Custom end point

Refrences Video

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

wp-learn-form-submissions-api.php

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

```php
<?php
/**
 * Plugin Name: WP Learn Form Submissions API
 * Version: 0.0.1
 */

register_activation_hook( __FILE__, 'wp_learn_setup_table' );

function wp_learn_setup_table() {
    global $wpdb;
    $table_name = $wpdb->prefix . 'form_submissions';

    $sql = "CREATE TABLE $table_name (
      id mediumint(9) NOT NULL AUTO_INCREMENT,
      name varchar (100) NOT NULL,
      email varchar (100) NOT NULL,
      PRIMARY KEY  (id)
    )";

    require_once( ABSPATH . 'wp-admin/includes/upgrade.php' );
    dbDelta( $sql );
}


add_action( 'rest_api_init', 'wp_learn_register_routes' );
function wp_learn_register_routes() {
    register_rest_route(
        'wp-learn-form-submissions-api/v1',
        '/form-submissions',
        array(
            'methods' => 'GET',
            'callback' => 'wp_learn_rest_get_form_submissions',
            'permission_callback' => '__return_true'
        )
    );
}

```

```php
    register_rest_route(
        'wp-learn-form-submissions-api/v1',
        '/form-submission/(?P<id>[\d]+)',
        array(
            'methods'             => 'GET',
            'callback'            => 'wp_learn_rest_get_form_submission',
            'permission_callback' => '__return_true',
        )
    );

    register_rest_route(
        'wp-learn-form-submissions-api/v1',
        '/form-submissions',
        array(
            'methods'             => 'POST',
            'callback'            => 'wp_learn_rest_create_form_submissions',
            'permission_callback' => '__return_true',
        )
    );


}


function wp_learn_rest_get_form_submissions() {
    global $wpdb;
    $table_name = $wpdb->prefix . 'form_submissions';

    $results = $wpdb->get_results( "SELECT * FROM $table_name" );
    return $results;
}

function wp_learn_rest_get_form_submission( $request ) {
    $id = $request['id'];
    global $wpdb;
    $table_name = $wpdb->prefix . 'form_submissions';

    $results = $wpdb->get_results( "SELECT * FROM $table_name WHERE id = $id" );
    return $results[0];
}

```

```php

function wp_learn_rest_create_form_submissions( $request ) {
    global $wpdb;
    $table_name = $wpdb->prefix . 'form_submissions';

    $rows = $wpdb->insert(
        $table_name,
        array(
            'name'  => $request['name'],
            'email' => $request['email'],
        )
    );
    return 'Form submission added';
}
```

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:

Copy

```
$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!'
);
```

Now we need to set up the rest of the values that will be passed to the second parameter of `wp_remote_post()`

```php
$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

```php
$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:

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 to 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.

```php
$url =
'https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=danielpataki&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

```php
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:

```php
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

```php
function delete_data_with_basic_auth_from_api() {
    $api_url = 'https://api.example.com/delete'; // Replace this with your A
    $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

```php
function delete_data_with_jwt_from_api() {
    $api_url = 'https://api.example.com/delete'; // Replace this with your A
    $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

```php
function update_data_with_basic_auth_to_api() {
    $api_url = 'https://api.example.com/update'; // Replace this with your A
    $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
    }
}
```

Reger

```php
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 requir
    },
    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);
    });
```

plugin in WordPress to add custom fields for storing image URL

```php
<?php
/*
Plugin Name: Image Custom Fields Plugin
Description: Adds custom fields for storing image URLs.

// Function to add custom field for image URL
function add_custom_image_field() {
    // Replace 'post' with the post type where you want to add the custom f
    $post_type = 'post';

    // Adding the custom field for image URL
    add_meta_box(
        'custom_image_field',
        'Custom Image Field',
        'display_custom_image_field',
        $post_type,
        'normal',
        'default'
    );
}
```

```php
// Function to display the custom field input
function display_custom_image_field($post) {
    $value = get_post_meta($post->ID, 'custom_image_field', true);
    echo '<input type="text" id="custom_image_field" name="custom_image_fie
}

// Function to save the custom field value
function save_custom_image_field($post_id) {
    if (array_key_exists('custom_image_field', $_POST)) {
        update_post_meta(
            $post_id,
            'custom_image_field',
            sanitize_text_field($_POST['custom_image_field'])
        );
    }
}

// Hooks to execute the functions at appropriate times
add_action('add_meta_boxes', 'add_custom_image_field');
add_action('save_post', 'save_custom_image_field');
```

```
stom_image_field', true);
mage_field" name="custom_image_field" value="' . esc_attr($value) . '" />';
```

Plugin Name: File & Checkbox Custom Fields Plugin

Description: Adds custom fields for file upload and multiple checkboxes.

```php
// Function to add custom fields for file upload and multiple checkboxes
function add_custom_file_checkbox_fields() {
    // Replace 'post' with the post type where you want to add the custom fields.
    $post_type = 'post';

    // Adding the custom field for file upload
    add_meta_box(
        'custom_file_field',
        'Custom File Field',
        'display_custom_file_field',
        $post_type,
        'normal',
        'default'
    );

    // Adding the custom field for multiple checkboxes
    add_meta_box(
        'custom_checkbox_field',
        'Custom Checkbox Field',
        'display_custom_checkbox_field',
        $post_type,
        'normal',
        'default'
    );
}

// Function to display the file upload custom field
function display_custom_file_field($post) {
    $value = get_post_meta($post->ID, 'custom_file_field', true);
    echo '<input type="file" id="custom_file_field" name="custom_file_field" />';
}

// Function to display the multiple checkboxes custom field
function display_custom_checkbox_field($post) {
    $value = get_post_meta($post->ID, 'custom_checkbox_field', true);
    $options = array('Option 1', 'Option 2', 'Option 3'); // Add your checkbox
options
    foreach ($options as $option) {
        $checked = in_array($option, $value) ? 'checked' : '';
```

```
        echo '<label><input type="checkbox" name="custom_checkbox_field[]"
value="' . esc_attr($option) . '" ' . $checked . ' />' . esc_html($option) .
'</label><br />';
    }
}

// Function to save the custom field values
function save_custom_file_checkbox_fields($post_id) {
    if (array_key_exists('custom_file_field', $_FILES)) {
        // Handle file upload here
        // Example: move_uploaded_file($_FILES['custom_file_field']['tmp_name'],
$upload_dir . '/' . $_FILES['custom_file_field']['name']);
    }

    if (array_key_exists('custom_checkbox_field', $_POST)) {
        update_post_meta($post_id, 'custom_checkbox_field',
$_POST['custom_checkbox_field']);
    }
}

// Hooks to execute the functions at appropriate times
add_action('add_meta_boxes', 'add_custom_file_checkbox_fields');
add_action('save_post', 'save_custom_file_checkbox_fields');
```

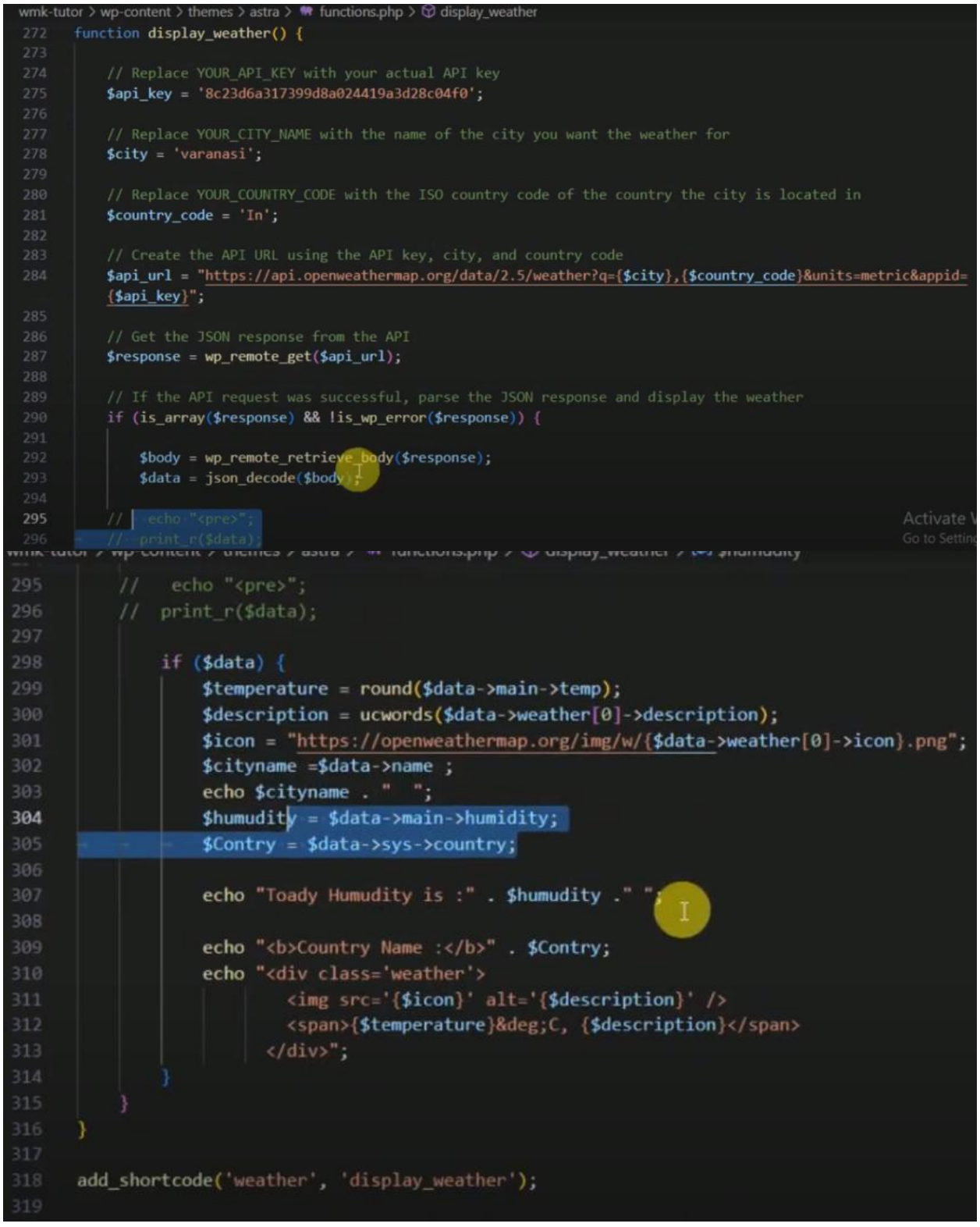
```php
// Function to display the select dropdown for gender
function display_custom_gender_field($post) {
    $value = get_post_meta($post->ID, 'custom_gender_field', true);
    $options = array('Male', 'Female'); // Options for gender select
    echo '<select id="custom_gender_field" name="custom_gender_field">';
    foreach ($options as $option) {
        $selected = ($value === $option) ? 'selected' : '';
        echo '<option value="' . esc_attr($option) . '" ' . $selected . '>'
    }
    echo '</select>';
}

// Function to save the custom field value
function save_custom_gender_field($post_id) {
    if (array_key_exists('custom_gender_field', $_POST)) {
        update_post_meta($post_id, 'custom_gender_field', $_POST['custom_ge
    }
}

// Hooks to execute the functions at appropriate times
add_action('add_meta_boxes', 'add_custom_gender_field');
add_action('save_post', 'save_custom_gender_field');
```

```php
c_attr($option) . '" ' . $selected . '>' . esc_html($option) . '</option>';
```

```php
nder_field', $_POST)) {
'custom_gender_field', $_POST['custom_gender_field']);
```

Wordpress Calling REST API

https://www.youtube.com/watch?v=c8kBOWPKpyU



JWT Autherntication.

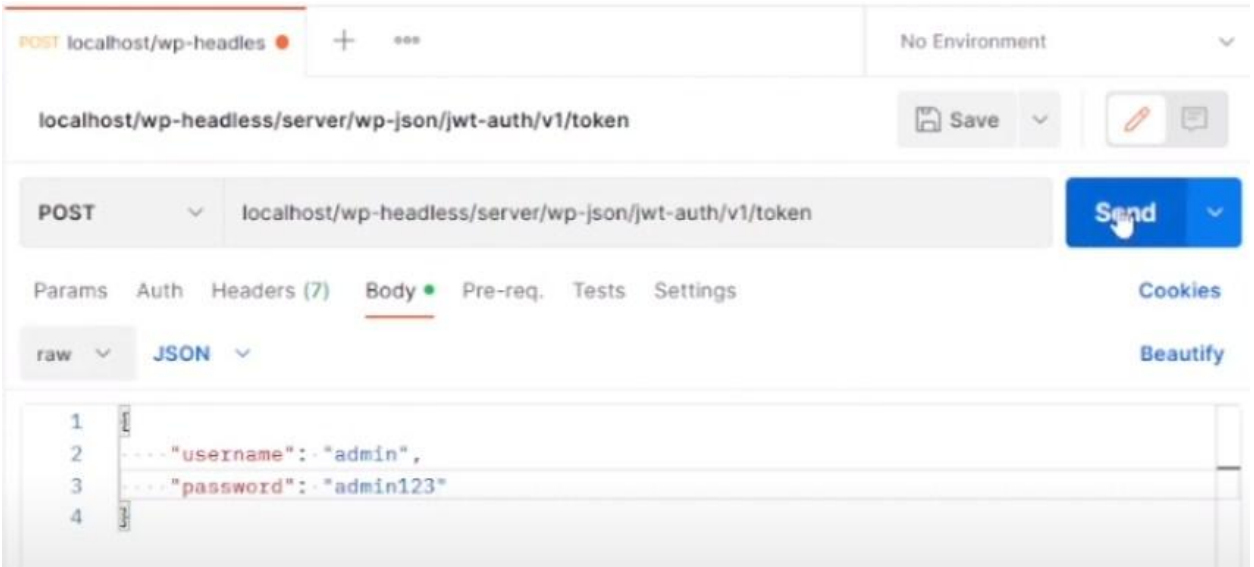
First install JWT Autherntication plugin for REST API.

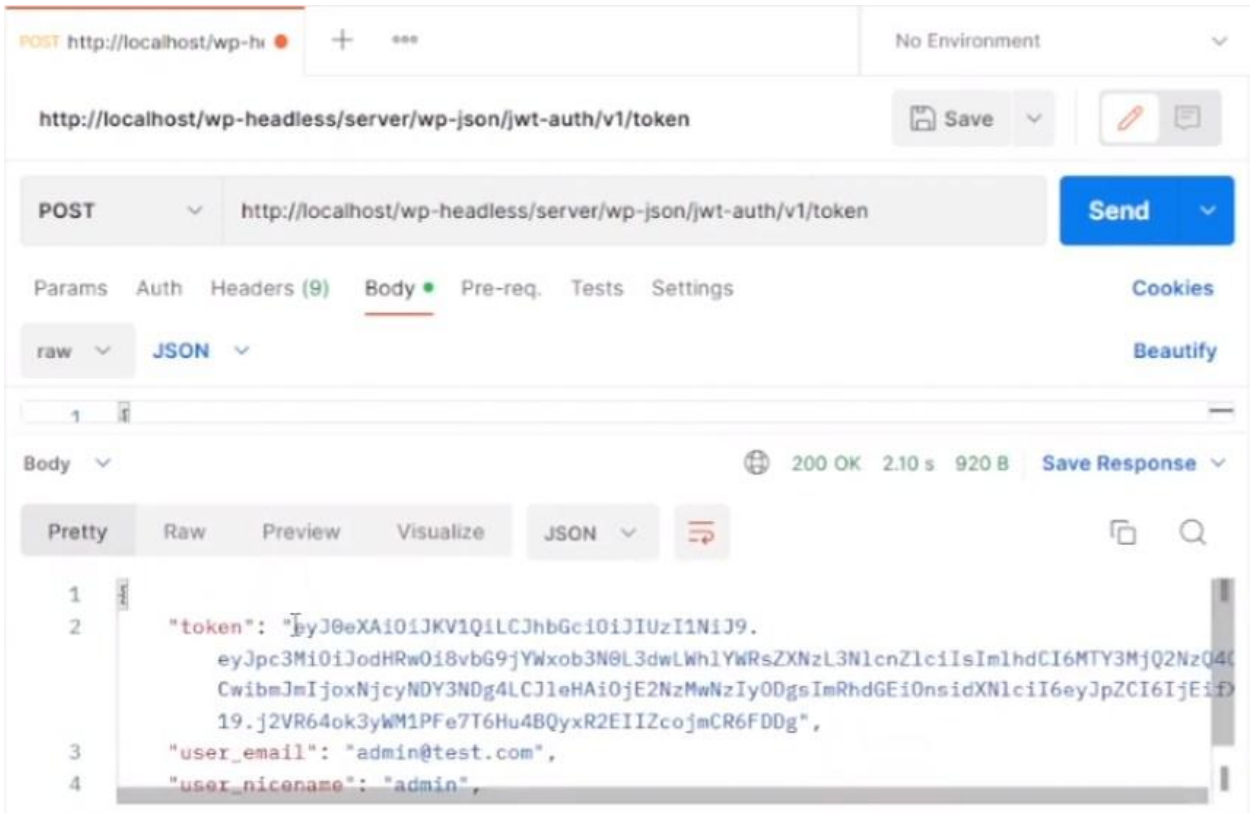
Second Configure plugin

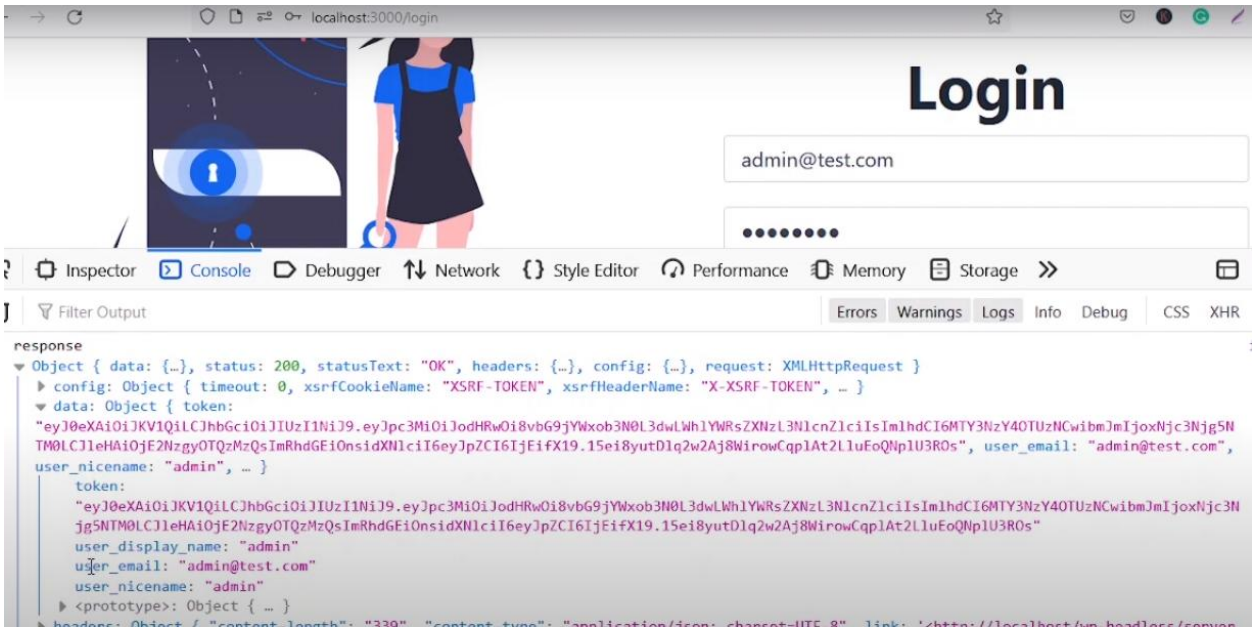
wp-config.php

```
 * 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$A./EnoBkPIL/5SX#j_C' );
define( 'LOGGED_IN_KEY',    'vy_2h&RenpRC;l&!y#vZq)[-$EFe@OXPIpv3}fQrX^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=$0zi/<!wrxz:+.}3>KGBd+6<_k+TV`7-' );
define( 'SECURE_AUTH_SALT', 'd(+49W/x8Lp.Hr4]LSz+M`9?p6;gl`y)kI6/6H(L8JbfE)8oom=BgYomTXQ[UQf%' );
define( 'LOGGED_IN_SALT',   'EbUstW(X`1}`A29qHzv=f0{eVIo(lo(l`r?VC1VmB2fK2<XZ%8>`K_f3~dF/?*JT' );
define( 'NONCE_SALT',       '?ZV!ia(!jzi5A6V6GBR%Hx2Ua=N(5z 6*ZuaF,6Ws$liWna<o(pKVbY8/m[MlX1n' );

define('JWT_AUTH_SECRET_KEY', '-[<bE4_wI-%m*KBcKxA5C<:cI2*a,&*44CSd[c-PO>6&6qj)OJ3P,E5}Y=|Oi|gq');
define('JWT_AUTH_CORS_ENABLE', true);

/**#@-*/
```

Token Generation through Postman API Tool




POST http://localhost/wp-h
No Environment
http://localhost/wp-headless/server/wp-json/jwt-auth/v1/token
Save
POST
http://localhost/wp-headless/server/wp-json/jwt-auth/v1/token
Send
Params Auth Headers (9) Body Pre-req. Tests Settings
Cookies
raw JSON
Beautify
Body
200 OK 2.10 s 920 B
Save Response
Pretty Raw Preview Visualize JSON
"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0L3dwLWhlYWRsZXNzL3NlcnZlciIsImlhdCI6MTY3MjQ2NzQ4
CwibmJmIjoxNjcyNDY3NDg4LCJleHAiOjE2NzMwNzIyODgsImRhdGEiOnsidXNlciI6eyJpZCI6IjEif
19.j2VR64ok3yWM1PFe7T6Hu4BQyxR2EIIZcojmCR6FDDg",
"user_email": "admin@test.com",
"user_nicename": "admin",


localhost:3000/login
Login
admin@test.com
Inspector Console Debugger Network Style Editor Performance Memory Storage
Filter Output
Errors Warnings Logs Info Debug CSS XHR
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.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0L3dwLWhlYWRsZXNzL3NlcnZlciIsImlhdCI6MTY3NzY4OTUzNCwibmJmIjoxNjc3Njg5NTM0LCJleHAiOjE2NzgyOTQzMzQsImRhdGEiOnsidXNlciI6eyJpZCI6IjEifX19.15ei8yutDlq2w2Aj8WirowCqplAt2LluEoQNplU3ROs", user_email: "admin@test.com",
user_nicename: "admin", … }
token:
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0L3dwLWhlYWRsZXNzL3NlcnZlciIsImlhdCI6MTY3NzY4OTUzNCwibmJmIjoxNjc3Njg5NTM0LCJleHAiOjE2NzgyOTQzMzQsImRhdGEiOnsidXNlciI6eyJpZCI6IjEifX19.15ei8yutDlq2w2Aj8WirowCqplAt2LluEoQNplU3ROs"
user_display_name: "admin"
user_email: "admin@test.com"
user_nicename: "admin"
<prototype>: Object { … }

How to Add Custom Endpoints to WordPress API

https://artisansweb.net/category/wordpress/
https://artisansweb.net/how-to-add-custom-endpoints-to-wordpress-api/

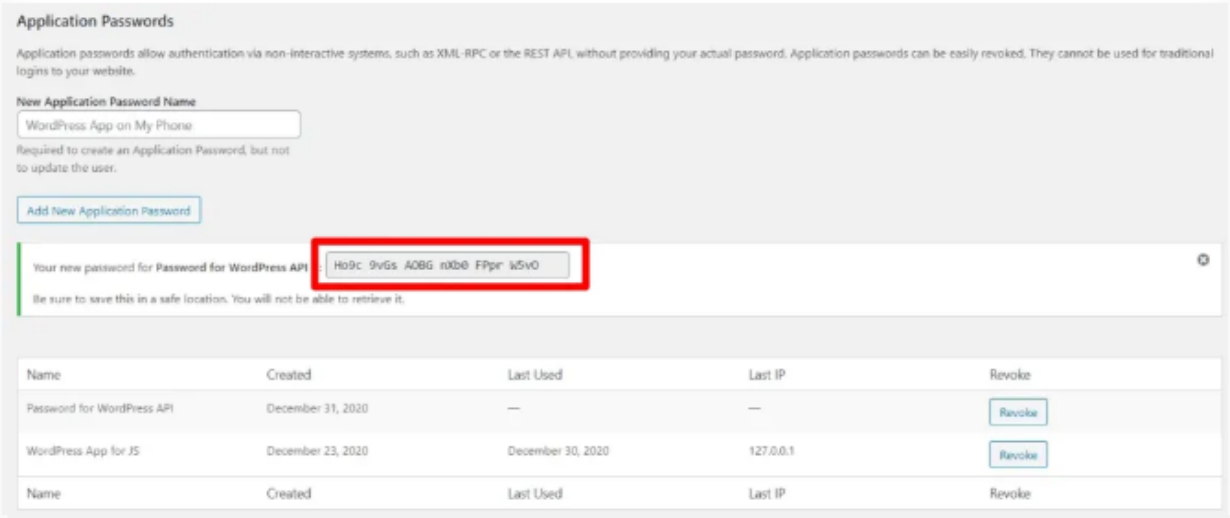
Authorization Using WordPress REST API

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.



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 nXb0 FPpr W5vO'. Use the following statement which gives you a final valid token.

```php
<?php
$username = 'admin';
$application_password = 'Ho9c 9vGs AOBG nXb0 FPpr W5vO';

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.

```php
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.

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
<?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.',
            ];
        }

        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, $use
            // 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.

```php
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=-30MKlNDHrA&t=27s

```php
<?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'])[1]);
        $file = wp_upload_bits($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,
        'starus' => 201,
    ];
}

add_action('rest_api_init', function () {
```

```php
    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/

```php
// 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

```php
<?php
include('db.php');
$sql="select * from user";
$res=mysqli_query($con,$sql);
$count=mysqli_num_rows($res);
header('Content-Type:application/json');

if($count>0){
    while($row=mysqli_fetch_assoc($res)){
        $arr[]=$row;
    }
    echo json_encode(['status'=>'true','data'=>$arr,'result'=>'found']);
}else{
    echo json_encode(['status'=>'true','data'=>'No data found','result'=>'not']);
}
?>
```

index.php

```php
<?php
$url="https://programmingwithvishal.com/api/";
$ch=curl_init();
curl_setopt($ch, CURLOPT_URL,$url);
curl_setopt($ch,CURLOPT_RETURNTRANSFER,true);
$result=curl_exec($ch);
curl_close($ch);
$result=json_decode($result,true);
if(isset($result['status'])){
    if($result['status']==true){
        if(isset($result['result'])){
            if($result['result']==true){
                ?>
                <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

```php
<?php
include('db.php');
header('Content-Type:application/json');
if(isset($_GET['key'])){
    $key=mysqli_real_escape_string($con,$_GET['key']);
    $checkRes=mysqli_query($con,"select * from api_token where token
        ='$key'");
    if(mysqli_num_rows($checkRes)>0){
        $checkRow=mysqli_fetch_assoc($checkRes);
        if($checkRow['status']==1){
            if($checkRow['hit_count']>=$checkRow['hit_limit']){
                echo json_encode(['status'=>'false','data'=>'API hit limit
                    exceed']);
                die();
            }else{
                mysqli_query($con,"update api_token set hit_count=hit_count
                    +1 where token='$key'");
            }


            $sql="select * from user";
            $res=mysqli_query($con,$sql);
            $count=mysqli_num_rows($res);


            if($count>0){
                while($row=mysqli_fetch_assoc($res)){
                    $arr[]=$row;
                }
                echo json_encode(['status'=>'true','data'=>$arr,'result'
                    =>'found']);
            }else{
                echo json_encode(['status'=>'true','data'=>'No data found'
                    ,'result'=>'not']);
            }
        }else{
            echo json_encode(['status'=>'false','data'=>'API key
                deactivated']);
        }
    }else{
            echo json_encode(['status'=>'false','data'=>'Please provide
                valid api key']);
    }
}else{
    echo json_encode(['status'=>'false','data'=>'Please provide api key']);
}
?>
```

+ Options

	id	token	hit_limit	hit_count	status
Edit Copy Delete	1	ahdhsasdnasbwdldkgldk	5	5	1

index.php

```
1
2   s://programmingwithvishal.com/api/index.php?key=ahdhsasdnasbwdldkgldk";
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
Edit Copy Delete	1	ahdhsasdnasbwdldkgldk	5	5	1

Check all *With selected:* Edit Copy Delete Export

user

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

	id	name	email
Edit Copy Delete	2	Amit Gupta	amit@gmail.com
Edit Copy Delete	3	Bhaavit	bhaavit@gmail.com

API token.php

```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 tok
    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';
    }
```

API insert.php

```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,$_POST['email']);
        mysqli_query($con,"insert into user(name,email) values('$name',
        $status='true';
        $data="Data inserted";
        $code='8';
    }else{
        $status='true';
        $data="Provide valid column count";
        $code='9';
    }
}
echo json_encode(["status"=>$status,"data"=>$data,"code"=>$code]);
?>
```

API delete.php

```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]);
```

Consumer delete.php

delete.php

```php
if(isset($_GET['id']) && $_GET['id']>0){
    $url="https://programmingwithvishal.com/api/delete.php?token=ahdhsa
    $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['c
        header('location:index.php');
        die();
    }else{
        echo $result['data'];
    }
}else{
    header('location:index.php');
    die();
}
```

API db.php

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

API token.php

```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
<?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
<?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
<?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 Respone Code

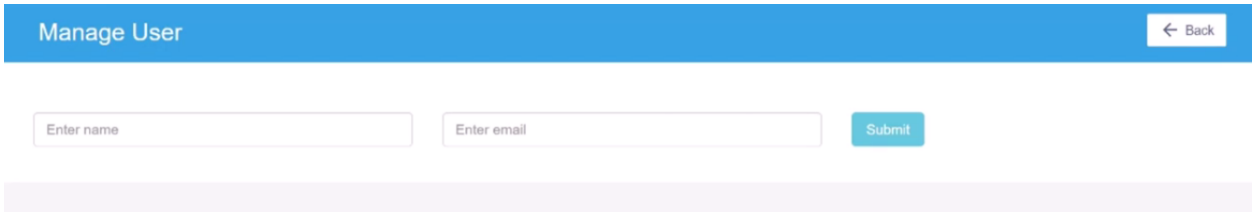
1=>Please provide API token
2=>Please provide valid API token
3=>API token deactivated
4=>Data not found
5=>Data Return
6=>Provide id
7=>Data deleted
8=>Data Inserted
9=>Provide valid column count
10=>Data updated

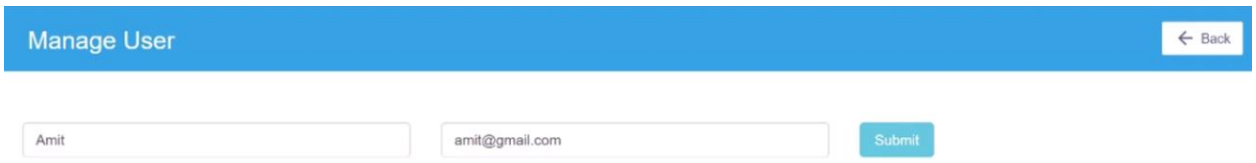
Consumer form.php

```php
<?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'];
    }
    $url="api_url/manage_form.php?token=ahdhsasdnasbwdldkgldk&id=".$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){
    $url="https://programmingwithvishal.com/api/index.php?token=ahdhsasdnasbw
dldkgldk&id=".$_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();
    }
}


?>
```

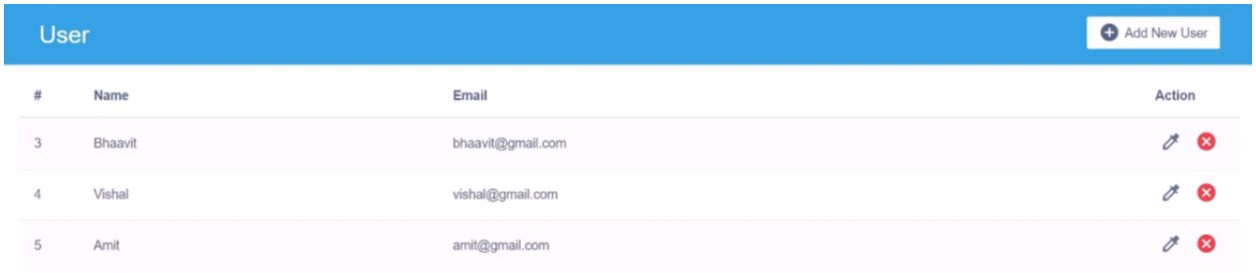
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>API Crud Operation</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scri
pt>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></s
cript>
    <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 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>
```





Consumer index.php



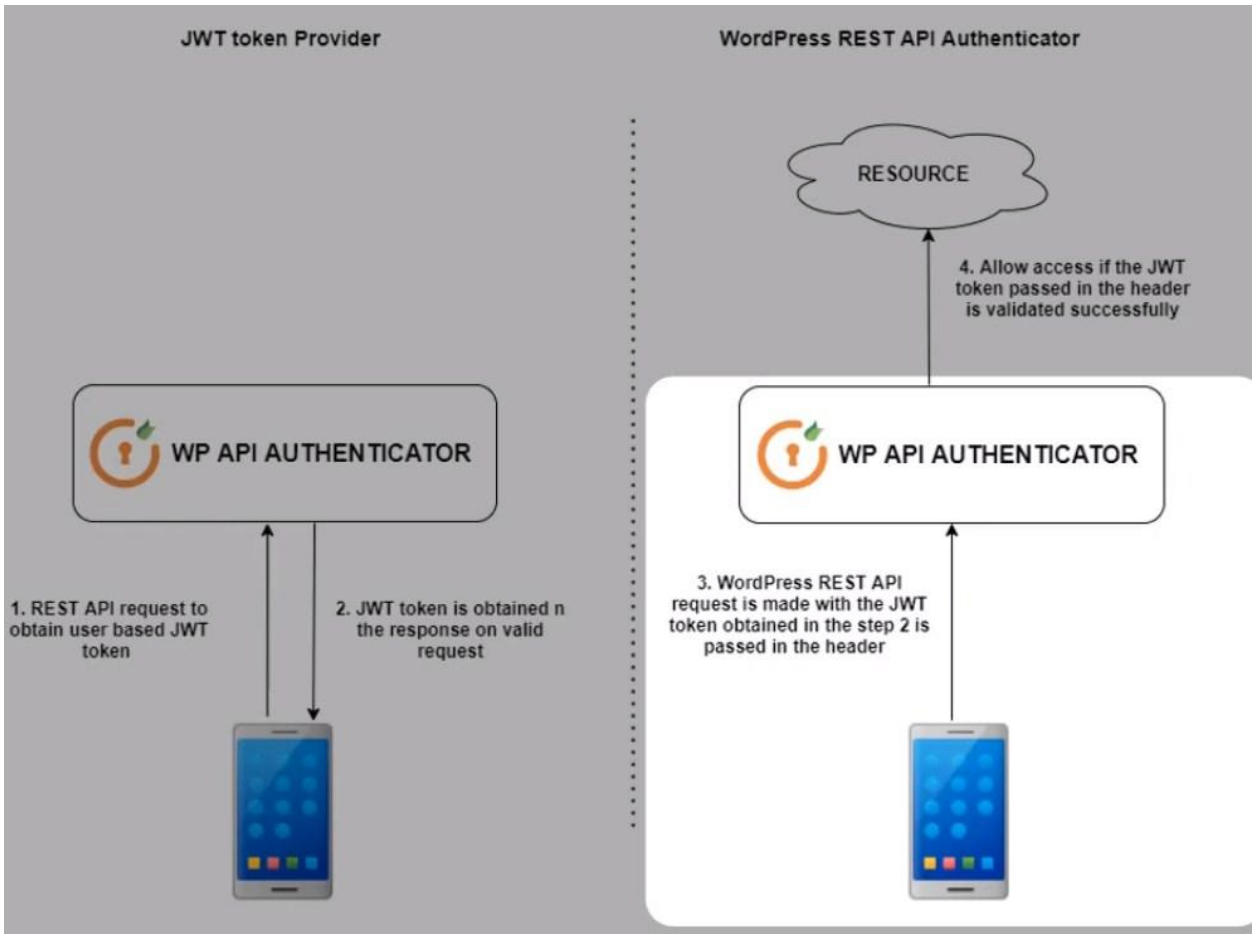

```php
<?php
$url="api_url/index.php?token=ahdhsasdnasbwdldkgldk";
$ch=curl_init();
curl_setopt($ch, CURLOPT_URL,$url);
curl_setopt($ch,CURLOPT_RETURNTRANSFER,true);
$result=curl_exec($ch);
curl_close($ch);
$result=json_decode($result,true);
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <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"></scri
pt>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></s
cript>
    <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">&#xE147;</i> <span>Add New
User</span></a>
                    </div>
                </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']?></a></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">&#xe3b8;</i></a>
                                <a href="delete.php?id=<?php echo
$list['id']?>" class="delete" title="Delete" ><i class="material-
icons">&#xE5C9;</i></a>
                                </td>
                            </tr>
```

```
                            <?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=ahdhsasdnasbwdldkgldk";
    $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:

```php
$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 . ':' . $passw
    ),
) );

// Process $response as needed
```

```php
'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:

```php
$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:

```php
$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 respor

$response = curl_exec($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:

```php
$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

```php
$response = wp_remote_get('https://your-wordpress-site.com/wp-json/wp/v2/pos

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

```javascript
$.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 po
        });
    },
    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.

```php
$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, publi
);

$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

```php
$post_data = array(
    'title' => 'New Post Title',
    'content' => 'Content of the new post.',
    'status' => 'publish' // Set the status as needed (draft, pending, publi
);

$response = wp_remote_post(
    'https://your-wordpress-site.com/wp-json/wp/v2/posts',
    array(
        'headers' => array(
            'Content-Type' => 'application/json',
        ),
        'body' => wp_json_encode($post_data),
    )
);

if ( is_wp_error( $response ) ) {
    echo 'Error occurred: ' . $response->get_error_message();
} else {
    $new_post = json_decode( wp_remote_retrieve_body( $response ), true );
    // Handle the newly created post
    print_r($new_post);
}
```

To add a new post in WordPress using WordPress functions and methods, you can use the `wp_insert_post` function. Here's an example:

php Copy code

```php
// Create a new post object
$new_post = array(
    'post_title'    => 'New Post Title',
    'post_content'  => 'Content of the new post.',
    'post_status'   => 'publish',
    'post_author'   => 1, // Replace with the author's user ID
    'post_type'     => 'post', // Replace with the post type you want
);

// Insert the post into the database
$post_id = wp_insert_post($new_post);

if (is_wp_error($post_id)) {
    echo 'Error occurred: ' . $post_id->get_error_message();
} else {
    echo 'New post added with ID: ' . $post_id;
}
```

Another way

```php
$token = 'YOUR_TOKEN_HERE'; // The token for authentication
$url = 'https://target-site.com/wp-json/wp/v2/posts'; // URL of the target \

$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

```php
$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

```php
$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

```php
$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($ch, CURLOPT_CUSTOMREQUEST, "POST"); // or "PUT" depending on y
curl_setopt($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
     }
});
```

Rege

How to fetch all user of wordpress in through rest api by js

javascript Copy code

```javascript
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

```javascript
$.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

```php
$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

```php
$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 to

        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
                // Handle the response after creating the user
            },
            error: function(error) {
                $('#responseMessage').text('Error creating user: ' + error.r
                // Handle errors during user creation
            }
        });
```

Reg

```
unction(response) {
sponseMessage').text('User created successfully. User ID: ' + response.id);
```

how to create Wordpress custom registration form

Custom Registration Form HTML:

```
<form id="custom-registration-form" action="<?php echo esc_url( admin_url('admin-post.php') ); ?>" method="post">
    <input type="text" name="username" placeholder="Username">
    <input type="email" name="email" placeholder="Email">
    <input type="password" name="password" placeholder="Password">
    <input type="text" name="mobile" placeholder="Mobile Number">
    <input type="text" name="address" placeholder="Address">
    <input type="date" name="dob" placeholder="Date of Birth">
    <input type="submit" name="submit" value="Register">
    <input type="hidden" name="action" value="custom_register">
    <?php wp_nonce_field( 'custom-register-nonce', 'custom-register-nonce' ); ?>
</form>
```

Form Processing in functions.php:

```
function custom_registration() {
    if (isset($_POST['submit'])) {
        if( wp_verify_nonce( $_POST['custom-register-nonce'], 'custom-register-nonce' ) ) {
            $username = sanitize_user( $_POST['username'] );
            $email = sanitize_email( $_POST['email'] );
            $password = $_POST['password'];
            $mobile = sanitize_text_field( $_POST['mobile'] );
            $address = sanitize_text_field( $_POST['address'] );
            $dob = sanitize_text_field( $_POST['dob'] );

            $userdata = array(
                'user_login' => $username,
                'user_email' => $email,
                'user_pass' => $password,
                'user_url' => $mobile, // Storing mobile number in the user's URL field
            );

            $user_id = wp_insert_user( $userdata );

            if ( is_wp_error($user_id) ) {
                echo $user_id->get_error_message();
            } else {
```

```
                // Additional meta fields for user data
                update_user_meta($user_id, 'user_address', $address);
                update_user_meta($user_id, 'user_dob', $dob);

                echo 'Registration successful!';
            }
        }
    }
}
add_action('admin_post_custom_register', 'custom_registration');
add_action('admin_post_nopriv_custom_register', 'custom_registration');
```

how to display custom this field on user profile page

```
// Function to add custom fields to user profile
function custom_user_profile_fields($user) {
    ?>
    <h3>Custom Profile Fields</h3>
    <table class="form-table">
        <tr>
            <th><label for="mobile">Mobile Number</label></th>
            <td><input type="text" name="mobile" id="mobile" value="<?php echo
esc_attr(get_the_author_meta('user_url', $user->ID)); ?>" class="regular-
text"></td>
        </tr>
        <tr>
            <th><label for="address">Address</label></th>
            <td><input type="text" name="address" id="address" value="<?php echo
esc_attr(get_user_meta($user->ID, 'user_address', true)); ?>" class="regular-
text"></td>
        </tr>
        <tr>
            <th><label for="dob">Date of Birth</label></th>
            <td><input type="date" name="dob" id="dob" value="<?php echo
esc_attr(get_user_meta($user->ID, 'user_dob', true)); ?>" class="regular-
text"></td>
        </tr>
    </table>
    <?php
}
add_action('show_user_profile', 'custom_user_profile_fields');
add_action('edit_user_profile', 'custom_user_profile_fields');
```

```php
// Function to save custom fields on user profile update
function save_custom_user_profile_fields($user_id) {
    if (!current_user_can('edit_user', $user_id)) {
        return false;
    }

    if (isset($_POST['mobile'])) {
        update_user_meta($user_id, 'user_url', $_POST['mobile']); // Save mobile
to user_url field
    }
    if (isset($_POST['address'])) {
        update_user_meta($user_id, 'user_address', $_POST['address']); // Save
address to user meta
    }
    if (isset($_POST['dob'])) {
        update_user_meta($user_id, 'user_dob', $_POST['dob']); // Save date of
birth to user meta
    }
}
add_action('edit_user_profile_update', 'save_custom_user_profile_fields');
```

Jquery

How to get field data from form by jquery

```html
<form id="frontEndPostForm">
    <input type="text" name="title" placeholder="Title"><br>
    <textarea name="content" placeholder="Content"></textarea><br>
    <button type="submit" id="submitPostBtn">Submit Post</button>
</form>
<div id="responseMessage"></div>
```

```
$(document).ready(function() {
    $('#frontEndPostForm').submit(function(event) {
        event.preventDefault();

        var newPost = {
            title: $('#frontEndPostForm input[name="title"]').val(),
            content: $('#frontEndPostForm textarea[name="content"]').val(),
            status: 'publish'
        };

        $.ajax({
            type: 'POST',
            url: 'https://your-wordpress-site.com/wp-json/wp/v2/posts',
            data: JSON.stringify(newPost),
            contentType: 'application/json',
            success: function(response) {
                $('#responseMessage').text('Post submitted successfully. Pos
                // Handle the response after post submission
            },
            error: function(error) {
```

how to submit a form with field name, email, mobile, message through with submit button by jquery

```html
<form id="contactForm">
    <input type="text" name="name" placeholder="Name"><br>
    <input type="email" name="email" placeholder="Email"><br>
    <input type="tel" name="mobile" placeholder="Mobile"><br>
    <textarea name="message" placeholder="Message"></textarea><br>
    <button type="submit">Submit</button>
</form>
```

```javascript
$(document).ready(function() {
    $('#contactForm').submit(function(event) {
        event.preventDefault(); // Prevent the default form submission

        var formData = $(this).serialize(); // Serialize form data

        $.ajax({
            type: 'POST',
            url: 'YOUR_SERVER_ENDPOINT_URL', // Replace with your server en
            data: formData,
            success: function(response) {
                console.log('Form submitted successfully:', response);
                // Handle the response after form submission
            },
            error: function(error) {
                console.log('Error submitting form:', error);
                // Handle errors during form submission
            }
        });
    });
});
```

how to submit a form with field name, email, mobile, message through with submit button by jquery but without reloading page

```html
<form id="contactForm">
    <input type="text" name="name" placeholder="Name"><br>
    <input type="email" name="email" placeholder="Email"><br>
    <input type="tel" name="mobile" placeholder="Mobile"><br>
    <textarea name="message" placeholder="Message"></textarea><br>
    <button type="submit" id="submitBtn">Submit</button>
</form>
<div id="responseMessage"></div>
```

```javascript
ument).ready(function() {
'#contactForm').submit(function(event) {
  event.preventDefault(); // Prevent the default form submission

  var formData = $(this).serialize(); // Serialize form data

  $.ajax({
      type: 'POST',
      url: 'YOUR_SERVER_ENDPOINT_URL', // Replace with your server endpoint
      data: formData,
      success: function(response) {
          $('#responseMessage').text('Form submitted successfully: ' + respon
          // Handle the response after form submission
      },
      error: function(error) {
          $('#responseMessage').text('Error submitting form: ' + error);
          // Handle errors during form submission
      }
  });
);
```

Form validation

```
$(document).ready(function() {
    $('#contactForm').submit(function(event) {
        event.preventDefault(); // Prevent the default form submission

        // Basic form validation
        var name = $('#name').val();
        var email = $('#email').val();
        var mobile = $('#mobile').val();
        var message = $('#message').val();

        // Regular expressions for email and mobile number validation
        var emailRegex = /^\S+@\S+\.\S+$/;
        var mobileRegex = /^\d{10}$/;

        if (name === '' || message === '') {
            alert('Please fill in name and message fields.');
            return;
        }

        if (!emailRegex.test(email)) {
            alert('Please enter a valid email address.');
            return;
        }

        if (!mobileRegex.test(mobile)) {
            alert('Please enter a valid 10-digit mobile number.');
            return;
        }

        var formData = $(this).serialize(); // Serialize form data

        $.ajax({
            type: 'POST',
            url: 'YOUR_SERVER_ENDPOINT_URL', // Replace with your server enc
```

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 provides 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":"1
8"}
```

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
<?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":"1
8"}
```

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
<?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
<?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'));
```

```php
// 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
<?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
<?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.

```php
// 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

```php
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');
```

Ln 670, Col 33 Tab Size: 4 UTF-8 LF PHP Go Live

functions.php custom.js page-load-posts.php

wp-content > themes > twentytwentyone > js > custom.js > jQuery() callback > on('click', '.loadmore') callback > $.post() callback

```js
var page = 2;
jQuery(function($) {
    $('body').on('click', '.loadmore', function() {
        var data = {
            'action': 'load_posts_by_ajax',
            'page': page,
            'security': blog.security
        };

        $.post(blog.ajaxurl, data, function(response) {
            if($.trim(response) != '') {
                $('.blog-posts').append(response);
            } else {
                $('.loadmore').hide();
            }
        });
    });
});
```

functions.php page-load-posts.php

wp-content > themes > twentytwentyone > page-load-posts.php

```php
<?php
get_header();

/* Start the Loop */
while ( have_posts() ) :
    the_post();
    ?>
    <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>

        <header class="entry-header alignwide">
            <?php the_title( '<h1 class="entry-title">', '</h1>' ); ?>
        </header>
        <div class="entry-content">
            <?php
            $args = array(
                'post_type' => 'post',
                'post_status' => 'publish',
                'posts_per_page' => '2',
                'paged' => 1,
            );

            $blog_posts = new WP_Query( $args );
            ?>
            <?php if ( $blog_posts->have_posts() ) : ?>
                <div class="blog-posts">
                    <?php while ( $blog_posts->have_posts() ) : $blog_posts->the_post(); ?>
                        <h2><?php the_title(); ?></h2>
                        <?php the_excerpt(); ?>
                    <?php endwhile; ?>
                    <?php wp_reset_postdata(); ?>
                </div>
                <div class="loadmore">Load More...</div>
            <?php endif; ?>
        </div><!-- .entry-content -->

    </article><!-- #post-<?php the_ID(); ?> -->

    <?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-lkGo&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

```php
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();
} ?>
```

```html
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript" src="/wordpress/js/jquery-1.11.2.min.js"></script>
        <script>
            function fetch(){
                $.post('/wordpress/wp-admin/admin-ajax.php', {'action':'my_action'}, function(response){
                    $('#datafetch').append(response);
                });
            }
        </script>
    </head>
    <body>
        <div id="datafetch">
            <button onclick="fetch()">Click Here</button>
        </div>
    </body>
</html>
```

WP Ajax call Visshal 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
    }
});
```

```php
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']
    ]);
```

```php
    if($result>0){
        wp_send_json_success("Data inserted");
    }else{
        wp_send_json_error("Please try again");
    }
```

Load more post by AJAX Call

```php
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');
```

Ln 670, Col 33 Tab Size: 4 UTF-8 LF PHP Go Live

functions.php custom.js page-load-posts.php

wp-content > themes > twentytwentyone > js > custom.js > jQuery() callback > on('click', '.loadmore') callback > $.post() callback

```js
var page = 2;
jQuery(function($) {
    $('body').on('click', '.loadmore', function() {
        var data = {
            'action': 'load_posts_by_ajax',
            'page': page,
            'security': blog.security
        };

        $.post(blog.ajaxurl, data, function(response) {
            if($.trim(response) != '') {
                $('.blog-posts').append(response);
            } else {
                $('.loadmore').hide();
            }
        });
    });
});
```

functions.php page-load-posts.php

wp-content > themes > twentytwentyone > page-load-posts.php

```php
<?php
get_header();

/* Start the Loop */
while ( have_posts() ) :
    the_post();
    ?>
    <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>

        <header class="entry-header alignwide">
            <?php the_title( '<h1 class="entry-title">', '</h1>' ); ?>
        </header>
        <div class="entry-content">
            <?php
            $args = array(
                'post_type' => 'post',
                'post_status' => 'publish',
                'posts_per_page' => '2',
                'paged' => 1,
            );

            $blog_posts = new WP_Query( $args );
            ?>
            <?php if ( $blog_posts->have_posts() ) : ?>
                <div class="blog-posts">
                    <?php while ( $blog_posts->have_posts() ) : $blog_posts->the_post(); ?>
                        <h2><?php the_title(); ?></h2>
                        <?php the_excerpt(); ?>
                    <?php endwhile; ?>
                    <?php wp_reset_postdata(); ?>
                </div>
                <div class="loadmore">Load More...</div>
            <?php endif; ?>
        </div><!-- .entry-content -->

    </article><!-- #post-<?php the_ID(); ?> -->

    <?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-lkGo&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

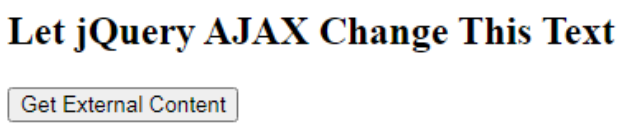
Back on the frontend, you can then handle receiving this data back.

```
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

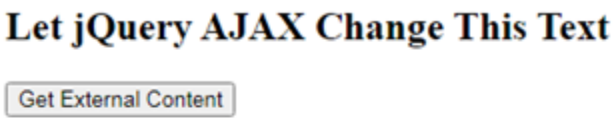
```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
        <script>
        $(document).ready(function(){
          $("button").click(function(){
            $("#div1").load("demo_test.txt");
          });
        });
        </script>
</head>
<body>
        <div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>
        <button>Get External Content</button>
</body>
```



After clicking result

Method :2

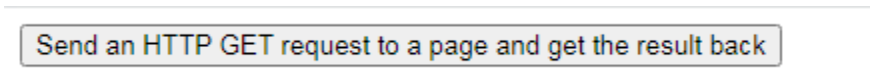
```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>
        <script>
                $(document).ready(function(){
                  $("button").click(function(){
                    $.ajax({url: "demo_test.txt", success: function(result){
                      $("#div1").html(result);
                    }});
                  });
                });
        </script>
</head>
<body>
        <div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>
        <button>Get External Content</button>
</body>
```



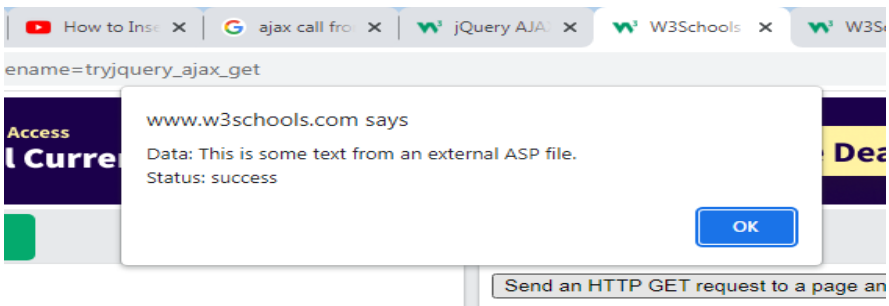
After clicking result

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>
```



Result



Method	Description
$.ajax()	Performs an async AJAX request
$.ajaxPrefilter()	Handle custom Ajax options or modify existing options before each request is sent and before they are processed by $.ajax()
$.ajaxSetup()	Sets the default values for future AJAX requests
$.ajaxTransport()	Creates an object that handles the actual transmission of Ajax data
$.get()	Loads data from a server using an AJAX HTTP GET request
$.getJSON()	Loads JSON-encoded data from a server using a HTTP GET request
$.parseJSON()	Deprecated in version 3.0, use JSON.parse() instead. Takes a well-formed JSON string and returns the resulting JavaScript value
$.getScript()	Loads (and executes) a JavaScript from a server using an AJAX HTTP GET request
$.param()	Creates a serialized representation of an array or object (can be used as URL query string for AJAX requests)
$.post()	Loads data from a server using an AJAX HTTP POST request
ajaxComplete()	Specifies a function to run when the AJAX request completes
ajaxError()	Specifies a function to run when the AJAX request completes with an error
ajaxSend()	Specifies a function to run before the AJAX request is sent
ajaxStart()	Specifies a function to run when the first AJAX request begins
ajaxStop()	Specifies a function to run when all AJAX requests have completed
ajaxSuccess()	Specifies a function to run when an AJAX request completes successfully
load()	Loads data from a server and puts the returned data into the selected element
serialize()	Encodes a set of form elements as a string for submission
serializeArray()	Encodes a set of form elements as an array of names and values

Ajax with PHP

index.php get.php

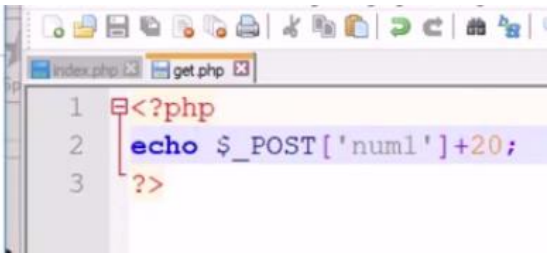
```
1  <script src="https://code.jquery.com/jquery-2.2.4.min.js"
   ></script>
2
3  <a href="javascript:void(0)" onclick="click_here()">Click Here
   </a>
4
5  <script>
6  function click_here(){
7      jQuery.ajax({
8          url:'get.php',
9          type:'post',
10         success:function(result){
11             alert(result);
12         }
13     });
14 }
15 </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"
   ></script>
2  <input type="textbox" id="num1"/>
3  <a href="javascript:void(0)" onclick="click_here()">Click Here
   </a>
4
5  <script>
6  function click_here(){
7      var num1=jQuery('#num1').val();
8      jQuery.ajax({
9          url:'get.php',
10         type:'post',
11         data:'num1='+num1,
12         success:function(result){
13             alert(result);
14         }
15     });
16 }
17 </script>
```



get.php

```
1  <?php
2  echo $_POST['num1']+20;
3  ?>
```

Syncronus AJAX- **Parallel, it is not waiting for other**

Asyncronus AJAX **it is depending on other, Its work one after one.**

index.php | get.php | types_ajax.php | get1.php | get2.php

```
<script src="https://code.jquery.com/jquery-2.2.4.min.js"
></script>
<script>
hit1();
hit2();
function hit1(){
    jQuery.ajax({
        url:'get1.php',
        type:'post',
        async:false,
        success:function(result){
            console.log(result);
        }
    });
}
function hit2(){
    jQuery.ajax({
        url:'get2.php',
        type:'post',
        success:function(result){
            console.log(result);
        }
```

index.php | get.php | types_ajax.php | get1.php | get2.php

```
<?php
sleep(5);
echo 'get1';
?>
```

index.php | get.php | types_ajax.php | get1.php | get2.php

```
<?php
echo 'get2';
?>
```

AJAX With JSON Dataserve API call

```
index.php  style.css
  <link rel="stylesheet" href="css/style.css">
</head>
<body>
  <div id="main">
    <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 : "https://jsonplaceholder.typicode.com/posts/50 ",
      type : "GET",
      success : function(data){
        $("#load-data").append(data.id + "<br>" + data.title + "<br>" + data.body);
        //console.log(data);
      }
    });
  });
</script>
</body>
</html>
```

For looping data

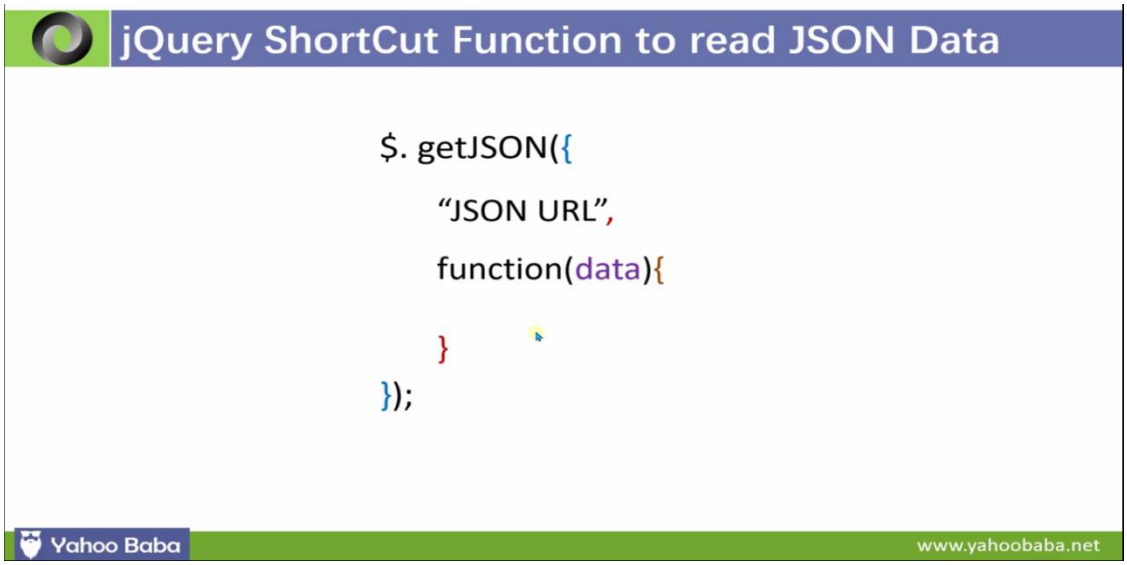
```
index.php  style.css
    <div id="header">
      <h1>Read JSON Data</h1>
    </div>

    <div id="load-data"></div>

  </div>

<script type="text/javascript" src="js/jquery.js"></script>
<script>
  $(document).ready(function(){
    $.ajax({
      url : "https://jsonplaceholder.typicode.com/posts ",
      type : "GET",
      success : function(data){
        //$("#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>
</html>
```

API call from Json file

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(){
    $.ajax({
      url : "json/my.json",
      type : "GET",
      success : function(data){
        //$("#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>
```

Sorcut method for AJAX call for Only JSON 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

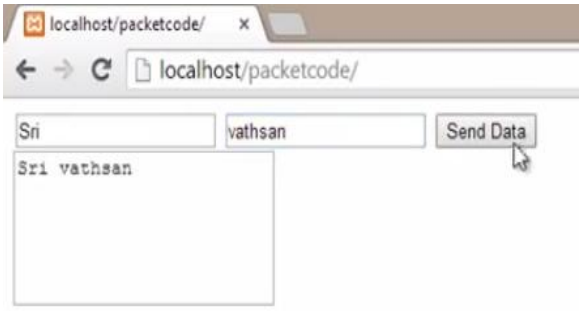
```
<script src = "http://code.jquery.com/jquery-1.11.1.min.js"></script>
<script src = "script.js"></script>

<div>
    <input type = "text" id = "fname">
    <input type = "text" id = "surname">
    <button id = "formsubmit">Send Data</button><br>
    <textarea id = "response" style = "width: 200px; height: 100px; resize:
    none;"></textarea>
</div>
```

script.js

```
$(document).ready(function(){

    $('#formsubmit').click(function(){

        $.post("submit.php",
            {fname: $('#fname').val(), surname: $('#surname').val()},
            function(data){
                $('#response').html(data);
            }
        );

    });

});
```

submit.php

```
<?php

$fname = $_POST['fname'];
$lname = $_POST['surname'];

echo "$fname $lname";

?>
```



index.html

```
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript">
   $.ajax({
      url : "https://api.covid19api.com/summary",
      type : "GET",
      dataType : "JSON",
      success : function(data){
         console.log(data);
         console.log(data.Global);

         $.each(data.Global, function(key, value){
            console.log(key + " : " + value);
         });
      }
   });
</script>
</body>
</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 | db.php

```
        </div>
        <script>
        jQuery('#contactForm').on('submit',function(e){
            jQuery('#btn').val('Please wait...');
            jQuery('#btn').attr('disabled',true);
            jQuery.ajax({
                url:'submit.php',
                type:'post',
                data:jQuery('#contactForm').serialize(),
                success:function(result){
                    jQuery('#thank_you_msg').html('Thank you');
                    jQuery('#contactForm')['0'].reset();
                    jQuery('#btn').val('Submit Now');
                    jQuery('#btn').attr('disabled',false);
                }
            });
            e.preventDefault();
        });
        </script>
    </body>
</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

ENG 1:16 AM 5/27/2019

index.html

```
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript">
   $.ajax({
      url : "https://api.covid19api.com/summary ",
      type : "GET",
      dataType : "JSON",
      success : function(data){
         console.log(data);

         console.log(data.Countries[101]);
         console.log(data.Countries[101].TotalConfirmed);

         $.each(data.Global, function(key, value){
            $("#global-wise").append("<tr><td bgcolor='yellow'>" + key +"</td><td>" + value +'
         });

         var sno = 1;
         $.each(data.Countries, function(key, value){
            $("#country-wise").append("<tr>" +
                                      "<td>" + sno +"</td>" +
                                      "<td>" + value.Country +"</td>" +
                                      "<td>" + value.NewConfirmed +"</td>" +
                                      "<td>" + value.NewDeaths +"</td>" +
                                      "<td>" + value.NewRecovered +"</td>" +
                                      "<td>" + value.TotalConfirmed +"</td>" +
```

FOLDERS

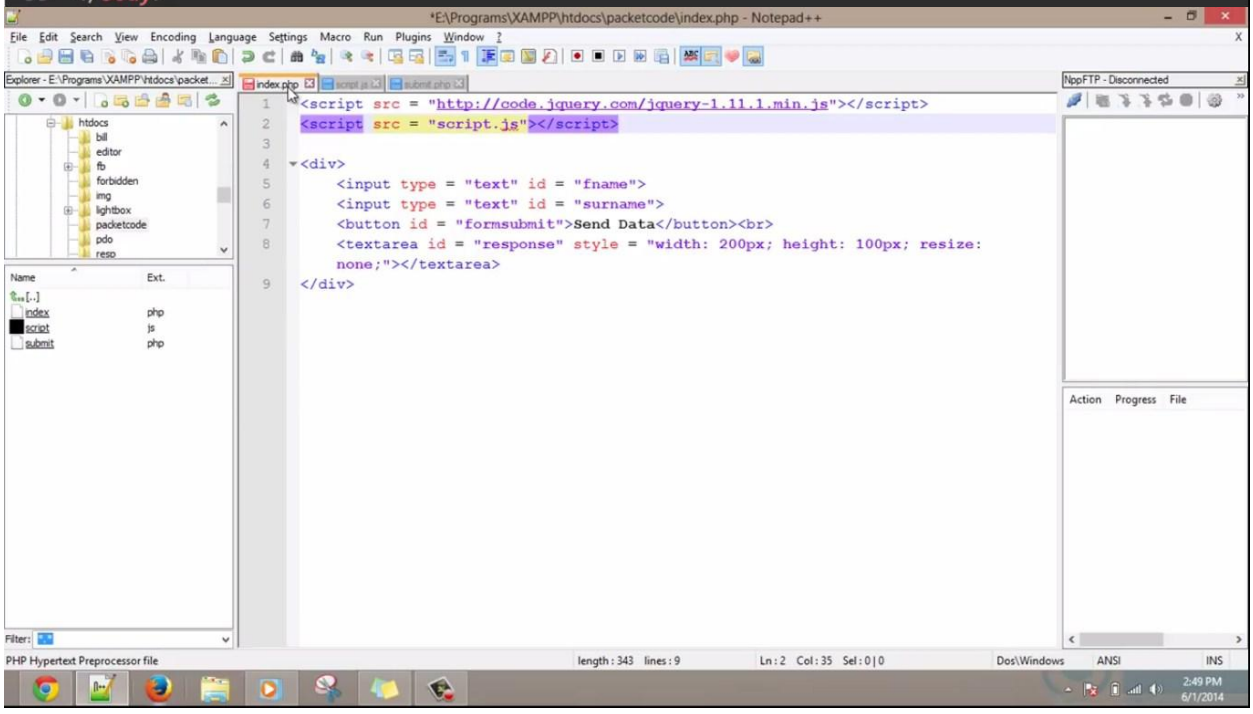
- json-data
  - css
  - js
  - json
  - index.php

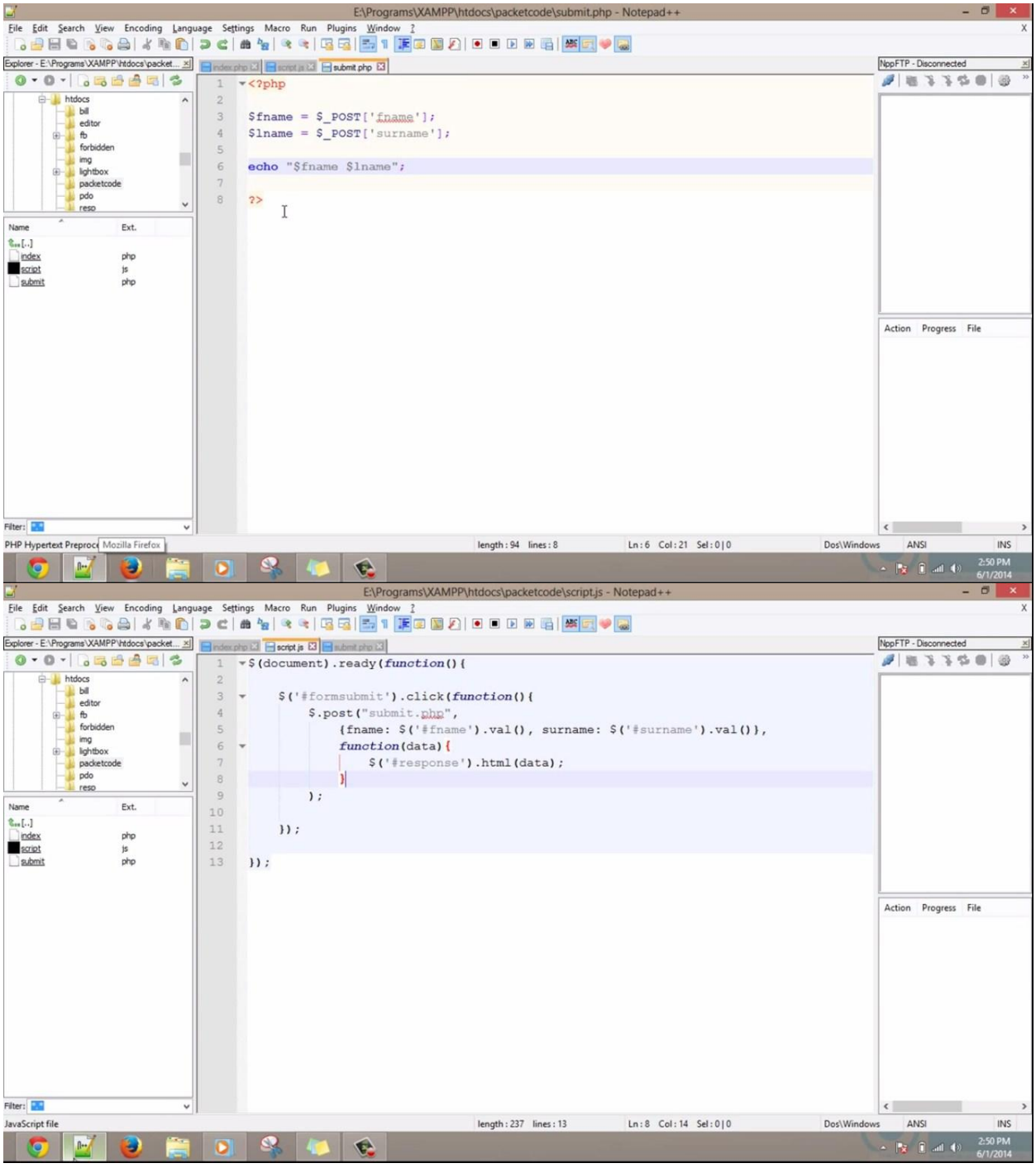
index.php | my.json | style.css

```
    <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(){
    $.ajax({
      url : "json/my.json",
      type : "GET",
      success : function(data){
        //$("#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>
```

index.php | my.json | 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      $.getJSON(
24        "json/my.json",
25        function(data){
26          //$("#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>
```

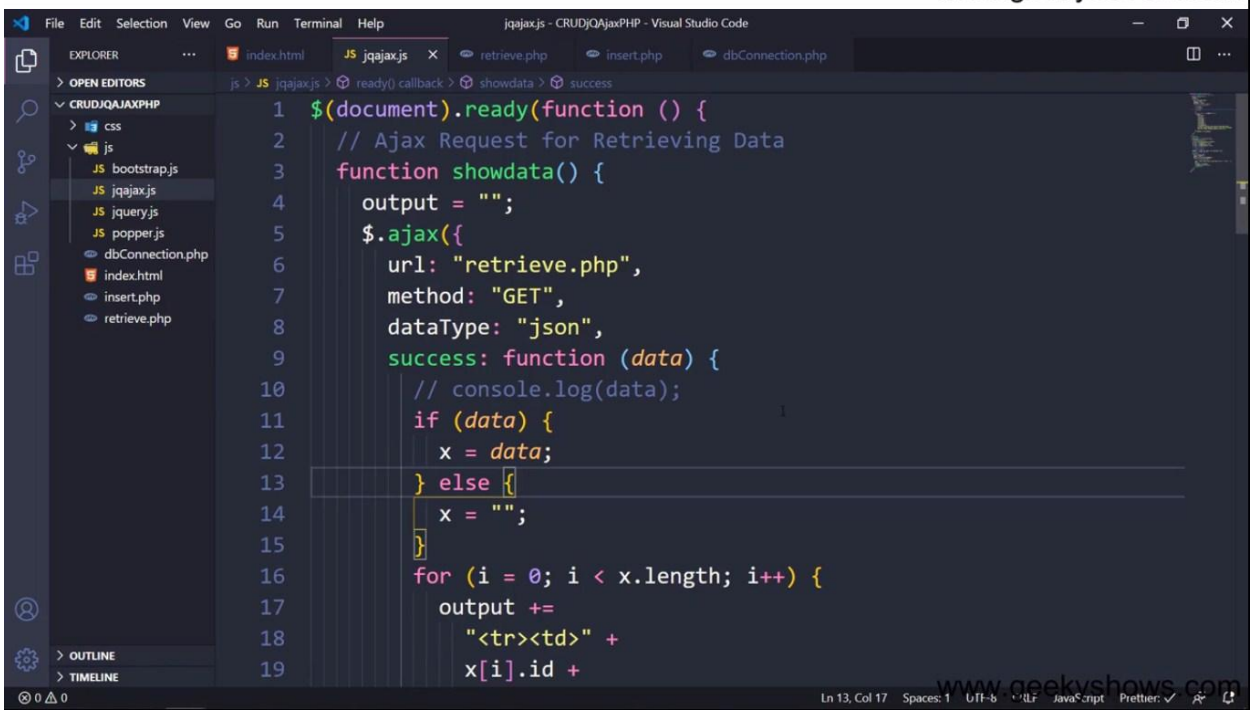


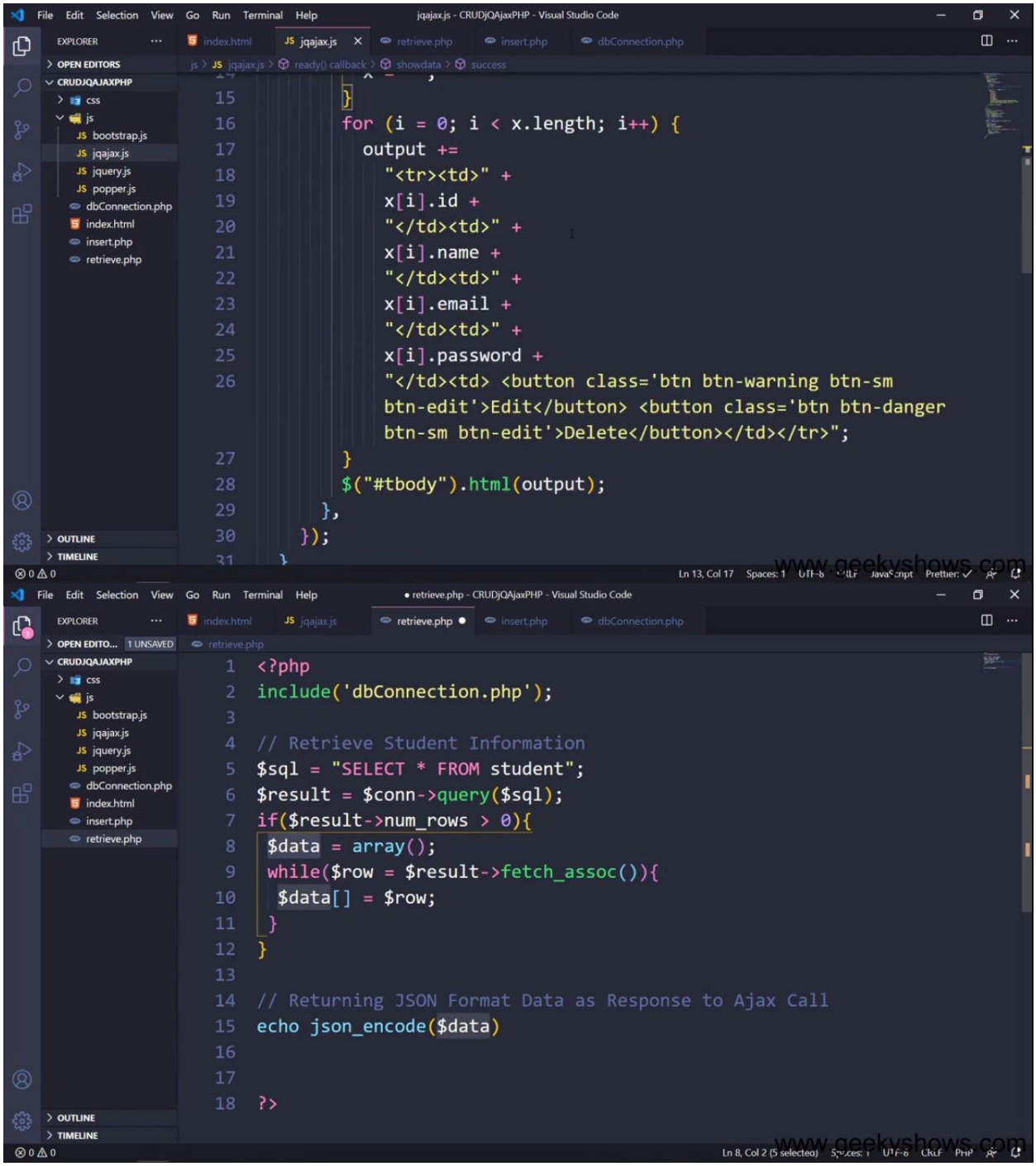

E:\Programs\XAMPP\htdocs\packetcode\submit.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Explorer - E:\Programs\XAMPP\htdocs\packet...
index.php
script.js
submit.php
NppFTP - Disconnected
htdocs
bill
editor
fb
forbidden
img
lightbox
packetcode
pdo
reso
Name
Ext.
index
php
script
js
submit
php
Filter:
Action
Progress
File
1 <?php
2
3 $fname = $_POST['fname'];
4 $lname = $_POST['surname'];
5
6 echo "$fname $lname";
7
8 ?>
PHP Hypertext Preproc
Mozilla Firefox
length : 94 lines : 8
Ln : 6 Col : 21 Sel : 0 | 0
Dos\Windows
ANSI
INS
2:50 PM
6/1/2014
E:\Programs\XAMPP\htdocs\packetcode\script.js - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Explorer - E:\Programs\XAMPP\htdocs\packet...
index.php
script.js
submit.php
NppFTP - Disconnected
htdocs
bill
editor
fb
forbidden
img
lightbox
packetcode
pdo
reso
Name
Ext.
index
php
script
js
submit
php
Filter:
Action
Progress
File
1 $(document).ready(function(){
2
3 $('#formsubmit').click(function(){
4 $.post("submit.php",
5 {fname: $('#fname').val(), surname: $('#surname').val()},
6 function(data){
7 $('#response').html(data);
8 }
9);
10
11 });
12
13 });
JavaScript file
length : 237 lines : 13
Ln : 8 Col : 14 Sel : 0 | 0
Dos\Windows
ANSI
INS
2:50 PM
6/1/2014

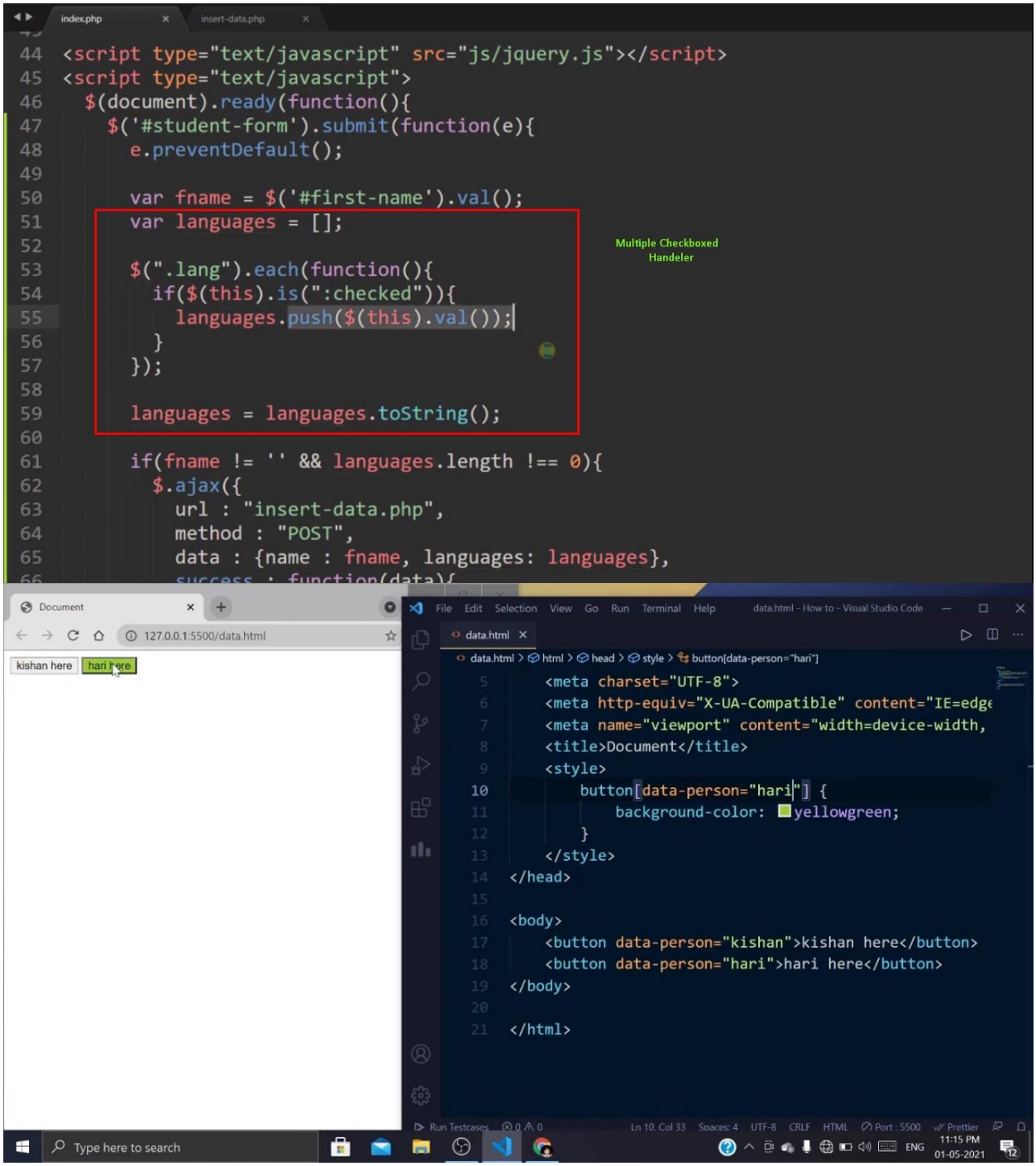
jQuery Ajax

Example:-

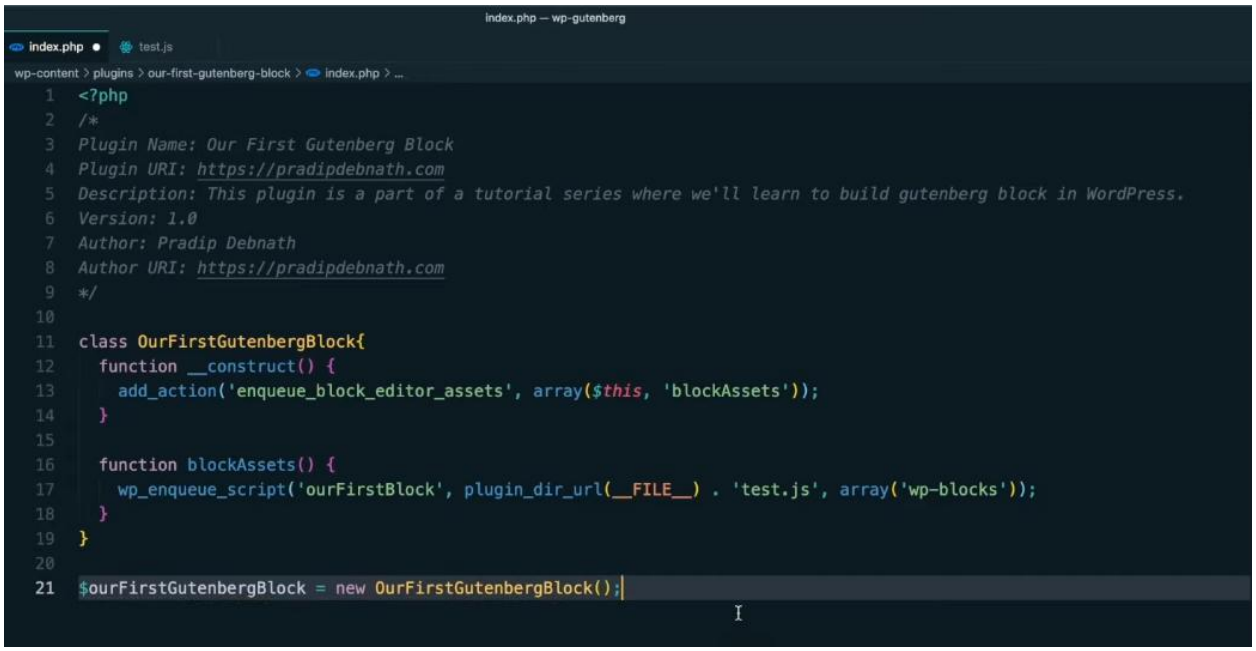
```
$(document).ready(function(){
        $("#btn").click(function(){
                $.ajax({
                        url: "insert.php",
                        method: "POST",
                        data: {"name": "sonam", "roll":101},
                        success: function(data){
                                // Process the Response Data
                        }
                })
        })
})
```


File Edit Selection View Go Run Terminal Help
jqajax.js - CRUDjQAjaxPHP - Visual Studio Code
EXPLORER
index.html
jqajax.js
retrieve.php
insert.php
dbConnection.php
OPEN EDITORS
CRUDJQAJAXPHP
css
js
bootstrap.js
jqajax.js
jquery.js
popper.js
dbConnection.php
index.html
insert.php
retrieve.php
OUTLINE
TIMELINE
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 btn-edit'>Edit</button> <button class='btn btn-danger btn-sm btn-edit'>Delete</button></td></tr>";
27 }
28 $("#tbody").html(output);
29 },
30 });
Ln 13, Col 17
www.geekyshows.com
File Edit Selection View Go Run Terminal Help
• retrieve.php - CRUDjQAjaxPHP - Visual Studio Code
EXPLORER
index.html
jqajax.js
retrieve.php •
insert.php
dbConnection.php
OPEN EDITO... 1 UNSAVED
retrieve.php
CRUDJQAJAXPHP
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 ?>
Ln 8, Col 2 (5 selected)
www.geekyshows.com


index.php
insert-data.php
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
$('#student-form').submit(function(e){
e.preventDefault();
var fname = $('#first-name').val();
var languages = [];
$(".lang").each(function(){
if($(this).is(":checked")){
languages.push($(this).val());
}
});
languages = languages.toString();
if(fname != '' && languages.length !== 0){
$.ajax({
url : "insert-data.php",
method : "POST",
data : {name : fname, languages: languages},
Multiple Checkboxed Handeler
Document
127.0.0.1:5500/data.html
kishan here
hari here
File Edit Selection View Go Run Terminal Help
data.html - How to - Visual Studio Code
data.html
data.html > html > head > style > button[data-person="hari"]
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge
<meta name="viewport" content="width=device-width,
<title>Document</title>
<style>
button[data-person="hari"] {
background-color: yellowgreen;
}
</style>
</head>
<body>
<button data-person="kishan">kishan here</button>
<button data-person="hari">hari here</button>
</body>
</html>
Run Testcases
Ln 10, Col 33
Spaces: 4
UTF-8
CRLF
HTML
Port : 5500
Prettier
Type here to search
ENG
11:15 PM
01-05-2021

Basic Gutenberg Block Development



Richtext Block

index.js .../basic-block | index.js .../rich-text-block | editor.css .../rich-text-block

```
import './editor.css';

const { RichText } = wp.editor;
const { registerBlockType } = wp.blocks;
const blockStyle = { backgroundColor: '#900', color: '#fff', padding: '20px' };

registerBlockType('my-block/rich-text-block', {
    title: 'My Rich Text Block',
    icon: 'welcome-write-blog',
    category: 'common',
    attributes: {
        content: {
            source: 'html',
            selector: '.text-content',
        }
    },
```

```
    edit: function( {className, attributes, setAttributes} ) {

        const { content } = attributes;

        function onChangeContent( newContent ) {
            setAttributes( {content: newContent} );
        }

        return (
            <RichText
                tagName="p"
                className= {className}
                onChange = { onChangeContent }
                value= {content}
            />
        );
    },
```

```
    save: function( {attributes, className} ) {

        const { content } = attributes;

        return (
            <RichText.Content
                tagName="p"
                className="text-content"
                value={content}
            />
        );
    }
});
```

```
import './block/block.js';
import './block/basic-block';
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
<?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

```javascript
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"
          />
```

```
        );
      }
      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
<?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
<?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 enqueuing 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:

```php
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:

```php
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 enqueuing 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:

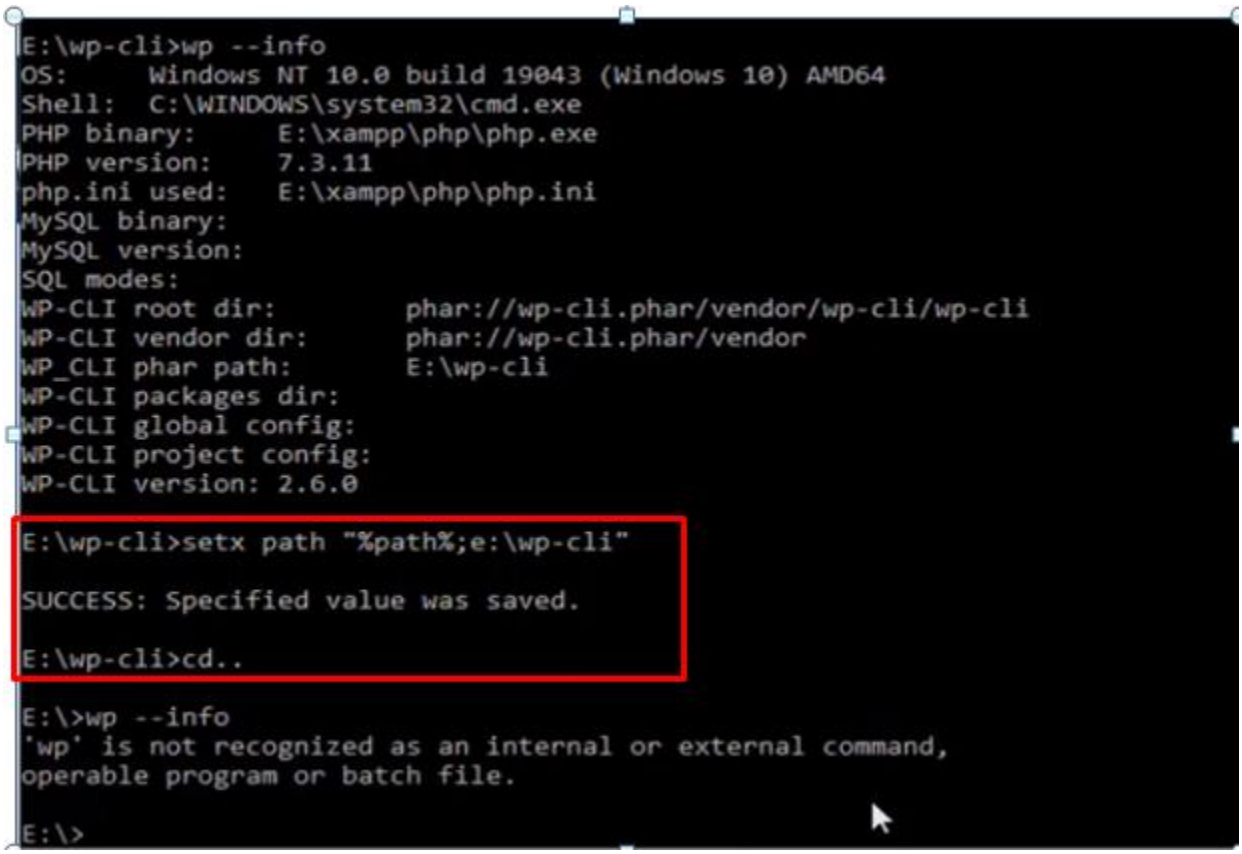
```
@ECHO OFF
php "c:/wp-cli/wp-cli.phar" %*
```

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:



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:

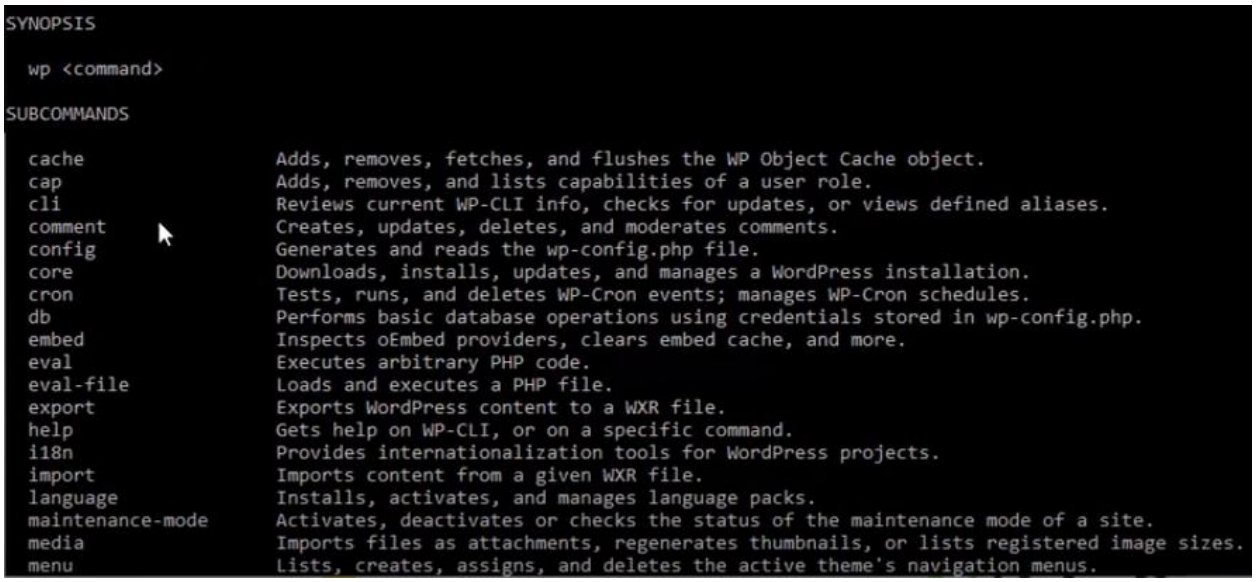
```
php wp-cli.phar --info
```

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



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 filesytem.
  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 tes
t 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:

1. **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.
2. **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.
3. **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.
4. **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.
5. **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.
6. **User Management**: WP-CLI makes it easy to create, update, and delete user accounts, change user passwords, and manage user roles and permissions.
7. **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.

Rege

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 &#8211; SEO Plugin for WordPress                | seo-by-rank-math   | 98     |
| The SEO Framework                                         | autodescription    | 98     |
| SEOPress, on-site SEO                                     | wp-seopress        | 98     |
| Slim SEO &#8211; Fast &amp; 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 &#8211; Clean, Compress, Cache.               | wp-optimize        | 96     |
+-----------------------------------------------------------+--------------------+--------+
```

You can go to the next page by using the `--page` flag:

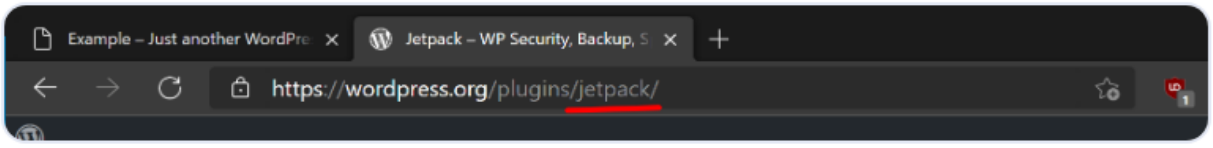
```
$ wp plugin search seo --page=2
```

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         | multicolors         | 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_stat
```
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:

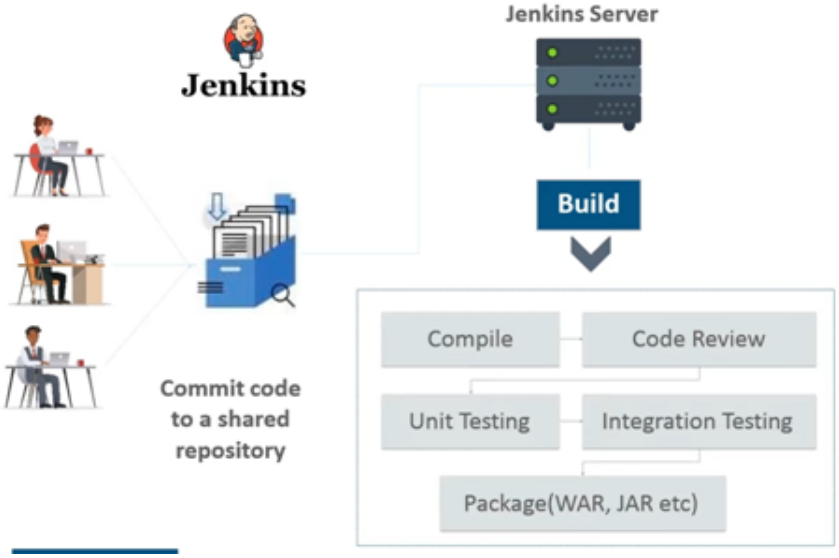
```
$ wp core update
```

Copy

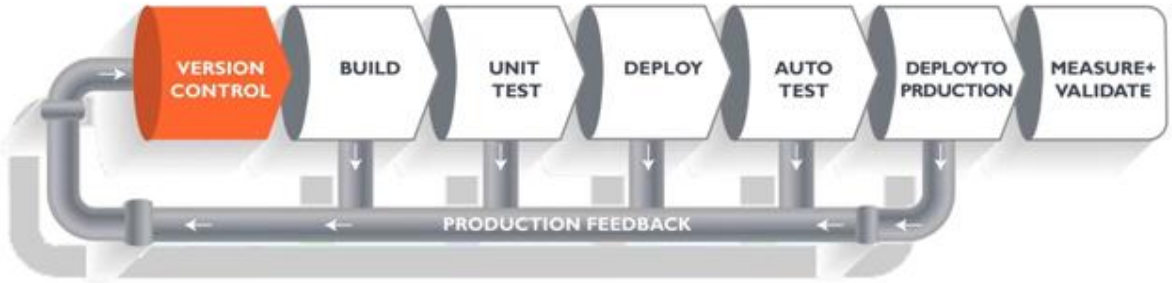
Dev OPS Another

DevOps Stages

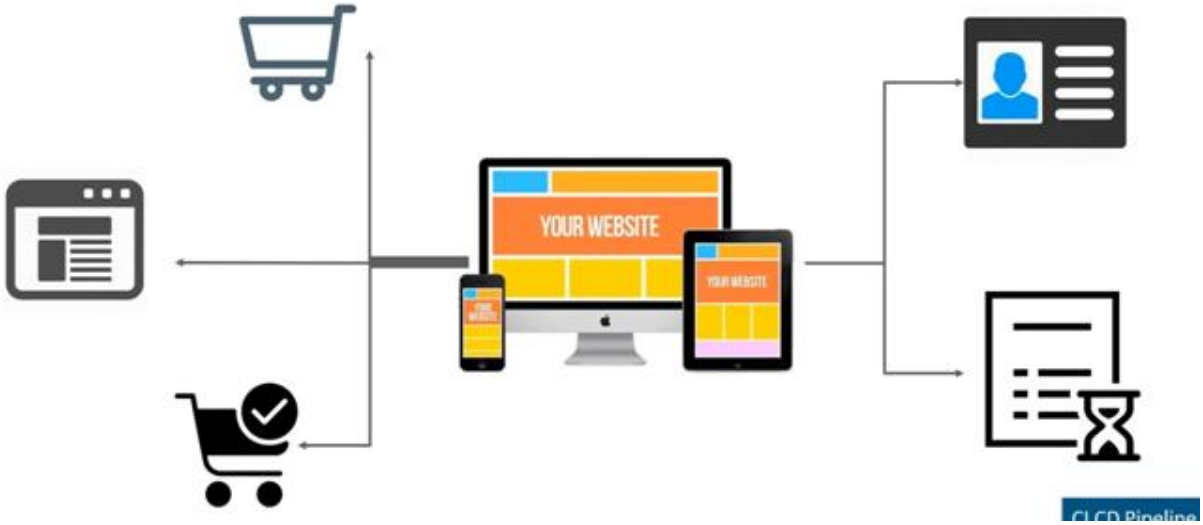
Continuous Integration



What is CI and CD?



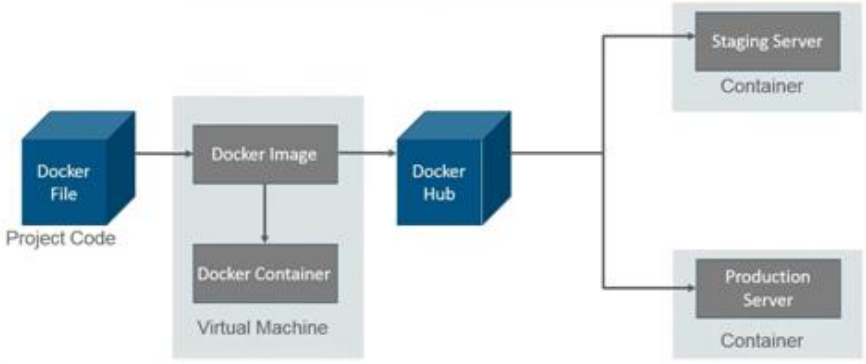
Continuous Integration Example



CI CD Pipeline

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**.