL to send this data.

```
[{"id": "110", "employee_name": "Robin S", "employee_salary": "7270", "employee_age": "34"}]
```

PHP program to UPDATE user data using PUT REST API command via cURL

In the following PHP program, we will update the user with id '10'. This data will be passed to the given URL using HTTP POST method in cURL and create a new user.

```
<?php

// User data to send using HTTP PUT method in curl
$data = array('name'=>'New Robin User', 'salary'=>'62000', 'age' => '34');

// Data should be passed as json format
$data_json = json_encode($data);

// API URL to update data with employee id
$url = 'https://dummy.restapiexample.com/api/v1/update/21';

// curl initiate
$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, $url);
```

```
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: application/json', 'Content-Length: ' . strlen($data_json)));

// SET Method as a PUT
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'PUT');

// Pass user data in POST command
curl_setopt($ch, CURLOPT_POSTFIELDS,$data_json);

curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Execute curl and assign returned data
$response = curl_exec($ch);

// Close curl
curl_close($ch);

// See response if data is posted successfully or any error
print_r ($response);

?>
```

Code Explanation

First, we have used API URL for employees (with user-id at the end) whose details need to be updated. Then, we passed updated user data to the given API URL via HTTP POST Method. After that, we executed CURL & stored received response to see if any error.

If a similar user data exists or not user id found in the API, it will throw an error for duplicate records. Therefore, pass the user id in JSON carefully and try again.

Note: Make sure not to add '/' at the end of ID or URL in this API.

After successful user creation, you can find a new user at the end of the API data. You can read employee data on the following URL.

API URL to GET Employees data: <https://dummy.restapiexample.com/api/v1/employees>

In the next chapters, we will use cURL methods to perform an update and delete operations using API data.

PHP : REST API – DELETE data using cURL

In this tutorial, we will learn how to DELETE some REST API data using cURL functions in PHP.

If you are not familiar with basic cURL process and functions, read the previous tutorial: [PHP: REST API – GET data using cURL](#)

SAMPLE REST API to DELETE data using cURL

We will use [Dummy REST API Example website](#) to work with HTTP DELETE method using cURL.

- API URL to DELETE an employee: <https://dummy.restapiexample.com/api/v1/delete/17>

In this API URL (also called route or endpoint), you will use the particular employee id to delete his records.

Note: The DELETE API URL is actually incorrect in the given (example) website. Please use '/delete/' instead of '/update/' as mentioned above.

PHP program to DELETE REST API data using cURL

In the following PHP program, we will delete an employee with id '19465'. If a user exists, it will be deleted and a successful response will be received. You need to make sure to pass an employee id that actually exists.

```
<?php

// User data to send using HTTP POST method in curl
$data = array();

// Data should be passed as json format
$data_json = json_encode($data);

// API URL to send data
$url = 'https://dummy.restapiexample.com/api/v1/delete/2';

// curl intitite
$curl_handle = curl_init();

curl_setopt($curl_handle, CURLOPT_URL, $url);

// Set json header to received json response properly
curl_setopt($curl_handle, CURLOPT_HTTPHEADER, array('Content-Type: application/json','Content-Length: ' . strlen($data_json)));

// SET Method as a DELETE
curl_setopt($curl_handle, CURLOPT_CUSTOMREQUEST, "DELETE");
// Pass user data in POST command
curl_setopt($curl_handle, CURLOPT_POSTFIELDS,$data_json);

curl_setopt($curl_handle, CURLOPT_RETURNTRANSFER, true);

// Execute curl and assign returned data into
$response = curl_exec($curl_handle);

// Close curl
curl_close($curl_handle);

// See response if data is posted successfully or any error
print_r ($response);

?>
```

Code Explanation

First, we have initiated a cURL session. After that, we sent HTTP DELETE command to REST API DELETE URL. Suppose you want to delete an employee having id '19465'. Then, the API URL will be '<http://dummy.restapiexample.com/api/v1/delete/19465>'. If the given employee found, it will be deleted. You will receive some successful response as well.

If you want to verify data after deletion, you can get employee data on the following URL.

API URL to GET Employees data: <https://dummy.restapiexample.com/api/v1/employees>

We have covered simple REST API methods using PHP cURL functions to interact with API data. There are few other methods also available. You can search for more online dummy REST APIs and practise more.

Disable WordPress REST API

```
add_filter( 'json_enabled', '__return_false' ); add_filter(  
'json_jsonp_enabled', '__return_false' );
```

Here's how to get and use REST API endpoints in WordPress

In WordPress, you can access the REST API endpoints by default without any additional configuration. The REST API is enabled by default in recent WordPress versions, and it provides access to various data and functionalities within your WordPress site.

Explore Available Endpoints: You can explore the available REST API endpoints by visiting the following URL in your browser

<http://yourwordpresssite.com/wp-json/>

This will give you a list of available routes and endpoints. You can navigate through this structure to find the specific endpoint you want to use.

Retrieve Data from an Endpoint

To retrieve data from a specific endpoint, you can use the `wp_remote_get()` function in WordPress. Here's an example of how to use it to retrieve data from the REST API.

```
// Define the API endpoint URL
$api_url = 'http://yourwordpresssite.com/wp-json/wp/v2/posts';

// Make the API request
$response = wp_remote_get($api_url);

// Check if the request was successful
if (is_array($response) && !is_wp_error($response)) {
    // Parse the JSON response
    $data = json_decode(wp_remote_retrieve_body($response));
}

// Process and use the data as needed
foreach ($data as $post) {
    echo 'Title: ' . $post->title->rendered . '<br>';
}
```

```
echo 'Content: ' . $post->content->rendered . '<br>';

}

}
```

In the code above, we are making a GET request to the `/wp-json/wp/v2/posts` endpoint to retrieve a list of posts. You can replace this endpoint with the one you need.

Authentication and Permissions: Some REST API endpoints may require authentication or have different permission levels. You may need to provide authentication credentials or set up custom endpoints with specific permissions if necessary.

Customizing and Extending Endpoints: WordPress allows you to customize and extend REST API endpoints to suit your specific needs. You can create custom endpoints using the `register_rest_route()` function in your theme's `functions.php` file or a custom plugin. This allows you to define your own routes and callback functions for handling API requests.

Here's an example of registering a custom REST API endpoint:

```
function custom_api_route() {

    register_rest_route('custom/v1', 'my-endpoint', array(
        'methods' => 'GET',
        'callback' => 'my_custom_callback',
    ));

}

function my_custom_callback($data) {
    // Your custom logic here
    return array('message' => 'Custom API endpoint response');
}

add_action('rest_api_init', 'custom_api_route');
```

In this example, we've registered a custom endpoint at `/wp-json/custom/v1/my-endpoint`. When you make a GET request to this endpoint, it will trigger the `my_custom_callback` function and return a custom response.

To call a REST API in WordPress,

you can use the built-in functions and classes provided by WordPress. Here's a step-by-step guide on how to call a REST API in WordPress:

1. Determine the API Endpoint:

Identify the REST API endpoint you want to call. WordPress provides several REST API endpoints for different resources, such as posts, pages, users, and custom endpoints created by plugins or themes.

2. Use the WP HTTP API:

WordPress provides the WP HTTP API, which is a set of functions and classes for making HTTP requests. The WP HTTP API supports various HTTP methods like GET, POST, PUT, and DELETE.

Here's an example of how to make a GET request to a REST API endpoint using the WP HTTP API:

```
// Make a GET request to the REST API endpoint
$response = wp_remote_get('https://example.com/wp-json/wp/v2/posts');
```

```
// Check if the request was successful
if (is_wp_error($response)) {
    $error_message = $response->get_error_message();
    echo "Error: $error_message";
} else {
    $body = wp_remote_retrieve_body($response);
    $data = json_decode($body, true);
    // Process the response data
}
```

Process the response data

```
}
```

What is localize script in ajax of wordpress

In WordPress, `wp_localize_script` is a function used to make data from the server-side (PHP) available for use in client-side (JavaScript) scripts. It's a powerful tool for passing dynamic data, such as configuration settings or localized strings, from your WordPress server to your JavaScript code, including those used in Ajax requests.

Here's how `wp_localize_script` works in the context of Ajax in WordPress:

Enqueue Your JavaScript File:

First, you enqueue your JavaScript file using the `wp_enqueue_script` function. This ensures that your JavaScript file is loaded on the page where you want to use it.

```
wp_enqueue_script('your-script-handle', 'path-to-your-script.js', array('jquery'), '1.0', true);
```

Use `wp_localize_script`:

Next, you use the `wp_localize_script` function to pass data from the server to your JavaScript file. This function takes the following parameters:

- Script handle: The handle of the JavaScript file you want to localize.
- Object name: The name of the JavaScript object that will hold the localized data.
- Data array: An array of key-value pairs containing the data you want to pass.

```
wp_localize_script('your-script-handle', 'yourLocalizedData', array(  
    'ajax_url' => admin_url('admin-ajax.php'),  
    'nonce' => wp_create_nonce('your-nonce'),  
    // Other data...  
));
```

Access Data in JavaScript:

In your JavaScript file, you can access the localized data through the `yourLocalizedData` object. For example:

```
// Access the Ajax URL  
var ajaxUrl = yourLocalizedData.ajax_url;  
  
// Access the nonce
```

```
var nonce = yourLocalizedData.nonce;
```

Use the Data in Ajax Requests:

When making Ajax requests in your JavaScript code, you can use the `ajaxUrl` and `nonce` variables to specify the Ajax URL and add security checks to your requests. For example:

```
jQuery.ajax({
    type: 'POST',
    url: ajaxUrl,
    data: {
        action: 'your_ajax_action',
        nonce: nonce,
        // Other data...
    },
    success: function(response) {
        // Handle the response
    }
});
```

In this code, '`'your_ajax_action'`' should be replaced with the name of the server-side action that handles your Ajax request.

By localizing scripts using `wp_localize_script`, you can keep your JavaScript code modular and organized while still having access to dynamic server-side data. This is particularly useful when working with Ajax in WordPress, as it allows you to pass essential data, such as security tokens or URLs, seamlessly from the server to the client.

```
// Define a custom shortcode to display the current date
function custom_current_date_shortcode() {
    $current_date = date('Y-m-d');
    return "Today's date is: $current_date";
}
add_shortcode('current_date', 'custom_current_date_shortcode');
```

Load more post by AJAX Call

```
function blog_scripts() {
    // Register the script
    wp_register_script( 'custom-script', get_stylesheet_directory_uri() . '/js/custom.js', array('jquery'), false, true );

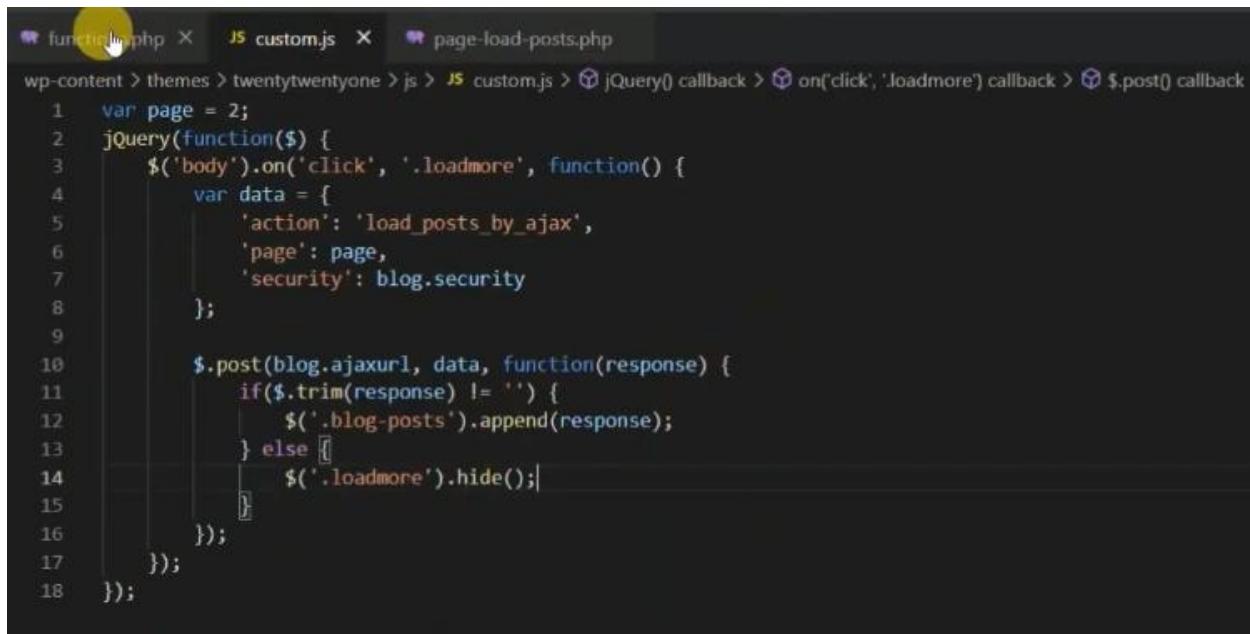
    // Localize the script with new data
    $script_data_array = array(
        'ajaxurl' => admin_url( 'admin-ajax.php' ),
        'security' => wp_create_nonce( 'load_more_posts' ),
    );
    wp_localize_script( 'custom-script', 'blog', $script_data_array );

    // Enqueue script with localized data.
    wp_enqueue_script( 'custom-script' );
}
add_action( 'wp_enqueue_scripts', 'blog_scripts' );

function load_posts_by_ajax_callback() {
    check_ajax_referer('load_more_posts', 'security');
    $args = array(
        'post_type' => 'post',
        'post_status' => 'publish',
        'posts_per_page' => '2',
        'paged' => $_POST['page'],
    );
    $blog_posts = new WP_Query( $args );
    ?>
    <?php if ( $blog_posts->have_posts() ) : ?>
        <?php while ( $blog_posts->have_posts() ) : $blog_posts->the_post(); ?>
            <h2><?php the_title(); ?></h2>
            [?php the_excerpt(); ?]
        <?php endwhile; ?>
    <?php endif; ?>
    <?php
    wp_die();
}

add_action('wp_ajax_load_posts_by_ajax', 'load_posts_by_ajax_callback');
```

In 670, Col 33 Tab Size: 4 UTF-8 LF PHP Go Live



The screenshot shows a code editor with three tabs: `functions.php`, `custom.js`, and `page-load-posts.php`. The `page-load-posts.php` tab is active, displaying the following JavaScript code:

```
1 var page = 2;
2 jQuery(function($) {
3     $('body').on('click', '.loadmore', function() {
4         var data = {
5             'action': 'load_posts_by_ajax',
6             'page': page,
7             'security': blog.security
8         };
9
10        $.post(blog.ajaxurl, data, function(response) {
11            if($.trim(response) != '') {
12                $('.blog-posts').append(response);
13            } else [
14                $('.loadmore').hide();
15            ]
16        });
17    });
18});
```

The code uses jQuery to bind a click event to elements with the class `.loadmore`. When clicked, it sends a POST request to the URL stored in `blog.ajaxurl` with data including the current page number and security nonce. It then appends the response to the element with the class `.blog-posts`. If there is no response, it hides the `.loadmore` button.

```
* functions.php      page-load-posts.php ●
wp-content > themes > twentytwentyone > page-load-posts.php
1  <?php
2   get_header();
3
4  /* Start the Loop */
5  while ( have_posts() ) :
6      the_post();
7      ?>
8      <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
9
10     <header class="entry-header alignwide">
11         <?php the_title( '<h1 class="entry-title">', '</h1>' ); ?>
12     </header>
13     <div class="entry-content">
14         <?php
15             $args = array(
16                 'post_type' => 'post',
17                 'post_status' => 'publish',
18                 'posts_per_page' => '2',
19                 'paged' => 1,
20             );
21
22             $blog_posts = new WP_Query( $args );
23         ?>
24         <?php if ( $blog_posts->have_posts() ) : ?>
25             <div class="blog-posts">
26                 <?php while ( $blog_posts->have_posts() ) : $blog_posts->the_post(); ?>
27                     <h2><?php the_title(); ?></h2>
28                     <?php the_excerpt(); ?>
29                     <?php endwhile; ?>
30                     <?php wp_reset_postdata(); ?>
31             </div>
32             <div class="loadmore">Load More...[</div>]
33         <?php endif; ?>
34     </div><!-- .entry-content -->
35
36     </article><!-- #post-<?php the_ID(); ?> -->
37
38     <?php
```

Reffrence link <https://www.youtube.com/watch?v=2k-hPpHzL1Q&t=28>
<https://artisansweb.net/load-wordpress-post-ajax/>

https://www.youtube.com/watch?v=bBsIHE-IkGo&list=PLaoZUFJYLaOYRd_WSV22NPv7q-ZV4gKWs&index=78

Block development link

AJAX call in WordPress, you can follow these steps:

Reference

<https://developer.wordpress.org/plugins/javascript/ajax/>

1.Enqueue jQuery:

```
function enqueue_scripts() {  
    wp_enqueue_script('jquery');  
}  
add_action('wp_enqueue_scripts', 'enqueue_scripts');
```

2.Create the JavaScript file:

Create a new JavaScript file where you'll write the AJAX code. You can create a separate file or include it in an existing JavaScript file in your theme or plugin.

For example, let's create a file called ajax-script.js:

```
jQuery(document).ready(function($) {  
  
    $.ajax({  
  
        url: ajax_object.ajax_url, // AJAX handler URL provided by WordPress  
        type: 'POST',  
        data: {  
            action: 'my_ajax_action',  
            // The PHP function to handle the AJAX request  
            // Add any additional data you want to send to the server  
        },  
        success: function(response) {  
  
        },  
        error: function(xhr, status, error) {  
            // Handle AJAX error  
        }  
    });  
});
```

3.Register the AJAX handler in PHP:

In your theme or plugin's functions.php file, register a PHP function to handle the AJAX request.

```
function my_ajax_handler() {  
    // Handle the AJAX request  
    // Perform any necessary processing or database operations  
    // Generate the response data  
    // Send the response  
    wp_send_json($response);
```

```
}
```

```
add_action('wp_ajax_my_ajax_action', 'my_ajax_handler');
```

```
add_action('wp_ajax_nopriv_my_ajax_action', 'my_ajax_handler'); // For non-logged-in users
```

Make sure to replace 'my_ajax_action' with the name you provided in the AJAX call (step 2).

4. Localize the JavaScript file:

In your theme or plugin's functions.php file, use the wp_localize_script() function to pass the AJAX handler URL to the JavaScript file. Add the following code:

```
function localize_scripts()
```

```
{
```

```
    wp_enqueue_script('ajax-script', get_template_directory_uri() . '/path/to/ajax-script.js',
```

```
array('jquery'), '1.0', true);
```



```
    wp_localize_script('ajax-script', 'ajax_object', array('ajax_url' => admin_url('admin-ajax.php')));
```

```
}
```



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

Make sure to replace '/path/to/ajax-script.js' with the correct path to your JavaScript file.

That's it! You now have an AJAX call set up in WordPress. When the JavaScript code is executed, it will send an AJAX request to the server, which will be handled by the PHP function specified in the AJAX action. You can perform any necessary processing, database operations, and generate a response that will be sent back to the JavaScript code for further handling.

Wordpress Ajax Call

```
add_action('wp_ajax_my_action','data_fetch');
add_action('wp_ajax_nopriv_my_action','data_fetch');
function data_fetch(){
    $the_query=new WP_Query(array('posts_per_page'=>10));
    if($the_query->have_posts()):
        while($the_query->have_posts()): $the_query->the_post(); ?>
            <h2><?php the_title(); ?></h2>
            <p><?php the_content(); ?></p>
        <?php endwhile;
        wp_reset_postdata();
        endif;
        die();
}
```

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <script type="text/javascript" src="/wordpress/js/jquery-1.11.2.min.js"></script>
5          <script>
6              function fetch(){
7                  $.post('/wordpress/wp-admin/admin-ajax.php', {'action':'my_action'}, function(response){
8                      $('#datafetch').append(response);
9                  });
10                 }
11             </script>
12         </head>
13         <body>
14             <div id="datafetch">
15                 <button onclick="fetch()">Click Here</button>
16             </div>
17         </body>
18     </html>
```

WP Ajax call Vissal programming

```
<script>
jQuery('#frmContactUs').submit(function(){
    event.preventDefault();
    var link=<?php echo admin_url('admin-ajax.php')?>";
    var form=jQuery('#frmContactUs').serialize();
    var formData=new FormData;
    formData.append('action','contact_us');
    formData.append('contact_us',form);
    jQuery.ajax({
        url:link
    });
    jQuery.ajax({
        url:link,
        data:formData,
        processData:false,
        contentType:false,
        type:'post',
        success:function(result){
            if(result.success==true){
                //
            }
            jQuery('#result_msg').html('<span class="'+result.s
                //result.success
                //result.data
            }
        });
    });
});
```

```
add_action('wp_ajax_contact_us','ajax_contact_us');
function ajax_contact_us(){
    $arr=[ ];
    wp_parse_str($_POST['contact_us'],$arr);
    global $wpdb;
    global $table_prefix;
    $table=$table_prefix.'contact_us';
    $result=$wpdb->insert($table,[
        "name"=>$arr['name'],
        "email"=>$arr['email']
    ]);

    if($result>0){
        wp_send_json_success("Data inserted");
    }else{
        wp_send_json_error("Please try again");
    }
}
```

Load more post by AJAX Call

```
function blog_scripts() {
    // Register the script
    wp_register_script( 'custom-script', get_stylesheet_directory_uri() . '/js/custom.js', array('jquery'), false, true );

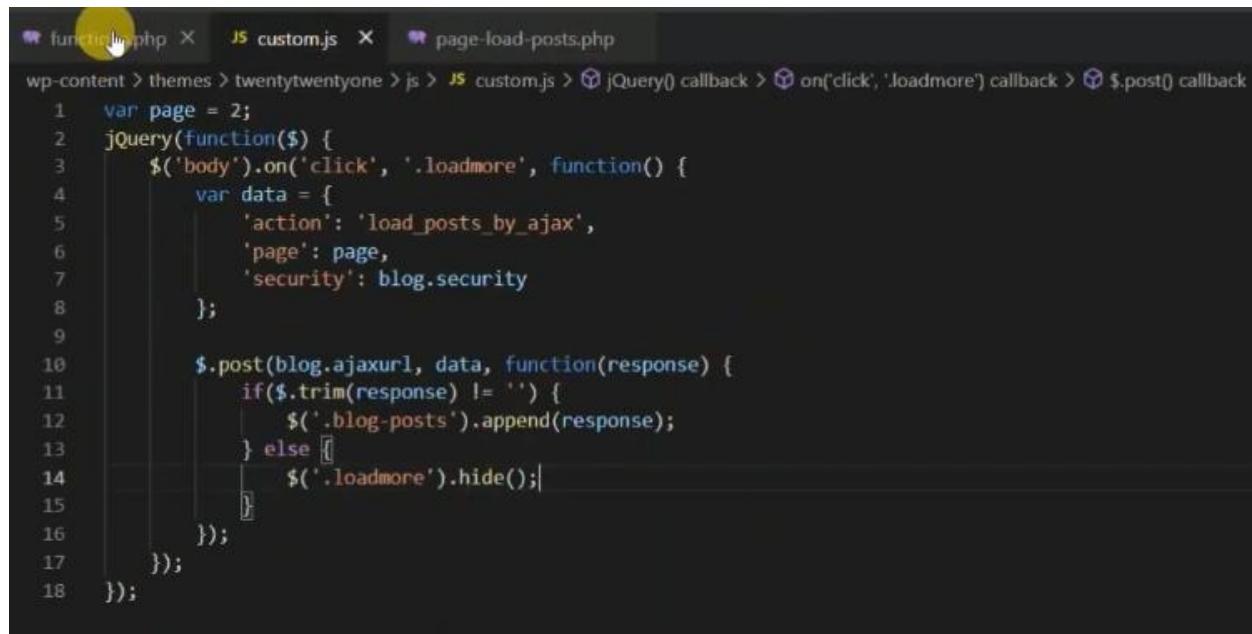
    // Localize the script with new data
    $script_data_array = array(
        'ajaxurl' => admin_url( 'admin-ajax.php' ),
        'security' => wp_create_nonce( 'load_more_posts' ),
    );
    wp_localize_script( 'custom-script', 'blog', $script_data_array );

    // Enqueue script with localized data.
    wp_enqueue_script( 'custom-script' );
}
add_action( 'wp_enqueue_scripts', 'blog_scripts' );

function load_posts_by_ajax_callback() {
    check_ajax_referer('load_more_posts', 'security');
    $args = array(
        'post_type' => 'post',
        'post_status' => 'publish',
        'posts_per_page' => '2',
        'paged' => $_POST['page'],
    );
    $blog_posts = new WP_Query( $args );
    ?>
    <?php if ( $blog_posts->have_posts() ) : ?>
        <?php while ( $blog_posts->have_posts() ) : $blog_posts->the_post(); ?>
            <h2><?php the_title(); ?></h2>
            [?php the_excerpt(); ?]
        <?php endwhile; ?>
    <?php endif; ?>
    <?php
    wp_die();
}

add_action('wp_ajax_load_posts_by_ajax', 'load_posts_by_ajax_callback');
```

In 670, Col 33 Tab Size: 4 UTF-8 LF PHP Go Live



The screenshot shows a code editor with three tabs: `functions.php`, `custom.js`, and `page-load-posts.php`. The `page-load-posts.php` tab is active, displaying the following JavaScript code:

```
1 var page = 2;
2 jQuery(function($) {
3     $('body').on('click', '.loadmore', function() {
4         var data = {
5             'action': 'load_posts_by_ajax',
6             'page': page,
7             'security': blog.security
8         };
9
10        $.post(blog.ajaxurl, data, function(response) {
11            if($.trim(response) != '') {
12                $('.blog-posts').append(response);
13            } else [
14                $('.loadmore').hide();
15            ]
16        });
17    });
18});
```

The code uses jQuery to bind a click event to elements with the class `.loadmore`. When clicked, it sends a POST request to the `blog.ajaxurl` endpoint with a JSON object containing `'action': 'load_posts_by_ajax'`, `'page'` (set to `page`), and `'security'` (set to `blog.security`). The response is appended to the element with the class `.blog-posts`. If the response is empty, the `.loadmore` button is hidden.

```
* functions.php      page-load-posts.php ●
wp-content > themes > twentytwentyone > page-load-posts.php
1  <?php
2   get_header();
3
4  /* Start the Loop */
5  while ( have_posts() ) :
6      the_post();
7      ?>
8      <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
9
10     <header class="entry-header alignwide">
11         <?php the_title( '<h1 class="entry-title">', '</h1>' ); ?>
12     </header>
13     <div class="entry-content">
14         <?php
15             $args = array(
16                 'post_type' => 'post',
17                 'post_status' => 'publish',
18                 'posts_per_page' => '2',
19                 'paged' => 1,
20             );
21
22             $blog_posts = new WP_Query( $args );
23         ?>
24         <?php if ( $blog_posts->have_posts() ) : ?>
25             <div class="blog-posts">
26                 <?php while ( $blog_posts->have_posts() ) : $blog_posts->the_post(); ?>
27                     <h2><?php the_title(); ?></h2>
28                     <?php the_excerpt(); ?>
29                     <?php endwhile; ?>
30                     <?php wp_reset_postdata(); ?>
31             </div>
32             <div class="loadmore">Load More...[</div>]
33         <?php endif; ?>
34     </div><!-- .entry-content -->
35
36     </article><!-- #post-<?php the_ID(); ?> -->
37
38     <?php
```

Reffrence link <https://www.youtube.com/watch?v=2k-hPpHzL1Q&t=28>
<https://artisansweb.net/load-wordpress-post-ajax/>

https://www.youtube.com/watch?v=bBsIHE-IkGo&list=PLaoZUFJYLaOYRd_WSV22NPv7q-ZV4gKWs&index=78

Ajax call

<https://developer.wordpress.org/plugins/javascript/ajax/>
[Client Side Summary](#)

Now that we've added our callback as the final parameter for the `$.post()` function, we've completed our sample jQuery Ajax script. All the pieces put together look like this:

```
jQuery(document).ready(function($) {          //wrapper
    $(".pref").change(function() {           //event
        var this2 = this;                  //use in callback
        $.post(my_ajax_obj.ajax_url, {      //POST request
            _ajax_nonce: my_ajax_obj.nonce, //nonce
            action: "my_tag_count",       //action
            title: this.value           //data
        }, function(data) {               //callback
            this2.nextSibling.remove(); //remove current title
            $(this2).after(data);       //insert server response
        }
    });
});
```



```
$title_nonce = wp_create_nonce( 'title_example' );
wp_localize_script(
    'ajax-script',
    'my_ajax_obj',
    array(
        'ajax_url' => admin_url( 'admin-ajax.php' ),
        'nonce'     => $title_nonce,
    )
);
```

Sample code

```
add_action( 'admin_enqueue_scripts', 'my_enqueue' );

/**
 * Enqueue my scripts and assets.
 *
 * @param $hook
 */
function my_enqueue( $hook ) {
    if ( 'myplugin_settings.php' !== $hook ) {
        return;
    }
    wp_enqueue_script(
        'ajax-script',
        plugins_url( '/js/myjquery.js', __FILE__ ),
        array( 'jquery' ),
        '1.0.0',
        true
    );
    wp_localize_script(
        'ajax-script',
        'my_ajax_obj',
        array(
            'ajax_url' => admin_url( 'admin-ajax.php' ),
            'nonce'     => wp_create_nonce( 'title_example' ),
        )
    );
}
```

Register vs. Enqueue

You will see examples in other tutorials that religiously use [wp_register_script\(\)](#). This is fine, but its use is optional. What is not optional is [wp_enqueue_script\(\)](#). This function must be called in order for your script file to be properly linked on the web page. So why register scripts? It creates a useful tag or handle with which you can easily reference the script in various parts of your code as needed. If you just need your script loaded and are not referencing it elsewhere in your code, there is no need to register it.

AJAX Handler Summary

<https://developer.wordpress.org/plugins/javascript/enqueuing/>

```
/**  
 * AJAX handler using JSON  
 */  
function my_ajax_handler_json() {  
    check_ajax_referer( 'title_example' );  
    $title = wp_unslash( $_POST['title'] );  
  
    update_user_meta( get_current_user_id(), 'title_preference',  
sanitize_post_title( $title ) );  
  
    $args      = array(  
        'tag' => $title,  
    );  
    $the_query = new WP_Query( $args );  
    wp_send_json( esc_html( $title ) . ' (' . $the_query->post_count .  
')' );  
}
```

Heartbeat API

How it works

When the page loads, the client-side heartbeat code sets up an interval (called the “tick”) to run every 15-120 seconds. When it runs, heartbeat gathers data to send via a jQuery event, then sends this to the server and waits for a response. On the server, an admin-ajax handler takes the passed data, prepares a response, filters the response, then returns the data in JSON format. The client receives this data and fires a final jQuery event to indicate the data has been received.

The basic process for custom Heartbeat events is:

1. Add additional fields to the data to be sent (JS **heartbeat-send** event)
2. Detect sent fields in PHP, and add additional response fields (**heartbeat_received** filter)
3. Process returned data in JS (JS **heartbeat-tick**)

(You can choose to use only one or two of these events, depending on what functionality you need.)

Using the API

Using the heartbeat API requires two separate pieces of functionality: send and receive callbacks in JavaScript, and a server-side filter to process passed data in PHP.

Sending Data to the Server

When Heartbeat sends data to the server, you can include custom data. This can be any data you want to send to the server, or a simple true value to indicate you are expecting data.

```
jQuery( document ).on( 'heartbeat-send', function ( event, data ) {  
    // Add additional data to Heartbeat data.  
    data.myplugin_customfield = 'some_data';  
});
```

Receiving and Responding on the Server

On the server side, you can then detect this data, and add additional data to the response.

```
/*
 * Receive Heartbeat data and respond.
 *
 * Processes data received via a Heartbeat request, and returns additional data
to pass back to the front end.
 *
 * @param array $response Heartbeat response data to pass back to front end.
 * @param array $data      Data received from the front end (unslashed).
 *
 * @return array
 */
function myplugin_receive_heartbeat( array $response, array $data ) {
    // If we didn't receive our data, don't send any back.
    if ( empty( $data['myplugin_customfield'] ) ) {
        return $response;
    }

    // Calculate our data and pass it back. For this example, we'll hash it.
    $received_data = $data['myplugin_customfield'];

    $response['myplugin_customfield_hashed'] = sha1( $received_data );
    return $response;
}
add_filter( 'heartbeat_received', 'myplugin_receive_heartbeat', 10, 2 );
```

Processing the Response

Back on the frontend, you can then handle receiving this data back.

```
jQuery( document ).on( 'heartbeat-tick', function ( event, data ) {
    // Check for our data, and use it.
    if ( ! data.myplugin_customfield_hashed ) {
        return;
    }
    alert( 'The hash is ' + data.myplugin_customfield_hashed );
});
```

Ajax Calling Method

Method :1

```
<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>

<script>

$(document).ready(function(){

    $("button").click(function(){

        $("#div1").load("demo_test.txt");

    });

});

</script>

</head>

<body>

    <div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>

    <button>Get External Content</button>

</body>
```

Let jQuery AJAX Change This Text

After clicking result

jQuery and AJAX is FUN!

This is some text in a paragraph.

Method :2

```
<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>

<script>

$(document).ready(function(){

    $("button").click(function(){

        $.ajax({url: "demo_test.txt", success: function(result){

            $("#div1").html(result);

        }});

    });

});

</script>

</head>

<body>

<div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>

<button>Get External Content</button>

</body>
```

Let jQuery AJAX Change This Text

After clicking result

jQuery and AJAX is FUN!

This is some text in a paragraph.

Method :3

```
<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>

<script>

$(document).ready(function(){

    $("button").click(function(){

        $.get("demo_test.asp", function(data, status){

            alert("Data: " + data + "\nStatus: " + status);

        });

    });

    </script>

</head>

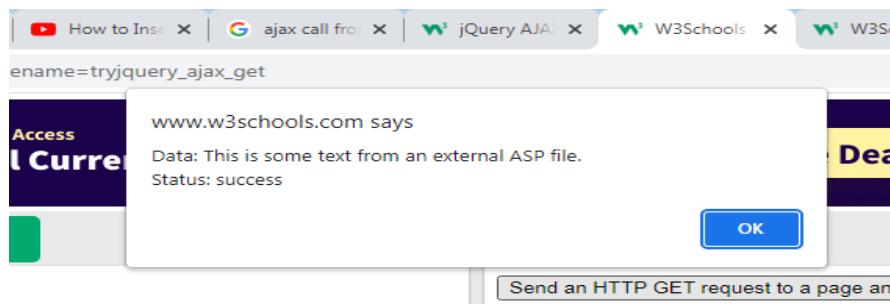
<body>

    <button>Send an HTTP GET request to a page and get the result back</button>

</body>
```

Send an HTTP GET request to a page and get the result back

Result

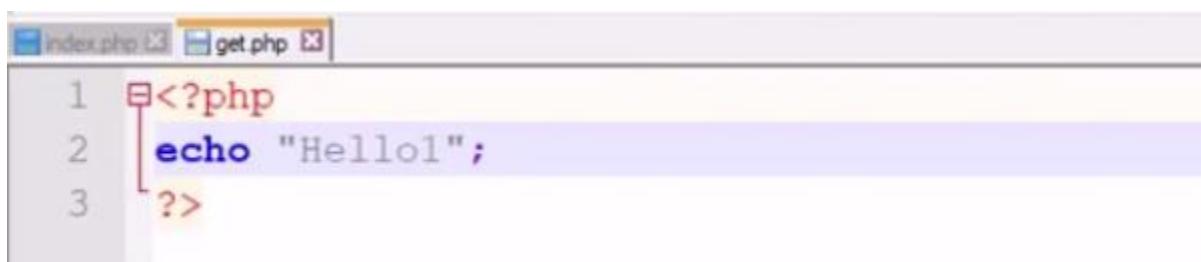


Method	Description
<code>\$.ajax()</code>	Performs an async AJAX request
<code>\$.ajaxPrefilter()</code>	Handle custom Ajax options or modify existing options before each request is sent and before they are processed by <code>\$.ajax()</code>
<code>\$.ajaxSetup()</code>	Sets the default values for future AJAX requests
<code>\$.ajaxTransport()</code>	Creates an object that handles the actual transmission of Ajax data
<code>\$.get()</code>	Loads data from a server using an AJAX HTTP GET request
<code>\$.getJSON()</code>	Loads JSON-encoded data from a server using a HTTP GET request
<code>\$.parseJSON()</code>	Deprecated in version 3.0, use <code>JSON.parse()</code> instead. Takes a well-formed JSON string and returns the resulting JavaScript value
<code>\$.getScript()</code>	Loads (and executes) a JavaScript from a server using an AJAX HTTP GET request
<code>\$.param()</code>	Creates a serialized representation of an array or object (can be used as URL query string for AJAX requests)
<code>\$.post()</code>	Loads data from a server using an AJAX HTTP POST request
<code>ajaxComplete()</code>	Specifies a function to run when the AJAX request completes
<code>ajaxError()</code>	Specifies a function to run when the AJAX request completes with an error
<code>ajaxSend()</code>	Specifies a function to run before the AJAX request is sent
<code>ajaxStart()</code>	Specifies a function to run when the first AJAX request begins
<code>ajaxStop()</code>	Specifies a function to run when all AJAX requests have completed
<code>ajaxSuccess()</code>	Specifies a function to run when an AJAX request completes successfully
<code>load()</code>	Loads data from a server and puts the returned data into the selected element
<code>serialize()</code>	Encodes a set of form elements as a string for submission
<code>serializeArray()</code>	Encodes a set of form elements as an array of names and values

Ajax with PHP

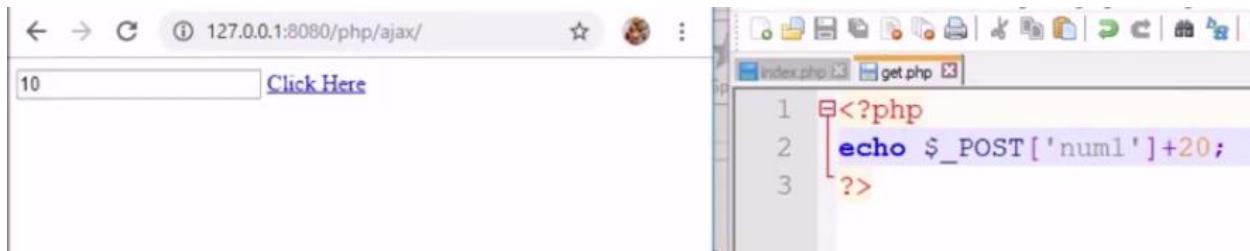


```
index.php get.php
1 <script src="https://code.jquery.com/jquery-2.2.4.min.js"
2   ></script>
3
4   <a href="javascript:void(0)" onclick="click_here()">Click Here
5   </a>
6
7   <script>
8     function click_here() {
9       jQuery.ajax({
10         url: 'get.php',
11         type: 'post',
12         success: function(result) {
13           alert(result);
14         }
15       });
16     </script>
```



```
index.php get.php
1 <?php
2 echo "Hello1";
3 ?>
```

```
index.php get.php
1 <script src="https://code.jquery.com/jquery-2.2.4.min.js"
2 ></script>
3 <input type="textbox" id="num1"/>
4 <a href="javascript:void(0)" onclick="click_here()">Click Here
5 I
6 <script>
7 function click_here(){
8     var num1=jQuery('#num1').val();
9     jQuery.ajax({
10         url:'get.php',
11         type:'post',
12         data:'num1='+num1,
13         success:function(result){
14             alert(result);
15         }
16     });
17 </script>
```



The screenshot shows a browser window with the URL 127.0.0.1:8080/php/ajax/. The page content is "Click Here". Below the browser is a code editor window showing two files: index.php and get.php.

index.php:

```
10 Click Here
```

get.php:

```
1 <?php
2 echo $_POST['num1']+20;
3 ?>
```

Synchronous AJAX - Parallel, it is not waiting for other

Asynchronous AJAX - it is depending on other, Its work one after one.

```
index.php x get.php x types_ajax.php x get1.php x get2.php x
1 <script src="https://code.jquery.com/jquery-2.2.4.min.js"
2   ></script>
3   <script>
4     hit1();
5     hit2();
6     function hit1(){
7       jQuery.ajax({
8         url:'get1.php',
9         type:'post',
10        async:false,
11        success:function(result){
12          console.log(result);
13        }
14      });
15     function hit2(){
16       jQuery.ajax({
17         url:'get2.php',
18         type:'post',
19         success:function(result){
20           console.log(result);
21         }
22     });
23   }
24 
```

```
index.php x get.php x types_ajax.php x get1.php x get2.php x
1 <?php
2 sleep(5);
3 echo 'get1';
4 ??
```

```
index.php x get.php x types_ajax.php x get1.php x get2.php x
1 <?php
2 echo 'get2';
3 ??
```

AJAX With JSON Dataserve API call

```
index.php style.css
8   <link rel="stylesheet" href="css/style.css">
9 </head>
10 <body>
11   <div id="main">
12     <div id="header">
13       <h1>Read JSON Data</h1>
14     </div>
15
16     <div id="load-data"></div>
17
18   </div>
19
20 <script type="text/javascript" src="js/jquery.js"></script>
21 <script>
22   $(document).ready(function(){
23     $.ajax({
24       url : "https://jsonplaceholder.typicode.com/posts/50",
25       type : "GET",
26       success : function(data){
27         $("#load-data").append(data.id + "<br>" + data.title + "<br>" + data.body);
28         //console.log(data);
29       }
30     });
31   });
32 </script>
33 </body>
34 </html>
```

For looping data

```
index.php style.css
12   <div id="header">
13     <h1>Read JSON Data</h1>
14   </div>
15
16   <div id="load-data"></div>
17
18 </div>
19
20 <script type="text/javascript" src="js/jquery.js"></script>
21 <script>
22   $(document).ready(function(){
23     $.ajax({
24       url : "https://jsonplaceholder.typicode.com/posts",
25       type : "GET",
26       success : function(data){
27         //$("#load-data").append(data.id + "<br>" + data.title + "<br>" + data.body);
28         $.each(data, function(key,value){
29           $("#load-data").append(value.id + " " + value.title + "<br>");
30         });
31       }
32     });
33   });
34 </script>
35 </body>
36 </html>
```

API call from Json file

```
index.php      x   my.json      x   style.css      x
<div id="header">
    <h1>Read JSON Data</h1>
</div>

<div id="load-data"></div>

</div>

<script type="text/javascript" src="js/jquery.js"></script>
<script>
$(document).ready(function(){
    $.ajax({
        url : "json/my.json",
        type : "GET",
        success : function(data){
            //$("#load-data").append(data.id + "<br>" + data.title + "<br>" + data.b
            $.each(data, function(key,value){
                $("#load-data").append(value.id + " " + value.title + "<br>");
            });
        }
    });
});
</script>
</body>
```

Sorcut method for AJAX call for Only JSON Data



jQuery ShortCut Function to read JSON Data

```
$.getJSON({
    "JSON URL",
    function(data){
        }
    });
}
```

```
index.php          my.json           style.css
<div id="header">
    <h1>Read JSON Data</h1>
</div>

<div id="load-data"></div>

</div>

<script type="text/javascript" src="js/jquery.js"></script>
<script>
$(document).ready(function(){
    $.getJSON(
        "json/my.json",
        function(data){
            //$("#load-data").append(data.id + "<br>" + data.title + "<br>" + data.body);
            $.each(data, function(key,value){
                $("#load-data").append(value.id + " " + value.title + "<br>");
            });
        }
    );
}</script>
</body>
```

Ajax call

```
index.php script.js submit.php
1 <script src = "http://code.jquery.com/jquery-1.11.1.min.js"></script>
2 <script src = "script.js"></script>
3
4 <div>
5   <input type = "text" id = "fname">
6   <input type = "text" id = "surname">
7   <button id = "formsubmit">Send Data</button><br>
8   <textarea id = "response" style = "width: 200px; height: 100px; resize: none;"></textarea>
9 </div>
```

```
index.php script.js submit.php
1 $(document).ready(function(){
2
3   $('#formsubmit').click(function(){
4
5     $.post("submit.php",
6           {fname: $('#fname').val(), surname: $('#surname').val()},
7           function(data){
8             $('#response').html(data);
9           }
10      );
11
12    });
13
14 });
});
```

```
index.php script.js submit.php localhost/packetcode/
1 <?php
2
3 $fname = $_POST['fname'];
4 $lname = $_POST['surname'];
5
6 echo "$fname $lname";
7
8 ?>
```

The browser window shows the URL `localhost/packetcode/`. In the form, the first input field has the value "Sri" and the second input field has the value "vathsan". The "Send Data" button is being clicked. The response area below the form displays the output "Sri vathsan".

```
index.html x
8
9 <script type="text/javascript" src="js/jquery.js"></script>
10 <script type="text/javascript">
11     $.ajax({
12         url : "https://api.covid19api.com/summary",
13         type : "GET",
14         dataType : "JSON",
15         success : function(data){
16             console.log(data);
17             console.log(data.Global);
18
19             $.each(data.Global, function(key, value){
20                 console.log(key + " : " + value);
21             });
22         }
23     });
24 </script>
25 </body>
26 </html>
```

```
D:\wamp\www\php\html5_validation_withajax\index.html - Notepad++
File Edit Search View Encoding Language Settings Macro Run Window ?
index.html submit.php do.php
123     </div>
124     <script>
125         jQuery('#contactForm').on('submit',function(e){
126             jQuery('#btn').val('Please wait...');
127             jQuery('#btn').attr('disabled',true);
128             jQuery.ajax({
129                 url:'submit.php',
130                 type:'post',
131                 data:jQuery('#contactForm').serialize(),
132                 success:function(result){
133                     jQuery('#thank_you_msg').html('Thank you');
134                     jQuery('#contactForm')[0].reset();
135                     jQuery('#btn').val('Submit Now');
136                     jQuery('#btn').attr('disabled',false);
137                 }
138             });
139             e.preventDefault();
140         });
141     </script>
142 </body>
143 </html>
```

Hyper Text Markup Language file length: 3877 lines: 143 Ln: 133 Col: 61 Sel: 0|0 Dos\Windows ANSI as UTF-8 INS

Type here to search

R ENG 1:16 AM 5/27/2019

The screenshot shows a code editor interface with two tabs open: `index.html` and `index.php`. Below the tabs is a sidebar labeled "FOLDERS" containing a single folder named `json-data`, which contains subfolders `css`, `js`, and `json`, and a file `my.json`.

index.html:

```
37 <script type="text/javascript" src="js/jquery.js"></script>
38 <script type="text/javascript">
39   $.ajax({
40     url : "https://api.covid19api.com/summary",
41     type : "GET",
42     dataType : "JSON",
43     success : function(data){
44       console.log(data);
45
46       console.log(data.Countries[101]);
47       console.log(data.Countries[101].TotalConfirmed);
48
49       $.each(data.Global, function(key, value){
50         $("#global-wise").append("<tr><td bgcolor='yellow'" + key + "</td><td>" + value +
51       });
52
53       var sno = 1;
54       $.each(data.Countries, function(key, value){
55         $("#country-wise").append("<tr>" +
56           "<td>" + sno + "</td>" +
57           "<td>" + value.Country + "</td>" +
58           "<td>" + value.NewConfirmed + "</td>" +
59           "<td>" + value.NewDeaths + "</td>" +
60           "<td>" + value.NewRecovered + "</td>" +
61           "<td>" + value.TotalConfirmed + "</td>" +
```

index.php:

```
12 <div id="header">
13   <h1>Read JSON Data</h1>
14 </div>
15
16 <div id="load-data"></div>
17
18 </div>
19
20 <script type="text/javascript" src="js/jquery.js"></script>
21 <script>
22   $(document).ready(function(){
23     $.ajax({
24       url : "json/my.json",
25       type : "GET",
26       success : function(data){
27         //$("#load-data").append(data.id + "<br>" + data.title + "<br>" + data.b
28         $.each(data, function(key,value){
29           $("#load-data").append(value.id + " " + value.title + "<br>");
30         });
31
32       }
33     });
34   });
35 </script>
36 </body>
```

The screenshot shows the Notepad++ interface with two open files:

- index.php** (Top Window):

```
12     <div id="header">
13         <h1>Read JSON Data</h1>
14     </div>
15
16     <div id="load-data"></div>
17
18     </div>
19
20 <script type="text/javascript" src="js/jquery.js"></script>
21 <script>
22     $(document).ready(function(){
23         $.getJSON(
24             "json/my.json",
25             function(data){
26                 //$("#load-data").append(data.id + "<br>" + data.title + "<br>" + data.body);
27                 $.each(data, function(key,value){
28                     $("#load-data").append(value.id + " " + value.title + "<br>");
29                 });
30
31             }
32         );
33     });
34 </script>
35 </body>
```
- submit.php** (Bottom Window):

```
1 <script src = "http://code.jquery.com/jquery-1.11.1.min.js"></script>
2 <script src = "script.js"></script>
3
4 <div>
5     <input type = "text" id = "fname">
6     <input type = "text" id = "surname">
7     <button id = "formsubmit">Send Data</button><br>
8     <textarea id = "response" style = "width: 200px; height: 100px; resize:
9         none;"></textarea>
</div>
```

The Explorer panel on the left shows the directory structure under E:\Programs\XAMPP\htdocs\packetcode. The status bar at the bottom indicates the file length (343), lines (9), and current position (Ln:2 Col:35 Sel:0|0). The bottom right corner shows the date and time (6/1/2014 2:49 PM).

E:\Programs\XAMPP\htdocs\packetcode\submit.php - Notepad++

```

<?php
$fname = $_POST['fname'];
$lname = $_POST['surname'];

echo "$fname $lname";
?>

```

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

Explorer - E:\Programs\XAMPP\htdocs\packet...

Name Ext.

- [..]
- index.php
- script.js
- submit.php

Filter:

PHP Hypertext Preproc Mozilla Firefox

E:\Programs\XAMPP\htdocs\packetcode\script.js - Notepad++

```

$(document).ready(function(){
    $('#formsubmit').click(function(){
        $.post("submit.php",
            {fname: $('#fname').val(), surname: $('#surname').val()},
            function(data){
                $('#response').html(data);
            }
        );
    });
});

```

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

Explorer - E:\Programs\XAMPP\htdocs\packet...

Name Ext.

- [..]
- index.php
- script.js
- submit.php

Filter:

JavaScript file

NppFTP - Disconnected

Action Progress File

length: 94 lines: 8 Ln:6 Col:21 Sel:0|0 Dos\Windows ANSI INS 2:50 PM 6/1/2014

Action Progress File

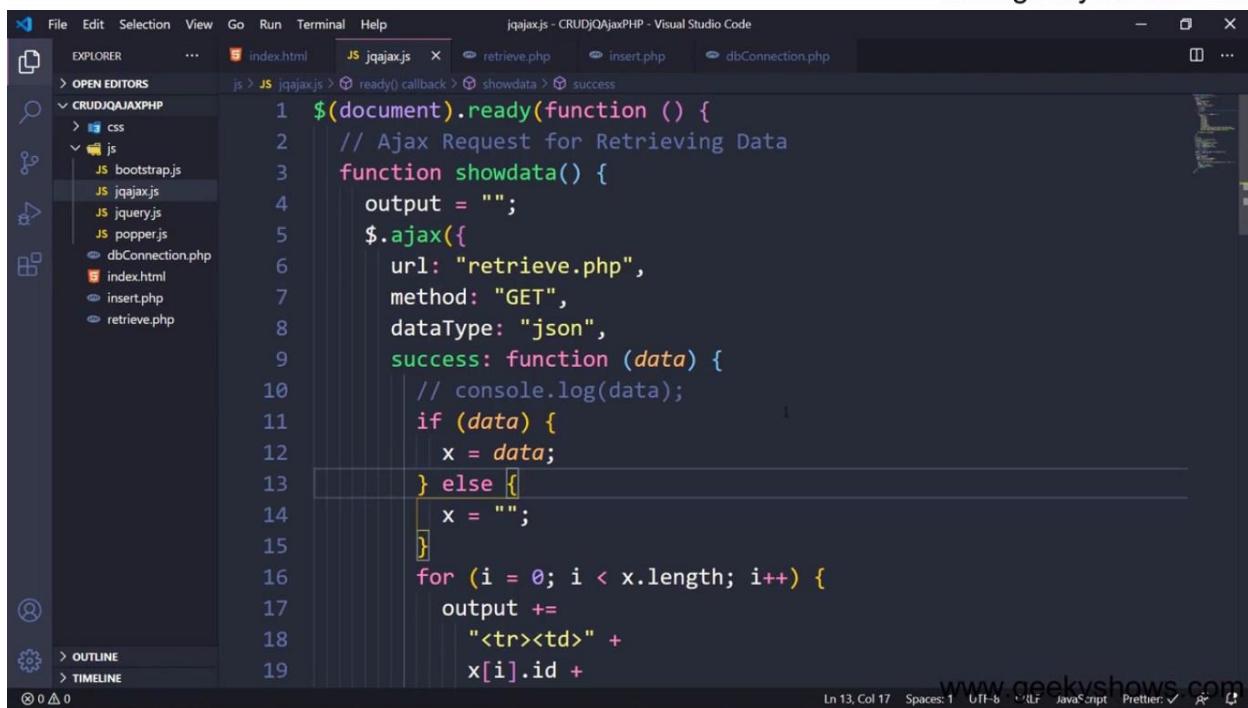
length: 237 lines: 13 Ln:8 Col:14 Sel:0|0 Dos\Windows ANSI INS 2:50 PM 6/1/2014

jQuery Ajax

Example:-

```
$(document).ready(function(){
    $("#btn").click(function(){
        $.ajax({
            url: "insert.php",
            method: "POST",
            data: {"name": "sonam", "roll": 101},
            success: function(data){
                // Process the Response Data
            }
        })
    })
})
```

www.geekyshows.com



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor Tab:** index.html, JS jqajax.js (active), retrieve.php, insert.php, dbConnection.php.
- Explorer:** Shows the project structure under CRUDJQAJAXPHP, including files like bootstrap.js, jquery.js, popper.js, dbConnection.php, index.html, insert.php, and retrieve.php.
- Code Editor:** The jqajax.js file contains the following JavaScript code:

```
1 $(document).ready(function () {
2     // Ajax Request for Retrieving Data
3     function showdata() {
4         output = "";
5         $.ajax({
6             url: "retrieve.php",
7             method: "GET",
8             dataType: "json",
9             success: function (data) {
10                 // console.log(data);
11                 if (data) {
12                     x = data;
13                 } else {
14                     x = "";
15                 }
16                 for (i = 0; i < x.length; i++) {
17                     output +=
18                         "<tr><td>" +
19                         x[i].id +
```

Ln 13, Col 17 Spaces:1 Diff:0 JavaScript Prettier:✓

The screenshot shows two instances of Visual Studio Code side-by-side, both displaying code related to a CRUD application using Ajax and PHP.

Top Window (jqajax.js):

```
15
16     for (i = 0; i < x.length; i++) {
17         output +=
18             "<tr><td>" +
19             x[i].id +
20             "</td><td>" +
21             x[i].name +
22             "</td><td>" +
23             x[i].email +
24             "</td><td>" +
25             x[i].password +
26             "</td><td> <button class='btn btn-warning btn-sm
27             btn-edit'>Edit</button> <button class='btn btn-danger
28             btn-sm btn-edit'>Delete</button></td></tr>";
29     }
30     $("#tbody").html(output);
31 },
```

Bottom Window (retrieve.php):

```
1  <?php
2  include('dbConnection.php');
3
4 // Retrieve Student Information
5 $sql = "SELECT * FROM student";
6 $result = $conn->query($sql);
7 if($result->num_rows > 0){
8     $data = array();
9     while($row = $result->fetch_assoc()){
10        $data[] = $row;
11    }
12 }
13
14 // Returning JSON Format Data as Response to Ajax Call
15 echo json_encode($data)
16
17
18 ?>
```

index.php insert-data.php

```
44 <script type="text/javascript" src="js/jquery.js"></script>
45 <script type="text/javascript">
46     $(document).ready(function(){
47         $('#student-form').submit(function(e){
48             e.preventDefault();
49
50             var fname = $('#first-name').val();
51             var languages = [];
52
53             $(".lang").each(function(){
54                 if($(this).is(":checked")){
55                     languages.push($(this).val());
56                 }
57             });
58
59             languages = languages.toString();
60
61             if(fname != '' && languages.length !== 0){
62                 $.ajax({
63                     url : "insert-data.php",
64                     method : "POST",
65                     data : {name : fname, languages: languages},
66                     success : function(data){
```

Multiple Checkboxed Handler

Document 127.0.0.1:5500/data.html

```
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width,
8   <title>Document</title>
9   <style>
10    button[data-person="hari"] {
11        background-color: yellowgreen;
12    }
13   </style>
14   </head>
15
16   <body>
17     <button data-person="kishan">kishan here</button>
18     <button data-person="hari">hari here</button>
19   </body>
20
21   </html>
```

Type here to search

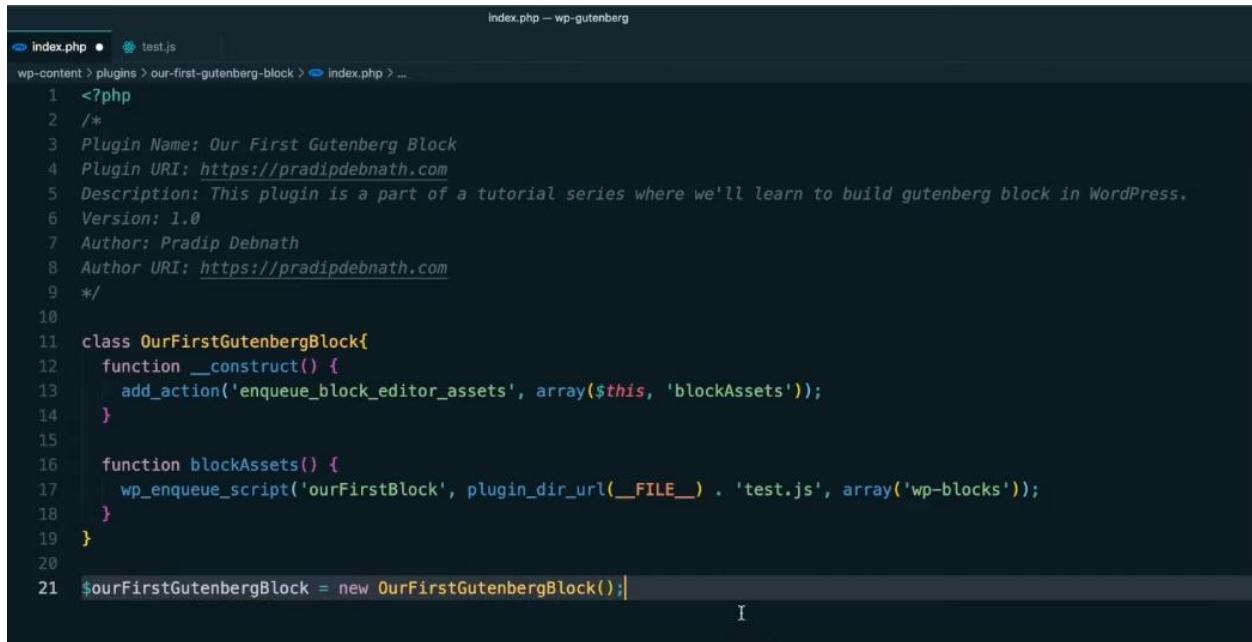
Ln 10: Col 33 Spaces: 4 UTF-8 CRLF HTML Port: 5500 Prettier 11:15 PM ENG 01-05-2021

Basic Gutenberg Block Development



```
test.js — wp-gutenberg
index.php  test.js  X
wp-content > plugins > our-first-gutenberg-block > test.js > edit
1 wp.blocks.registerBlockType('ourblockplugin/our-first-gutenberg-block', {
2   title: 'Our First Gutenberg Block',
3   icon: 'admin-plugins',
4   category: 'text',
5   edit: function() {
6     return wp.element.createElement('h3', null, 'Hello, this is from the admin editor screen.');
7   },
8   save: function() {
9     return wp.element.createElement('h1', null, 'Hello, this is frontend.');
10 }
11 });


```



```
index.php — wp-gutenberg
index.php  test.js
wp-content > plugins > our-first-gutenberg-block > index.php > ...
1 <?php
2 /**
3  Plugin Name: Our First Gutenberg Block
4  Plugin URI: https://pradipdebnath.com
5  Description: This plugin is a part of a tutorial series where we'll learn to build gutenberg block in WordPress.
6  Version: 1.0
7  Author: Pradip Debnath
8  Author URI: https://pradipdebnath.com
9 */
10
11 class OurFirstGutenbergBlock{
12   function __construct() {
13     add_action('enqueue_block_editor_assets', array($this, 'blockAssets'));
14   }
15
16   function blockAssets() {
17     wp_enqueue_script('ourFirstBlock', plugin_dir_url(__FILE__) . 'test.js', array('wp-blocks'));
18   }
19 }
20
21 $ourFirstGutenbergBlock = new OurFirstGutenbergBlock();
```

Richtext Block

```
js index.js .../basic-block    js index.js .../rich-text-block x  editor.css .../rich-text-block
1 import './editor.css';
2
3 const { RichText } = wp.editor;
4 const { registerBlockType } = wp.blocks;
5 const blockStyle = { backgroundColor: '#900', color: 'fff', padding: '20px' };
6
7 registerBlockType('my-block/rich-text-block', {
8   title: 'My Rich Text Block',
9   icon: 'welcome-write-blog',
10  category: 'common',
11  attributes: {
12    content: {
13      source: 'html',
14      selector: '.text-content',
15    }
16  },
17  edit: function( {className, attributes, setAttributes} ) {
18
19    const { content } = attributes;
20
21    function onChangeContent( newContent ) {
22      setAttributes( {content: newContent} );
23    }
24
25    return (
26      <RichText
27        tagName="p"
28        className= {className}
29        onChange = { onChangeContent }
30        value= {content}
31      />
32    );
33  },
34}
```

```
31     save: function( {attributes, className} ) {
32
33         const { content } = attributes;
34
35         return (
36             <RichText.Content
37                 tagName="p"
38                 className="text-content"
39                 value={content}
40             />
41         );
42     }
43 });
44
```

```
44
45 import './block/block.js';
46 import './block/basic-block';
47 import './block/rich-text-block';
```

INSIDE: /local_dev_site/wp-content/plugins/my-block

```
├── .gitignore
├── plugin.php
├── package.json
├── README.md
|
├── dist
|   ├── blocks.build.js
|   ├── blocks.editor.build.css
|   └── blocks.style.build.css
|
└── src
    ├── block
    |   ├── rich-text-block
    |   ├── block.js
    |   ├── editor.scss
    |   └── style.scss
    |
    ├── blocks.js
    ├── common.scss
    └── init.php
```

Another example

```
registerBlockType('namespace/richtext-block', {
  title: __('RichText Block'),
  description: __('A simple block using the RichText element'),
  icon: 'shield',
  category: 'common',
  keywords: [
    __('richtext-block'),
    __('RichText Block'),
    __('RichText')
  ],
  attributes: {
    content: {
      type: 'array',
      source: 'children',
      selector: 'h2',
    },
  },
  edit: function ({ attributes, setAttributes, className, isSelected }) {
    return (
      <RichText
        tagName="h2"
        className={className}
        value={attributes.content}
        onChange={({content}) => setAttributes({ content })}
        placeholder={__('Enter text...', 'custom-block')}
        keepPlaceholderOnFocus={true}
      />
    );
  },
  save: function( { attributes } ) {
    return (
      <RichText.Content tagName="h2" value={ attributes.content } />
    );
  }
});
```

Slider block

```
<?php
/*
Plugin Name: Custom Slider Block
Description: Custom Gutenberg Slider Block
Version: 1.0
Author: Your Name
*/
// Enqueue your script and styles
function custom_slider_enqueue_scripts() {
    wp_enqueue_script(
        'custom-slider-block',
        plugins_url('dist/block.js', __FILE__),
        array('wp-blocks', 'wp-element', 'wp-editor'),
        true
    );
}
add_action('enqueue_block_editor_assets', 'custom_slider_enqueue_scripts');
```

block.js code

```
import { registerBlockType } from '@wordpress/blocks';
import { InspectorControls } from '@wordpress/editor';
import { PanelBody, TextControl } from '@wordpress/components';

registerBlockType('custom-slider/slider-block', {
    title: 'Custom Slider',
    icon: 'slides',
    category: 'common',
    attributes: {
        slides: {
            type: 'array',
            default: [],
        },
    },
});
```

```
edit: function (props) {
  const { attributes, setAttributes } = props;

  return (
    <div className={props.className}>
      <InspectorControls>
        <PanelBody title="Slider Settings">
          <TextControl
            label="Add a slide"
            value=""
            onChange={(slide) => {
              const newSlides = [...attributes.slides,
                slide];
              setAttributes({ slides: newSlides });
            }}
          />
        </PanelBody>
      </InspectorControls>
      <div className="slider">
        {/* Render your slider here */}
        {attributes.slides.map((slide, index) => (
          <div key={index} className="slide">
            {slide}
          </div>
        )));
      </div>
    </div>
  );
},

save: function () {
  // Save your slider content here
  return null;
},
});
```

Card block Development

```
const { RichText, MediaUpload, PlainText } = wp.editor;
const { registerBlockType } = wp.blocks;
const { Button } = wp.components;

// Import our CSS files
import './style.scss';
import './editor.scss';

registerBlockType('card-block/main', {
    title: 'Card',
    icon: 'heart',
    category: 'common',
    attributes: {
        title: {
            source: 'text',
            selector: '.card__title'
        },
        body: {
            type: 'array',
            source: 'children',
            selector: '.card__body'
        },
        imageAlt: {
            attribute: 'alt',
            selector: '.card__image'
        },
        imageUrl: {
            attribute: 'src',
            selector: '.card__image'
        }
    },
    edit({ attributes, className, setAttributes }) {

        const getImageButton = (openEvent) => {
            if(attributes.imageUrl) {
                return (
                    <img
                        src={ attributes.imageUrl }
                        onClick={ openEvent }
                        className="image"
                    />
                );
            }
        }

        return (
            <div
                className={ className }
            >
                <RichText
                    value={ attributes.title }
                    onChange={ setAttributes }
                />
                <div>
                    { attributes.body.map((child, index) => {
                        if(child.type === 'media') {
                            return getImageButton();
                        }
                        return child;
                    })}
                </div>
            </div>
        );
    }
});
```

```

        );
    }
    else {
      return (
        <div className="button-container">
          <Button
            onClick={ openEvent }
            className="button button-large"
          >
            Pick an image
          </Button>
        </div>
      );
    }
  );
}

return (
  <div className="container">
    <MediaUpload
      onSelect={ media => { setAttributes({ imageAlt: media.alt,
imageUrl: media.url }); } }
      type="image"
      value={ attributes.imageID }
      render={ ({ open }) => getImageButton(open) }
    />
    <PlainText
      onChange={ content => setAttributes({ title: content }) }
      value={ attributes.title }
      placeholder="Your card title"
      className="heading"
    />
    <RichText
      onChange={ content => setAttributes({ body: content }) }
      value={ attributes.body }
      multiline="p"
      placeholder="Your card text"
      formattingControls={ ['bold', 'italic', 'underline'] }
      isSelected={ attributes.isSelected }
    />
  </div>
);
},

```

```
save({ attributes }) {  
  
  const cardImage = (src, alt) => {  
    if(!src) return null;  
  
    if(alt) {  
      return (  
        <img  
          className="card__image"  
          src={ src }  
          alt={ alt }  
        />  
      );  
    }  
  
    // No alt set, so let's hide it from screen readers  
    return (  
      <img  
        className="card__image"  
        src={ src }  
        alt=""  
        aria-hidden="true"  
      />  
    );  
  };  
  
  return (  
    <div className="card">  
      { cardImage(attributes.imageUrl, attributes.imageAlt) }  
      <div className="card__content">  
        <h3 className="card__title">{ attributes.title }</h3>  
        <div className="card__body">  
          { attributes.body }  
        </div>  
      </div>  
    </div>  
  );  
}  
});
```

Custom Block" with a simple text field to the Gutenberg editor.

Create a Plugin Directory:

Create a directory for your custom Gutenberg block plugin in the `wp-content/plugins/` directory of your WordPress installation. Name it something like `custom-gutenberg-block`.

Inside your plugin directory, create the main PHP file (e.g., `custom-gutenberg-block.php`) and add the following code:

```
<?php
/*
Plugin Name: Custom Gutenberg Block
Description: A simple custom Gutenberg block example.
*/

// Register the block script and style.
function custom_gutenberg_block_enqueue() {
    wp_enqueue_script(
        'custom-gutenberg-block',
        plugin_dir_url(__FILE__) . 'block.js',
        array('wp-blocks', 'wp-editor'),
        filemtime(plugin_dir_path(__FILE__) . 'block.js')
    );

    wp_enqueue_style(
        'custom-gutenberg-block',
        plugin_dir_url(__FILE__) . 'style.css',
        array('wp-edit-blocks'),
        filemtime(plugin_dir_path(__FILE__) . 'style.css')
    );
}
add_action('enqueue_block_editor_assets', 'custom_gutenberg_block_enqueue');
```

We enqueue the JavaScript and CSS files needed for our custom block. The JavaScript file (`block.js`) will define the block's behavior, and the CSS file (`style.css`) will provide styling.

Create the JavaScript File (`block.js`):

Inside your plugin directory, create a `block.js` file with the following code:

```
<script>
(function() {
    var el = wp.element.createElement;
    var registerBlockType = wp.blocks.registerBlockType;
    var TextControl = wp.components.TextControl;
    var RichText = wp.editor.RichText;

registerBlockType('custom-gutenberg-block/custom-block', {
    title: 'Custom Block',
    icon: 'shield',
    category: 'common',
    attributes: {
        content: {
            type: 'string',
            default: 'Hello, Custom Block!'
        }
    },
    edit: function(props) {
        return el(
            'div',
            {},
            el(
                TextControl,
                {
                    value: props.attributes.content,
                    onChange: function(newContent) {
                        props.setAttributes({ content: newContent });
                    },
                    placeholder: 'Enter your custom content here...'
                }
            )
        );
    },
    save: function(props) {
        return el(RichText.Content, {
            tagName: 'p',
            value: props.attributes.content
        });
    }
});
})();
</script>
```

This JavaScript code registers a custom block named "Custom Block" with a text field for entering content.

Create the CSS File (style.css):

Inside your plugin directory, create a `style.css` file to add custom styling for your block.

Activate the Plugin:

In the WordPress admin dashboard, navigate to the "Plugins" section and activate your "Custom Gutenberg Block" plugin.

Use the Custom Block:

In the Gutenberg editor, you should now see a "Custom Block" in the "Common" category. Add the block to your post or page and use the text field to enter your custom content.

Custom Gutenberg block in JSX procedure

Creating a custom Gutenberg block in JSX involves several steps, as Gutenberg block development primarily relies on JavaScript, especially JSX, which is a JavaScript syntax extension often used with React. Here's a step-by-step procedure to create a custom Gutenberg block in JSX:

Prerequisites:

1. You should have a development environment set up with Node.js and npm installed.
2. Familiarity with JavaScript, React, and JSX is beneficial.

Procedure:

1. Create a WordPress Plugin Directory:

Start by creating a new directory for your custom Gutenberg block plugin in the `wp-content/plugins/` directory of your WordPress installation. Name it something like `custom-gutenberg-block`.

2. Inside your plugin directory, create the main PHP file (e.g., `custom-gutenberg-block.php`) and add the following code:

```
<?php
/*
Plugin Name: Custom Gutenberg Block
Description: A custom Gutenberg block developed with JSX.
*/

// Enqueue the JavaScript file containing your block.
function enqueue_custom_block() {
    wp_enqueue_script(
        'custom-gutenberg-block',
        plugin_dir_url(__FILE__) . 'build/index.js',
        array('wp-blocks', 'wp-editor'),
        filemtime(plugin_dir_path(__FILE__) . 'build/index.js')
    );
}
add_action('enqueue_block_editor_assets', 'enqueue_custom_block');
```

we're enqueueing the JavaScript file (`index.js`) that contains our custom block.

Set Up Your JavaScript Development Environment:

You'll need a development environment to write and compile JSX into JavaScript. You can use tools like Webpack and Babel for this purpose. Create a `src` directory in your plugin folder for your JavaScript source files.

Create a JSX File for Your Block:

Inside the `src` directory, create a JSX file for your custom block (e.g., `custom-block.js`). Here's a simplified example:

```

const { registerBlockType } = wp.blocks;
const { TextControl } = wp.components;

registerBlockType('custom-gutenberg-block/custom-block', {
    title: 'Custom Block',
    icon: 'shield',
    category: 'common',
    attributes: {
        content: {
            type: 'string',
            default: 'Hello, Custom Block!',
        },
    },
    edit: function (props) {
        return (
            <div>
                <h2>Custom Gutenberg Block</h2>
                <TextControl
                    label="Enter custom content"
                    value={props.attributes.content}
                    onChange={(newContent) => props.setAttributes({ content: newContent })}
                />
            </div>
        );
    },
    save: function (props) {
        return <p>{props.attributes.content}</p>;
    },
});

```

This JSX code registers a custom block that includes a simple text input field.

Compile JSX to JavaScript:

Use a build tool like Webpack and Babel to compile your JSX code into JavaScript. You'll need to configure your build process to output the compiled JavaScript file into a directory like `build`. Ensure that your compiled JavaScript file is named `index.js`.

Activate the Plugin and Use the Block:

In your WordPress admin dashboard, activate the "Custom Gutenberg Block" plugin. You can then use the "Custom Block" in the Gutenberg editor.

Custom Gutenberg block development Node Js procedure

Creating custom Gutenberg blocks in WordPress allows you to extend the functionality of the Gutenberg block editor by adding your own custom blocks with unique features. Here is a step-by-step procedure for developing a custom Gutenberg block:

Step 1: Set Up Your Development Environment

- 1.A local or remote WordPress installation.
- 2.A code editor for writing JavaScript, CSS, and PHP code.
- 3.Node.js and npm (Node Package Manager) installed on your computer.

Step 2: Create a New WordPress Plugin

Custom Gutenberg blocks are typically packaged as WordPress plugins. Create a new directory for your plugin inside the `wp-content/plugins/` directory of your WordPress installation. Name it something unique and descriptive, like "my-custom-gutenberg-block."

Inside your plugin directory, create a main PHP file, e.g., `my-custom-gutenberg-block.php`. This file will serve as the entry point for your plugin.

Step 3: Initialize Your Plugin

In your main PHP file (`my-custom-gutenberg-block.php`), start by defining the basic plugin information and initializing it. Here's a minimal example:

```
<?php  
/**  
 * Plugin Name: My Custom Gutenberg Block  
 * Description: A custom Gutenberg block for WordPress.  
 * Version: 1.0  
 * Author: Your Name  
 */
```

```
// Initialize your plugin here.
```

Step 4: Set Up the Block Development Environment

To build and manage your custom Gutenberg block, you'll need to set up a development environment using npm and the WordPress scripts package. Open your terminal, navigate to your plugin's directory, and run the following commands:

```
npm init -y
```

```
npm install @wordpress/scripts --save-dev
```

Create a `src` directory inside your plugin directory to store your block's JavaScript files.

Step 5: Create the Block JavaScript File

Inside the `src` directory, create a JavaScript file for your block, e.g., `my-custom-block.js`. This file will contain the JavaScript code for your block.

Step 6: Define Your Block

In your block JavaScript file (`my-custom-block.js`), define your custom block using the WordPress block development API. Here's a simplified example of a custom block that displays a "Hello, Gutenberg!" message:

```
// Import necessary dependencies
import { registerBlockType } from '@wordpress/blocks';
import { TextControl } from '@wordpress/components';

// Register the block
registerBlockType('my-custom-gutenberg-block/hello-gutenberg', {
    title: 'Hello Gutenberg Block',
    icon: 'smiley',
    category: 'common',
    attributes: {
        message: {
            type: 'string',
            default: 'Hello, Gutenberg!',
        },
    },
});
```

```
edit: function(props) {
  return (
    <div>
      <TextControl
        label="Message"
        value={props.attributes.message}
        onChange={message => props.setAttributes({ message })}
      />
    </div>
  );
},
save: function() {
  return null; // The block's content is saved on the server.
},
);
};
```

- We import the necessary WordPress block development functions and components.
- We register the custom block using the `registerBlockType` function.
- The block has a title, icon, and is categorized under "Common."
- It has a single attribute named "message" for the block's content.
- The `edit` function defines the block's editing interface using JSX (React).
- The `save` function is empty because the block's content is saved on the server.

Step 7: Build and Enqueue Your Block JavaScript

Back in your terminal, run the following command to build your block JavaScript:

npm run build

This command generates a production-ready JavaScript file in the `build` directory of your plugin.

Next, enqueue the block JavaScript in your plugin's PHP file (`my-custom-gutenberg-block.php`). You can use the `wp_enqueue_script` function:

```
function my_custom_gutenberg_block_enqueue() {
    wp_enqueue_script(
        'my-custom-gutenberg-block',
        plugins_url('build/my-custom-block.js', __FILE__),
        array('wp-blocks', 'wp-components', 'wp-editor'),
        filemtime(plugin_dir_path(__FILE__) . 'build/my-custom-block.js')
    );
}
add_action('enqueue_block_editor_assets','my_custom_gutenberg_block_enqueue');
```

This code enqueues your block's JavaScript and its dependencies for use in the Gutenberg editor.

Step 8: Activate Your Plugin

Activate your custom plugin through the WordPress admin dashboard.

Step 9: Add Your Custom Block in Gutenberg

After activation, you can add your custom block to any post or page using the Gutenberg block editor. Search for your block's title ("Hello Gutenberg Block" in this example) and add it to the content. You can then configure and customize the block as needed.

Step 10: Further Development and Styling

From here, you can further develop your custom Gutenberg block by adding additional attributes, settings, and styling using CSS. You can also integrate server-side functionality to save and retrieve block data

Custom block for Elementor Pro

Developing a custom block for Elementor Pro involves creating a custom widget that integrates seamlessly with the Elementor page builder. Elementor provides a framework for building custom widgets that you can use to add new elements or functionality to your pages. Below are the steps to develop a custom block for Elementor Pro:

Step 1: Set Up Your Development Environment

Before you start, make sure you have the following tools and environment set up:

- *WordPress installed with Elementor Pro.
- *A code editor for writing PHP, CSS, and JavaScript.
- *Access to your server's file system or an environment where you can create and upload files.

Step 2: Create a Plugin for Your Custom Block

To keep your custom block organized, it's a good practice to create a custom plugin for it. Create a new directory in your WordPress plugins folder (usually located at `wp-content/plugins/`) and give it a unique name. Inside this directory, create a PHP file to initialize your plugin.

For example, you can create a file named `custom-elementor-block.php` with the following content:

```
<?php
/*
Plugin Name: Custom Elementor Block
Description: A custom Elementor Pro block.
Version: 1.0
Author: Your Name
*/
// Define plugin functionality here.
```

Step 3: Define the Custom Widget Class

To create your custom block, you'll need to define a custom widget class that extends the `\Elementor\Widget_Base` class. This class should be placed within your plugin file or a separate PHP file included by your plugin file.

Here's a basic example:

```
class Custom_Elementor_Block_Widget extends \Elementor\Widget_Base {

    public function get_name() {
        return 'custom-elementor-block'; // The name of your widget.
    }

    public function get_title() {
        return 'Custom Elementor Block'; // The title displayed in the
Elementor editor.
    }

    public function get_icon() {
        return 'fa fa-code'; // The icon for your block (Font Awesome icon
class).
    }

    public function get_categories() {
        return ['general']; // The category in which your block will appear.
    }

    protected function _register_controls() {
        // Define your widget controls (settings) here.
    }

    protected function render() {
        // Define how your block should be displayed on the front end here.
    }
}

// Register the widget
\Elementor\Plugin::instance()->widgets_manager->register_widget_type(new
Custom_Elementor_Block_Widget());
```

This code creates a basic widget class with essential properties and methods. You will need to customize it to suit your specific block's functionality.

Step 4: Define Widget Controls (Settings)

In the `_register_controls` method of your widget class, define the settings (controls) for your custom block. These controls allow users to configure the block's appearance and behavior in the Elementor editor. Elementor provides various control types, such as text fields, color pickers, and image selectors. Here's an example of how to define a text control:

```
protected function _register_controls() {
    $this->start_controls_section(
        'section_content',
        [
            'label' => ___('Content', 'your-text-domain'),
        ]
    );
    $this->add_control(
        'custom_text',
        [
            'label' => ___('Custom Text', 'your-text-domain'),
            'type' => \Elementor\Controls_Manager::TEXT,
            'default' => ___('Hello, World!', 'your-text-domain'),
        ]
    );
    $this->end_controls_section();
}
```

Step 5: Define the Frontend Output

In the `render` method of your widget class, define how your custom block should be displayed on the front end of the website. You can use PHP and HTML to generate the block's output. Here's a simple example:

```
protected function render() {
    $settings = $this->get_settings_for_display();

    echo '<div class="custom-elementor-block">';
    echo '<p>' . esc_html($settings['custom_text']) . '</p>';
    echo '</div>';
}
```

Step 6: Style Your Block

To ensure your block looks good on the front end, you can add custom CSS styles using Elementor's built-in styling options or by enqueueing an external stylesheet.

Step 7: Test Your Custom Block

Activate your custom plugin, and you should be able to add your custom block to Elementor pages by searching for it in the Elementor editor's widget panel. Configure the block's settings, add it to your page, and preview it to ensure it works as expected.

Step 8: Refine and Document

Test your block thoroughly, make any necessary refinements, and document its usage and customization options for users or other developers.

This is a basic guide to creating a custom block for Elementor Pro. Depending on your requirements, you may need to implement additional features, such as dynamic content, dynamic tags, and advanced controls. Be sure to consult the Elementor developer documentation for detailed information on creating custom blocks and widgets:

[Elementor Developers Documentation](#).

WP-CLI

Installing on Windows

Install via [composer as described above](#) or use the following method.

Make sure you have php installed and [in your path](#) so you can execute it globally.

Download [wp-cli.phar](#) manually and save it to a folder, for example `c:\wp-cli`

Create a file named `wp.bat` in `c:\wp-cli` with the following contents:

```
@ECHO OFF  
php "c:/wp-cli/wp-cli.phar" %*
```

Add `c:\wp-cli` to your path:

```
setx path "%path%;c:\wp-cli"
```

You can now use WP-CLI from anywhere in Windows command line.

Environment variable path setting example:

```
E:\wp-cli>wp --info  
OS: Windows NT 10.0 build 19043 (Windows 10) AMD64  
Shell: C:\WINDOWS\system32\cmd.exe  
PHP binary: E:\xampp\php\php.exe  
PHP version: 7.3.11  
php.ini used: E:\xampp\php\php.ini  
MySQL binary:  
MySQL version:  
SQL modes:  
WP-CLI root dir: phar://wp-cli.phar/vendor/wp-cli/wp-cli  
WP-CLI vendor dir: phar://wp-cli.phar/vendor  
WP_CLI phar path: E:\wp-cli  
WP-CLI packages dir:  
WP-CLI global config:  
WP-CLI project config:  
WP-CLI version: 2.6.0
```

```
E:\wp-cli>setx path "%path%;e:\wp-cli"
```

```
SUCCESS: Specified value was saved.
```

```
E:\wp-cli>cd..
```

```
E:\>wp --info  
'wp' is not recognized as an internal or external command,  
operable program or batch file.
```

```
E:\>
```

The recommended way to install [WP-CLI](#) is by downloading the Phar build (archives similar to Java JAR files, [see this article for more detail](#)), marking it executable, and placing it on your PATH.

First, download [wp-cli.phar](#) using `wget` or `curl`. For example:

```
curl -O https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar
```

Then, check if it works:

```
php wp-cli.phar --info
```

To be able to type just `wp`, instead of `php wp-cli.phar`, you need to make the file executable and move it to somewhere in your PATH. For example:

```
chmod +x wp-cli.phar
sudo mv wp-cli.phar /usr/local/bin/wp
```

Now try running `wp --info`. If WP-CLI is installed successfully, you'll see output like this:

wp+ enter key (wp help command) for showing all command

```
SYNOPSIS
wp <command>

SUBCOMMANDS
cache          Adds, removes, fetches, and flushes the WP Object Cache object.
cap            Adds, removes, and lists capabilities of a user role.
cli             Reviews current WP-CLI info, checks for updates, or views defined aliases.
comment        Creates, updates, deletes, and moderates comments.
config         Generates and reads the wp-config.php file.
core           Downloads, installs, updates, and manages a WordPress installation.
cron           Tests, runs, and deletes WP-Cron events; manages WP-Cron schedules.
db              Performs basic database operations using credentials stored in wp-config.php.
embed          Inspects oEmbed providers, clears embed cache, and more.
eval            Executes arbitrary PHP code.
eval-file       Loads and executes a PHP file.
export         Exports WordPress content to a WXR file.
help            Gets help on WP-CLI, or on a specific command.
i18n           Provides internationalization tools for WordPress projects.
import         Imports content from a given WXR file.
language        Installs, activates, and manages language packs.
maintenance-mode Activates, deactivates or checks the status of the maintenance mode of a site.
media           Imports files as attachments, regenerates thumbnails, or lists registered image sizes.
menu            Lists, creates, assigns, and deletes the active theme's navigation menus.
```

Sub command

```
wp core

DESCRIPTION
Downloads, installs, updates, and manages a WordPress installation.

SYNOPSIS
wp core <command>

SUBCOMMANDS
check-update          Checks for WordPress updates via Version Check API.
download              Downloads core WordPress files.
install               Runs the standard WordPress installation process.
is-installed          Checks if WordPress is installed.
multisite-convert    Transforms an existing single-site installation into a multisite installation.
multisite-install     Installs WordPress multisite from scratch.
update                Updates WordPress to a newer version.
update-db             Runs the WordPress database update procedure.
verify-checksums      Verifies WordPress files against WordPress.org's checksums.
version               Displays the WordPress version.

EXAMPLES
# Download WordPress core
$ wp core download --locale=nl_NL
-- More --
```

a

```
wp core download

DESCRIPTION
Downloads core WordPress files.

SYNOPSIS
wp core download [<download-url>] [--path=<path>] [--locale=<locale>] [--version=<version>] [--skip-content] [--force] [--insecure]

Downloads and extracts WordPress core files to the specified path. Uses current directory when no path is specified. Downloaded build is verified to have the correct md5 and then cached to the local filesystem. Subsequent uses of command will use the local cache if it still exists.

OPTIONS
[<download-url>]
Download directly from a provided URL instead of fetching the URL from the wordpress.org servers.

[--path=<path>]
Specify the path in which to install WordPress. Defaults to current directory.

[--locale=<locale>]
Select which language you want to download.

[--version=<version>]
Select which version you want to download. Accepts a version number, 'latest' or 'nightly'.

[--skip-content]
Download WP without the default themes and plugins.

[--force]
Overwrites existing files, if present.
```

Downloading example

```
E:\xampp\htdocs\wpcli>wp core download
Downloading WordPress 6.0.1 (en_US)...
Using cached file 'C:\Users\SD/.wp-cli/cache/core/wordpress-6.0.1-en_US.tar.gz'...
```

```
wp db

DESCRIPTION
    Performs basic database operations using credentials stored in wp-config.php.

SYNOPSIS
    wp db <command>

SUBCOMMANDS

check      Checks the current status of the database.
clean       Removes all tables with `{$table_prefix}` from the database.
cli        Opens a MySQL console using credentials from wp-config.php
columns    Displays information about a given table.
create     Creates a new database.
drop       Deletes the existing database.
export     Exports the database to a file or to STDOUT.
import     Imports a database from a file or from STDIN.
optimize   Optimizes the database.
prefix     Displays the database table prefix.
query      Executes a SQL query against the database.
repair    Repairs the database.
reset     Removes all tables from the database.
search    Finds a string in the database.
size      Displays the database name and size.
```

E:\xampp\htdocs\wpcli>_

```
E:\xampp\htdocs\wpcli>wp post create --post_content="test post content" --post_title="test post" --post_status=publish
Success: Created post 5.

E:\xampp\htdocs\wpcli>wp post create --post_content="second test post content" --post_title="second test post" --post_status=publish --post_author=1
Success: Created post 7.

E:\xampp\htdocs\wpcli>wp post list
+---+-----+-----+-----+-----+
| ID | post_title      | post_name      | post_date      | post_status |
+---+-----+-----+-----+-----+
| 7  | second test post | second-test-post | 2022-07-24 14:12:34 | publish      |
| 5  | test post        | test-post      | 2022-07-24 14:11:15 | publish      |
| 1  | Hello world!     | hello-world    | 2022-07-24 12:27:51 | publish      |
+---+-----+-----+-----+-----+

E:\xampp\htdocs\wpcli>wp post create --post_content="test page content" --post_title="test page" --post_status=publish --post_author=1 --post_type=page
Success: Created post 8.

E:\xampp\htdocs\wpcli>
```

```
E:\xampp\htdocs\wpcli>wp post generate --count=4 --post_type=page
Generating posts 100% [=====] 0:01 / 0:01

E:\xampp\htdocs\wpcli>wp post list --post_type=page
+-----+-----+-----+-----+
| ID | post_title | post_name | post_date | post_status |
+-----+-----+-----+-----+
| 11 | test page | test-page-3 | 2022-07-30 00:00:00 | future |
| 22 | Page 5 | post-5 | 2022-07-24 18:32:35 | publish |
| 23 | Page 6 | post-6 | 2022-07-24 18:32:35 | publish |
| 24 | Page 7 | post-7 | 2022-07-24 18:32:35 | publish |
| 25 | Page 8 | post-8 | 2022-07-24 18:32:35 | publish |
| 10 | test page | test-page-2 | 2022-07-24 14:15:17 | publish |
| 8 | test page | test-page | 2022-07-24 14:14:17 | publish |
| 2 | About Us | sample-page | 2022-07-24 12:27:51 | publish |
| 3 | Privacy Policy | privacy-policy | 2022-07-24 12:27:51 | draft |
+-----+-----+-----+-----+
```

```
E:\xampp\htdocs\wpcli>wp theme list
+-----+-----+-----+-----+
| name | status | update | version |
+-----+-----+-----+-----+
| hello-elementor | active | none | 2.6.1 |
| twentytwenty | inactive | none | 2.0 |
| twentytwentyone | inactive | none | 1.6 |
| twentytwentytwo | inactive | none | 1.2 |
+-----+-----+-----+-----+

E:\xampp\htdocs\wpcli>wp theme install ./hestia.3.0.23.zip --activate
Unpacking the package...
Installing the theme...
Theme installed successfully.
Activating 'hestia'...
Success: Switched to 'Hestia' theme.
Success: Installed 1 of 1 themes.
```

```
wp plugin <command>

SUBCOMMANDS

  activate          Activates one or more plugins.
  auto-updates     Manages plugin auto-updates.
  deactivate        Deactivates one or more plugins.
  delete ↗         Deletes plugin files without deactivating or uninstalling.
  get               Gets details about an installed plugin.
  install           Installs one or more plugins.
  is-active         Checks if a given plugin is active.
  is-installed      Checks if a given plugin is installed.
  list              Gets a list of plugins.
  path              Gets the path to a plugin or to the plugin directory.
```

```
E:\xampp\htdocs\wpcli>wp user list-caps demo2@test.com
upload_files
edit_posts
edit_published_posts
publish_posts
read
level_2
level_1
level_0
delete_posts
delete_published_posts
rank_math_onpage_analysis
rank_math_onpage_general
rank_math_onpage_snippet
rank_math_onpage_social
author
```

```
wp user generate
```

DESCRIPTION

Generates some users.

SYNOPSIS

```
wp user generate [--count=<number>] [--role=<role>] [--format=<format>]
```

Creates a specified number of new users with dummy data.

OPTIONS

```
[--count=<number>]
```

How many users to generate?

default: 100

```
[--role=<role>]
```

The role of the generated users. Default: default role from WP

```
-- More --
```

```
E:\xampp\htdocs\wpcli>wp user generate --count=5 --role=author
Generating users 100% [=====] 0:01 / 0:02

E:\xampp\htdocs\wpcli>wp user list
+---+-----+-----+-----+-----+-----+
| ID | user_login      | display_name    | user_email        | user_registered | roles          |
+---+-----+-----+-----+-----+-----+
| 1  | admin           | admin           | admin@admin.com   | 2022-07-24 12:27:51 | administrator |
| 3  | demo2           | demo 2          | demo2@test.com   | 2022-07-31 13:25:22 | author         |
| 5  | diwakar          | Diwakar Academy | diwakar@classes.com | 0000-00-00 00:00:00 | editor         |
| 4  | diwakar_academy | Diwakar Academy | diwakar@academy.com | 0000-00-00 00:00:00 | author         |
| 2  | domo1            | demo1           | demo1@test.com   | 2022-07-31 13:24:21 | editor         |
| 6  | user_1_5          | User 5          |                   | 2022-07-31 14:08:43 | author         |
| 7  | user_1_6          | User 6          |                   | 2022-07-31 14:08:43 | author         |
| 8  | user_1_7          | User 7          |                   | 2022-07-31 14:08:43 | author         |
| 9  | user_1_8          | User 8          |                   | 2022-07-31 14:08:44 | author         |
| 10 | user_1_9          | User 9          |                   | 2022-07-31 14:08:44 | author         |
+---+-----+-----+-----+-----+-----+
```

WP CLI several benefits for developers, site administrators, and anyone who works with WordPress:

- Automation:** WP-CLI allows you to automate various tasks, such as updating plugins and themes, creating backups, importing/exporting content, and more. This can save you a lot of time and reduce the potential for human error.
- Speed:** Command-line operations are typically faster than performing the same tasks through the WordPress admin interface. This can be especially helpful when working with large websites or complex operations.
- Scripting:** You can write custom scripts and batch operations with WP-CLI, making it easier to perform repetitive tasks. This is useful for tasks like setting up a new WordPress instance, migrating content, or performing site maintenance.
- Managing Plugins and Themes:** WP-CLI allows you to install, activate, update, and delete plugins and themes with simple commands. This is a faster and more efficient way to manage your site's extensions.
- Database Management:** You can interact with the WordPress database using WP-CLI, making it easier to run SQL queries, repair the database, or perform other maintenance tasks.
- User Management:** WP-CLI makes it easy to create, update, and delete user accounts, change user passwords, and manage user roles and permissions.
- Debugging:** You can use WP-CLI to troubleshoot issues on your site by enabling or disabling plugins, themes, or debugging features.

8. **Content Management:** WP-CLI allows you to create and manage content, including posts, pages, and custom post types. You can even generate dummy content for testing purposes.
9. **Security:** WP-CLI can help enhance the security of your WordPress site by automating security tasks such as user password resets, security plugin management, and vulnerability scans.
10. **Version Control Integration:** WP-CLI can be integrated into version control workflows (e.g., Git), making it easier to deploy changes to your WordPress site and ensure consistency across different environments.
11. **Multisite Management:** If you're running a WordPress multisite network, WP-CLI simplifies the management of network-wide settings, sites, and user accounts.
12. **Customization:** WP-CLI is highly customizable, allowing you to create custom commands and scripts tailored to your specific needs.
13. **Community and Third-Party Plugins:** There is a growing community of developers who contribute to the WP-CLI project, and there are also third-party packages and plugins available to extend its functionality.

1. **Efficiency:** WP-CLI allows you to perform various tasks more efficiently than using the WordPress admin dashboard. Commands are typically faster and can be automated, saving you time and effort.
2. **Scripting and Automation:** You can create custom scripts and automate tasks with WP-CLI. This is particularly useful for batch operations, maintenance tasks, and deployment processes.
3. **Batch Operations:** You can perform actions on multiple items at once. For example, you can install, activate, or update multiple plugins or themes in one command.
4. **Database Management:** WP-CLI provides commands for database operations, allowing you to interact with the database, run SQL queries, repair tables, and manage the database efficiently.
5. **User Management:** You can create, update, and delete user accounts, change user roles and permissions, and perform user-related tasks via WP-CLI.
6. **Theme and Plugin Management:** WP-CLI simplifies theme and plugin management. You can install, activate, deactivate, update, or delete themes and plugins from the command line.

7. **Content Management:** You can create and manage posts, pages, custom post types, and even generate dummy content for testing purposes using WP-CLI commands.
8. **Site Maintenance:** WP-CLI offers tools for site maintenance, including options to optimize the database, repair the database, and perform other maintenance tasks.
9. **Debugging and Troubleshooting:** You can troubleshoot issues by enabling or disabling plugins and themes, setting debugging options, and diagnosing problems more efficiently.
10. **Security:** WP-CLI can be used to enhance site security by automating tasks like password resets and managing security plugins.
11. **Multisite Management:** For WordPress multisite installations, WP-CLI simplifies the management of network-wide settings, sites, and users.
12. **Customization:** You can extend WP-CLI's functionality by creating custom commands to suit your specific needs or integrate it with other tools and scripts.
13. **Version Control Integration:** WP-CLI can be integrated into version control workflows (e.g., Git) to streamline development, deployment, and consistency across different environments.
14. **Third-Party Plugins and Packages:** There is a community of developers who contribute to WP-CLI, and third-party packages and plugins can extend its functionality for various use cases.

Reg

WP-CLI offers advanced capabilities that can greatly streamline your WordPress development and management tasks. Here are some advanced use cases for WP-CLI:

1. **Custom Command Development:** One of the most powerful features of WP-CLI is the ability to create custom commands tailored to your specific needs. These commands can be used to automate complex or site-specific tasks. For example, you could create a custom command to migrate data from a legacy CMS to WordPress.
2. **Continuous Integration and Deployment (CI/CD):** WP-CLI can be integrated into CI/CD pipelines to automate testing, deployment, and maintenance processes. You can use WP-CLI to automate the deployment of code, database updates, and content synchronization across environments.
3. **Multisite Network Management:** WP-CLI can be used to manage WordPress multisite networks effectively. You can create, update, and delete sites, users, and themes across the network, as well as synchronize content and configurations.
4. **Headless WordPress:** If you're building a headless WordPress site where WordPress serves as a content management system, you can use WP-CLI to automate content synchronization between WordPress and your front-end application.

5. **Database Migration and Seeding:** WP-CLI is useful for database migration tasks. You can export and import database data, create database backups, seed databases with dummy content for testing, and perform data transformations.
 6. **Theme and Plugin Development:** Developers can use WP-CLI to scaffold themes and plugins, generate template files, create custom post types and taxonomies, and manage code repositories. This is particularly useful when working on large projects with numerous theme and plugin files.
 7. **Custom Post Type and Taxonomy Management:** You can create, update, or delete custom post types and taxonomies using WP-CLI, streamlining the development of complex content structures.
 8. **Internationalization and Localization:** WP-CLI provides tools for generating translation files and managing language packs for internationalization and localization efforts, making it easier to build multilingual sites.
 9. **Performance Optimization:** WP-CLI commands can be used to optimize site performance by regenerating image thumbnails, purging cache, and minifying styles and scripts.
 10. **Security and Auditing:** Advanced users can implement automated security checks and audits using WP-CLI, allowing you to monitor the integrity of your WordPress installation and plugins.
-
11. **Scalability:** For large WordPress sites, you can use WP-CLI to automate tasks like server provisioning, environment configuration, and content distribution to enhance scalability.
 12. **API Integration:** WP-CLI can be integrated with various APIs to automate content imports, exports, and synchronization with external systems.
 13. **Site Setup and Configuration:** WP-CLI can streamline the setup and configuration of new WordPress installations, including database creation, user setup, and default settings.
 14. **Server and Environment Management:** WP-CLI can help you manage server and hosting environment tasks, such as updating PHP versions, configuring server settings, and checking system requirements.
 15. **Logging and Monitoring:** You can set up automated monitoring and logging scripts using WP-CLI to track site performance, errors, and changes over time.

Step 1 – Installing WP-CLI

In this step, you'll install the latest version of the WP-CLI tool on your server. The tool is packaged in a [Phar file](#), which is a packaging format for PHP applications that makes app deployment and distribution convenient.

You can download the Phar file for WP-CLI through `curl`:

```
$ curl -O https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar
```

[Copy](#)

Once you have downloaded the file, run the following command to verify that it is working:

```
$ php wp-cli.phar --info
```

[Copy](#)

You will receive the following output:

Output

```
OS: Linux 5.4.0-51-generic #56-Ubuntu SMP Mon Oct 5 14:28:49 UTC 2020 x86_64
Shell: /bin/bash
PHP binary: /usr/bin/php7.4
PHP version: 7.4.3
php.ini used: /etc/php/7.4/cli/php.ini
WP-CLI root dir: phar://wp-cli.phar/vendor/wp-cli/wp-cli
WP-CLI vendor dir: phar://wp-cli.phar/vendor
WP_CLI phar path: /home/ayo
WP-CLI packages dir:
WP-CLI global config:
WP-CLI project config:
WP-CLI version: 2.4.0
```

Next, make the file executable with the following command:

```
$ chmod +x wp-cli.phar
```

[Copy](#)

At this point, you can execute the `wp-cli.phar` file directly to access the WP-CLI tool. To make it available globally on the system, move it to your `/usr/local/bin/` directory and rename it to `wp`. This ensures that you can access WP-CLI from any directory by entering the `wp` command at the start of a prompt:

```
$ sudo mv wp-cli.phar /usr/local/bin/wp
```

Copy

Now, you will be able to issue the following command to check the installed version of WP-CLI:

```
$ wp cli version
```

Copy

Output

WP-CLI 2.4.0

In this step, you installed WP-CLI on your server. You can check out [alternative installation methods](#) in the documentation. In subsequent sections, you'll explore the tasks you can accomplish through the WP-CLI interface.

Step 2 – Configuring WordPress Plugins

It can be tedious to install and manage WordPress plugins through the admin user interface. It's possible to offload such tasks to WP-CLI to make the process much faster. In this section you will learn to install, update, and delete plugins on a WordPress site through the command line.

Before you proceed, make sure you are in the directory of your WordPress installation:

```
$ cd /var/www/wordpress
```

Copy

Remember to change the highlighted directory name to the directory that contains your WordPress installation. This might be your domain name, if you followed the prerequisite tutorials.

Listing Current Plugins

You can list the currently installed plugins on a WordPress site with the following command:

```
$ wp plugin list
```

Copy

It displays a list of plugin names along with their status, version, and an indication of an available update.

Output

name	status	update	version
akismet	inactive	available	4.1.7
hello	inactive	none	1.7.2

Searching for Plugins

You can search for plugins through the search bar on the [WordPress plugin repository page](#) or you can use the following command for quicker access:

```
$ wp plugin search seo
```

[Copy](#)

Once you run this command, you will receive a list of the top 10 plugins that match the search term (as of early 2021). The expected output for the `seo` query is:

Output

Success: Showing 10 of 4278 plugins.

name	slug	rating
Yoast SEO	wordpress-seo	98
All in One SEO	all-in-one-seo-pack	92
Rank Math – SEO Plugin for WordPress	seo-by-rank-math	98
The SEO Framework	autodescription	98
SEOPress, on-site SEO	wp-seopress	98
Slim SEO – Fast & Automated WordPress SEO Plugin	slim-seo	92
W3 Total Cache	w3-total-cache	88
LiteSpeed Cache	litespeed-cache	98
SEO 2021 by Squirrly (Smart Strategy)	squirrly-seo	92
WP-Optimize – Clean, Compress, Cache.	wp-optimize	96

You can go to the next page by using the `--page` flag:

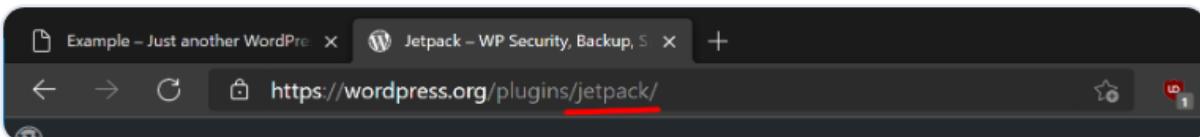
```
$ wp plugin search seo --page=2
```

[Copy](#)

Take note of the value in the `slug` column. You'll use this value to install or update the plugin on the command line.

Installing Plugins

You can install one or more plugins by using the `wp plugin install` command. You find the name of the plugin you want to install (in the `slug` column) and pass it as an argument to `wp plugin install`. You can also find the name of the plugin in the URL of the plugin page.



```
$ wp plugin install jetpack wordpress-seo gutenberg
```

[Copy](#)

The output indicates the progress and completion of the installation of each of the plugins:

Output

name	status	update	version
akismet	inactive	available	4.1.7
gutenberg	inactive	none	9.8.1
hello	inactive	none	1.7.2
jetpack	inactive	none	9.3.1
wordpress-seo	inactive	none	15.6.2

If you want to install a plugin from a remote source other than the WordPress plugin repository, you can pass the zip file's URL as an argument to `wp plugin install`. This can be helpful for installing custom or premium plugins. For example, the following command will install the `myplugin.zip` file hosted on `example.com`. Make sure to replace the highlighted URL with a link to the plugin zip file before running the command:

```
$ wp plugin install https://example.com/wp-content/uploads/myplugin.zip
```

Copy

To install an older version of a plugin in the WordPress repository, specify the desired plugin version through the `--version` flag:

```
$ wp plugin install jetpack --version=8.0
```

Copy

Activating and Deactivating Plugins

You can install and activate plugins in one go by appending the `--activate` flag to `wp plugin install`:

```
$ wp plugin install redirection --activate
```

Copy

To activate or deactivate one or more plugins, use the `wp plugin activate` and `wp plugin deactivate` commands respectively:

```
$ wp plugin activate jetpack gutenberg  
$ wp plugin deactivate jetpack gutenberg
```

Copy

Or you can use the `--all` flag to activate or deactivate all plugins at once. This is useful if you want to debug a problem in your WordPress installation:

```
$ wp plugin activate --all  
$ wp plugin deactivate --all
```

Copy

Updating Plugins

You can update plugins through the `wp plugin update` command. You can choose to update a set of plugins or all of them at once by appending the `--all` flag. For example, to update the `akismet` plugin, you can run the following command:

```
$ wp plugin update akismet
```

Copy

Deleting plugins

To delete WordPress plugins, you can use the `wp plugin delete` command. You can specify one or more plugins to delete like the following:

```
$ wp plugin delete redirection
```

Copy

Your output will confirm the deletion:

Output

```
Deleted 'redirection' plugin.  
Success: Deleted 1 of 1 plugins.
```

You can also delete all the installed plugins in one go by appending the `--all` flag instead of specifying the plugin names one after the other:

```
$ wp plugin delete --all
```

Copy

In this step, you've used WP-CLI to manage the plugins on your WordPress website. It's much faster to perform actions compared to clicking through the admin dashboard. In the next section, you'll leverage WP-CLI for installing and managing WordPress themes.

Step 3 – Configuring Themes

The process of managing themes through WP-CLI is almost identical to the way you can use it to manage plugins. In this section, you'll source and apply new themes to a WordPress website through the `wp theme` subcommand.

First, check what themes you currently have installed on the website:

```
$ wp theme list
```

Copy

You'll receive a list of the installed themes:

something with more features, you can try a search like the following:

```
$ wp theme search color
```

Copy

The output shows there are 832 themes that match the `color` search term:

Output

Success: Showing 10 of 832 themes.

name	slug	rating
Color	color	0
All Colors	all-colors	100
Color Blog	color-blog	98
Color Block	color-block	0
X Blog color	x-blog-color	0
Multicolor Business	multicolor-business	0
ColorNews	colornews	100
Colorist	colorist	100
ColorMag	colormag	98
MultiColors	multic平ors	74

You can page through the results using the `--page` flag. For this example, just go ahead and install the `ColorMag` theme since it has a pretty good rating. The `--activate` flag activates the theme immediately:

```
$ wp theme install colormag --activate
```

Copy

The output will confirm the installation:

Output

```
Installing ColorMag (2.0.4)
Downloading installation package from https://downloads.wordpress.org/theme/colormag.2.0.4.zip...
Unpacking the package...
Installing the theme...
Theme installed successfully.
```

```
$ wp theme update --all
```

Copy

The `wp theme` command offers many subcommands that can help you achieve tasks like getting the details of a theme, checking if a particular theme is installed, or even deleting one or more themes. You can explore all of the options by prepending `help` before the subcommand, as in `wp help theme` or `wp help theme install`.

Now that you can manage themes through WP-CLI, you'll review the options that the tool provides for managing WordPress content.

Step 4 – Creating Posts and Pages

WP-CLI provides several ways to manage content through the command line. It can be more comfortable to write posts in the terminal if you're familiar with a command-line editor like [nano](#) or [vim](#).

You can browse the list of posts on the site with:

```
$ wp post list
```

Copy

You'll receive a list of posts:

Output

ID	post_title	post_name	post_date	post_status
1	Hello world!	hello-world	2021-01-24 12:32:06	publish

The output shows one published post with the title of `Hello world!` and an ID of `1`. To delete this post, use the `wp post delete` command and pass it the post ID:

```
$ wp post delete 1
```

Copy

Your output will confirm the post's deletion:

Output

Success: Trashed post 1.

To create a new post, run the following command:

```
$ wp post create --post_status=publish --post_title="Sample post created with WP-CLI" --e
```

Copy

This command uses the `--post_status` flag to set the status of the post. Setting it to `publish` ensures that the post is published immediately after running the command. If you want to create a draft instead, set the `--post_status` flag to `draft`. The `--post_title` flag is how you can specify the title of the post, while `--edit` causes the post body to be opened in the default system editor (`vim`). You can find out the other flags that you can use in conjunction with the `create` subcommand by typing `wp help post create` in your terminal.

Once the `vim` editor is open, press the `i` key to enter INSERT mode then enter the content of the post into the editor. Once you're done editing the post, exit the `vim` editor by pressing the `esc` button then type `:wq` and press `ENTER`. You will receive the following output after exiting `vim`:

Output

```
Success: Created post 6.
```

If you enter the `wp post list` command again, you will find the post you just created. You can also check the frontend of the website.

Instead of writing the post on the command line, it's also possible to import the post content from a text file. First, you need to create the file. For example:

```
$ touch content.txt
```

Copy

Next, open the file in a command-line editor to add or edit your content:

```
$ nano content.txt
```

Copy

Once you're through with the edits, save and close the file by pressing `CTRL-X` followed by `y` to save. You can import the contents of that file as a WordPress post by using the following command. All you need to do is specify the path to the file after the `create` subcommand. For the example file here, you would run:

```
$ wp post create ./content.txt --post_title='Sample Post Created with WP-CLI' --post_status=auto
```

Copy

If you want to create a page instead of a post, append the `--post_type` flag and set it to `page`:

```
$ wp post create --post_title="A simple page" --post_status=draft --post_type=page
```

Copy

Generating Posts or Pages

WP-CLI also provides an option to cleanly generate posts and pages with dummy data. This is useful if you need custom data to quickly test a theme or plugin that you are developing. The following command is to generate posts. If you don't include additional flags, it will generate 100 posts by default.

```
$ wp post generate
```

Copy

You can change the number of posts generated by using the `--count` flag:

```
$ wp post generate --count=20
```

Copy

If you want to generate pages instead of posts, append the `--post_type` flag and set it to `page`:

```
$ wp post generate --count=20 --post_type=page
```

Copy

You can also use the `wp help post generate` to see other available options that can help you get your desired result.

WordPress Revisions

It is not uncommon for older sites to have tens or hundreds of revisions on their main pages due to years of editing and updating content. Revisions can be helpful in case you need to revert back to a previous version of your content, but they can also hurt the performance if there are too many. You can clean up all the post revisions in the WordPress database by executing the following command:

```
$ wp post delete $(wp post list --post_type='revision' --format=ids) --force
```

Copy

The command enclosed in the parenthesis is evaluated first and it will produce the `ids` of all the post revisions that are present passing them to the `delete` subcommand. The `--force` flag is necessary because posts of type `'revision'` do not support being sent to trash.

Step 5 – Managing Your Database

One of the most useful features of WP-CLI is its ability to interact with the MySQL database. For example, if you need an interactive session, you can enter a MySQL prompt with the following command:

```
$ wp db cli
```

Copy

You can then use the MySQL shell as you normally would and, once you are through with your tasks, exit the shell by typing `exit`.

For one-off queries, you can use the `wp db query` command by passing a valid SQL query as an argument to the command. For example, to list all the registered users in the WordPress database, you could run:

```
$ wp db query "SELECT user_login, ID FROM wp_users;"
```

Copy

You will be presented with an output similar to the following:

```
Output
+-----+----+
| user_login | ID |
+-----+----+
| admin      | 1  |
+-----+----+
```

With `wp db query` you can run any one-off SQL query for the WordPress database.

Backing Up and Restoring

WP-CLI also allows you to back up your WordPress database. Running this following command will place a SQL dump file in the current directory. This file contains your entire WordPress database including your posts, pages, user accounts, menus, and so on:

```
wp db export
```

Once the file is produced, you can move it to a different location for safekeeping:

Output

```
Success: Exported to 'wordpress-2021-01-25-25618e7.sql'.
```

You can also import a SQL dump file into your database through the `wp db import` command. This is useful when you are migrating a WordPress website from one location to another.

```
$ wp db import file.sql
```

Copy

Searching and Replacing

Another common operation you can perform with WP-CLI is a find-and-replace operation. You can make a dry run first to find out how many instances it would modify. The first string is the search component while the second is the replacement:

```
$ wp search-replace --dry-run 'example.com' 'example.net'
```

Copy

After running this, your output would be similar to the following:

Output

```
Success: 10 replacements to be made.
```

Once you are sure you want to proceed, remove the `--dry-run` flag from the previous command:

```
$ wp search-replace 'example.com' 'example.net'
```

Copy

In this step, you've reviewed several database operations that you can perform using WP-CLI. You can also complete other operations, such as optimizing the database, viewing database tables, deleting a database, or resetting one. You can explore the other options under the `wp db` subcommand by typing `wp help db` in your terminal.

Step 6 – Updating WordPress

You can update the core WordPress file with WP-CLI. You can examine the current version of WordPress that you have installed by running:

```
wp core version
```

Output

5.6

You can check for updates through the `wp core check-update` command. If your version is not the latest, this will produce an output similar to the following:

Output

version	update_type	package_url
5.6.1	minor	https://downloads.wordpress.org/release/wordpress-5.6.1-partial-0.zip

If an update is available, you can install it with:

```
$ wp core update
```

Copy

Dev OPS Another

Agenda

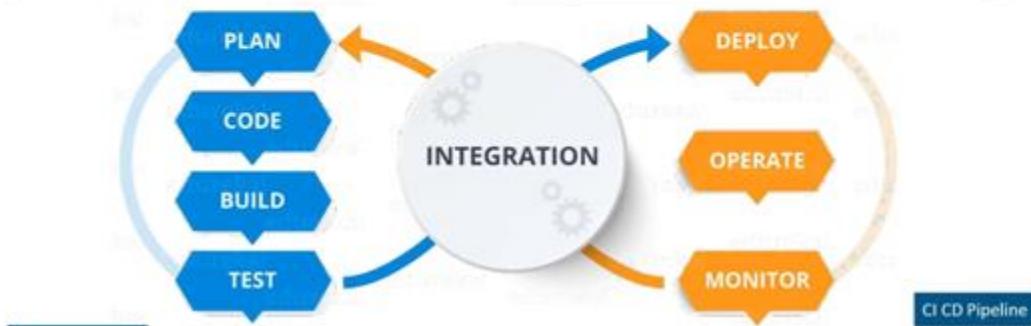


edureka!

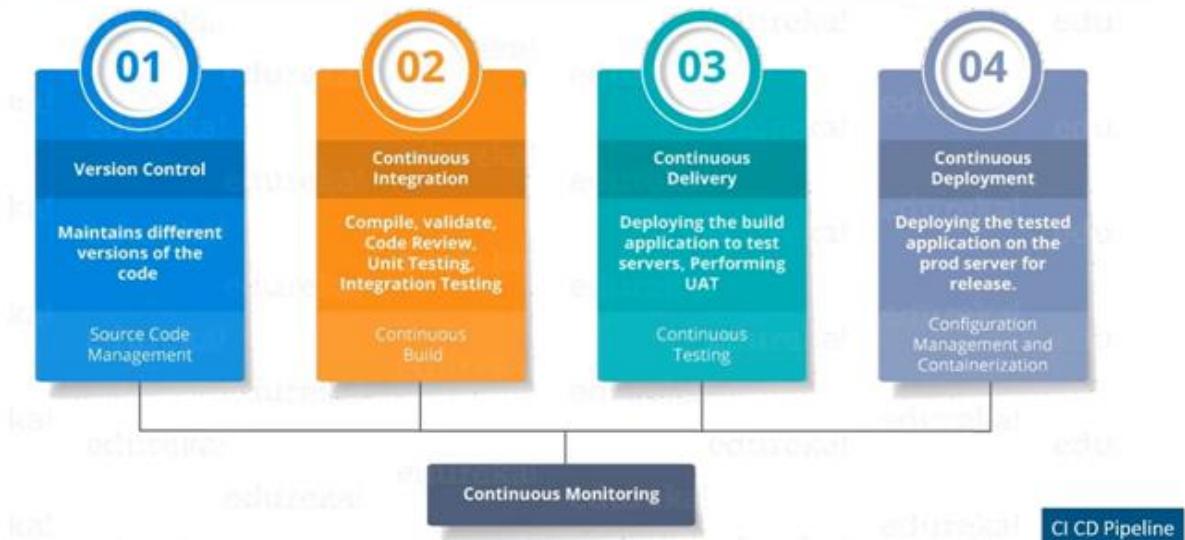
Looking for DevOps Online Training? Call us at IN: 9606058406 / US: 18338555775 or visit www.edureka.co

What is DevOps?

DevOps is a software development approach which involves Continuous Development, Continuous testing, Continuous Integration, Continuous Deployment and Continuous Monitoring throughout its development lifecycle.



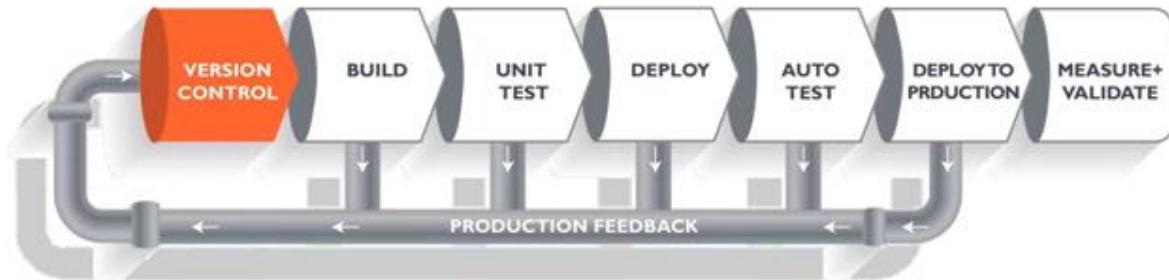
DevOps Stages



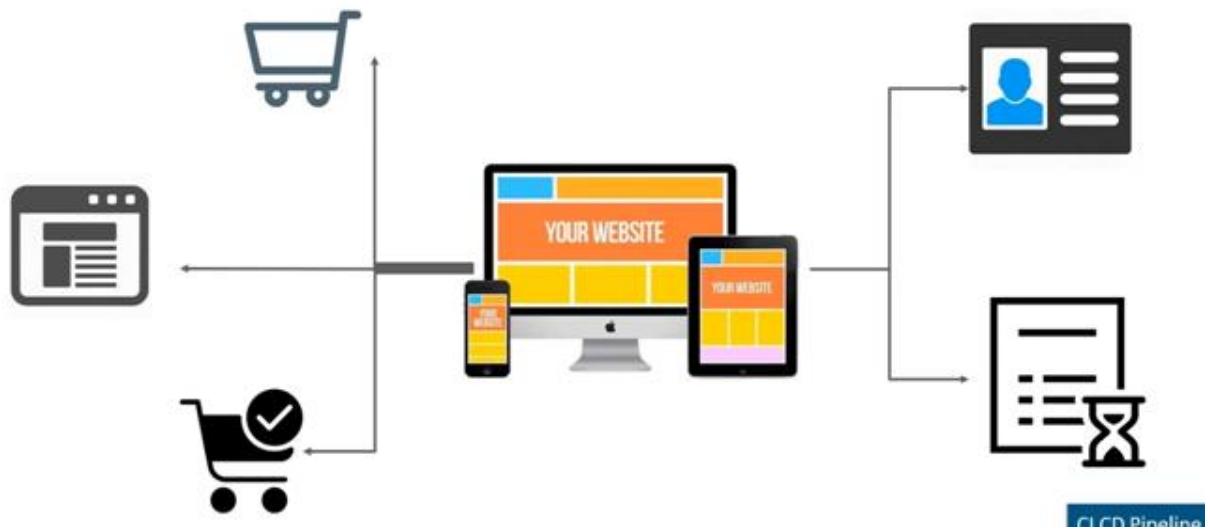
Continuous Integration



What is CI and CD?



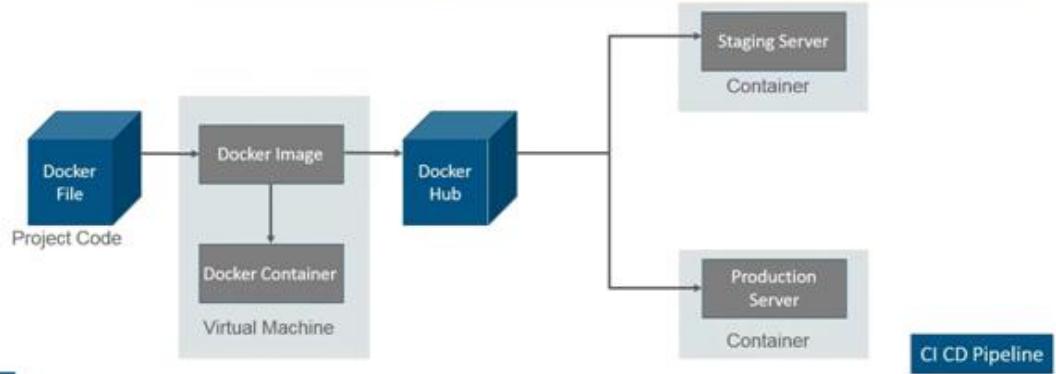
Continuous Integration Example



What is Docker?



- ✓ Docker file builds a Docker image and that image contains all the project's code
- ✓ You can run that image to create as many docker containers as you want
- ✓ Then this Image can be uploaded on Docker hub, from Docker hub any one can pull the image and build a container





Amazon Elastic Compute
Cloud (Amazon EC2)

- EC2 stands for **Elastic Compute Cloud**
- Used to create **virtual machines** in the AWS cloud
- Can be used as a server for **hosting websites** and other server management features such as **storage, ports, and security**.



Amazon Relational Database
Service (Amazon RDS)

- Used to create **dedicated database instances**
- Makes **designing infrastructure** more user friendly
- Can **support multiple database engines** like MySQL, SQL server, PostgreSQL and many more



Amazon Simple Storage
Service (Amazon S3)

- It offers a **highly secure** and **redundant** file storage service.
- Amazon S3 also offers **integrations** to help prevent data breaches (PCI-DSS, HIPAA)
- You get **data flexibility** without almost **zero latency**.