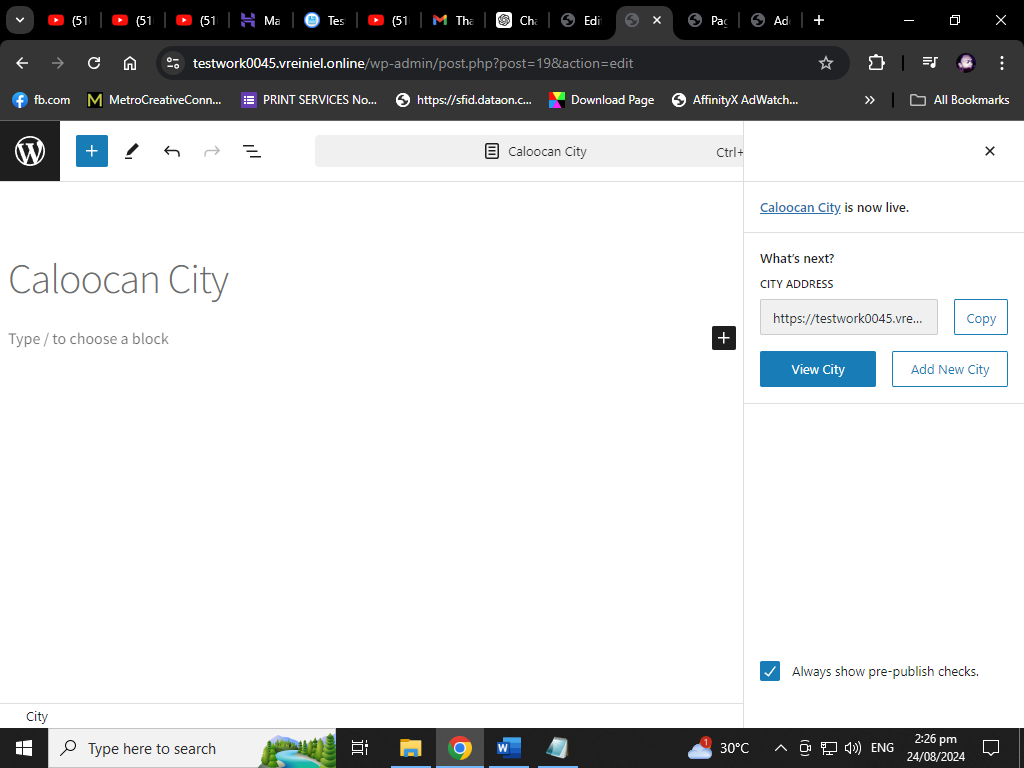
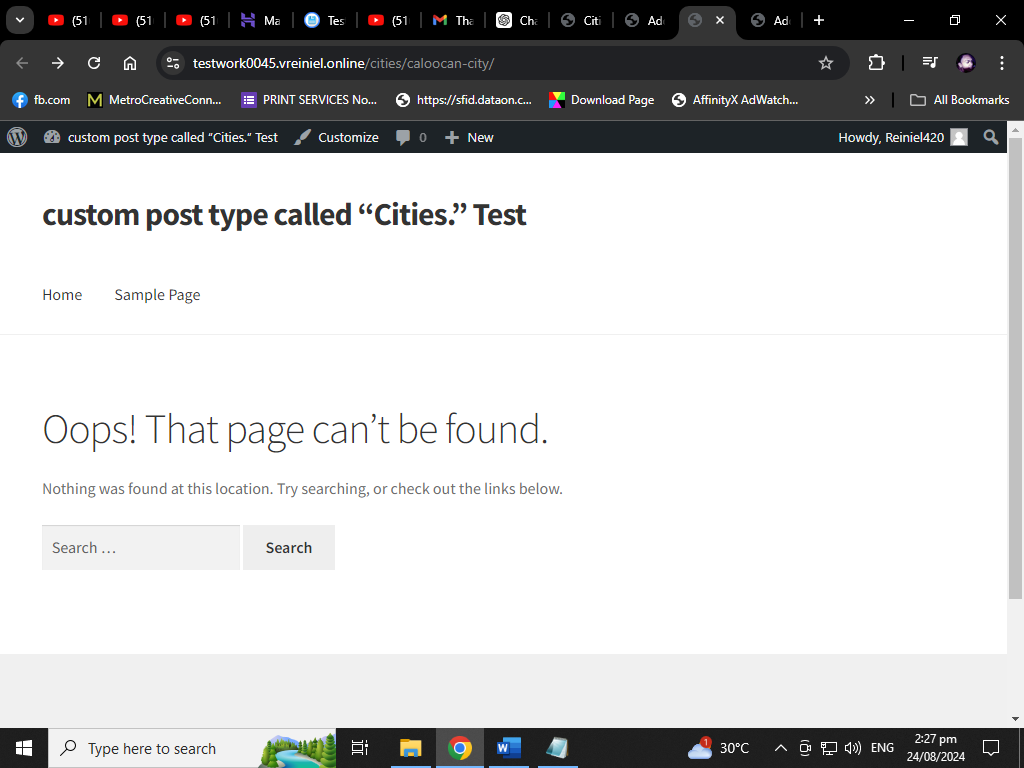


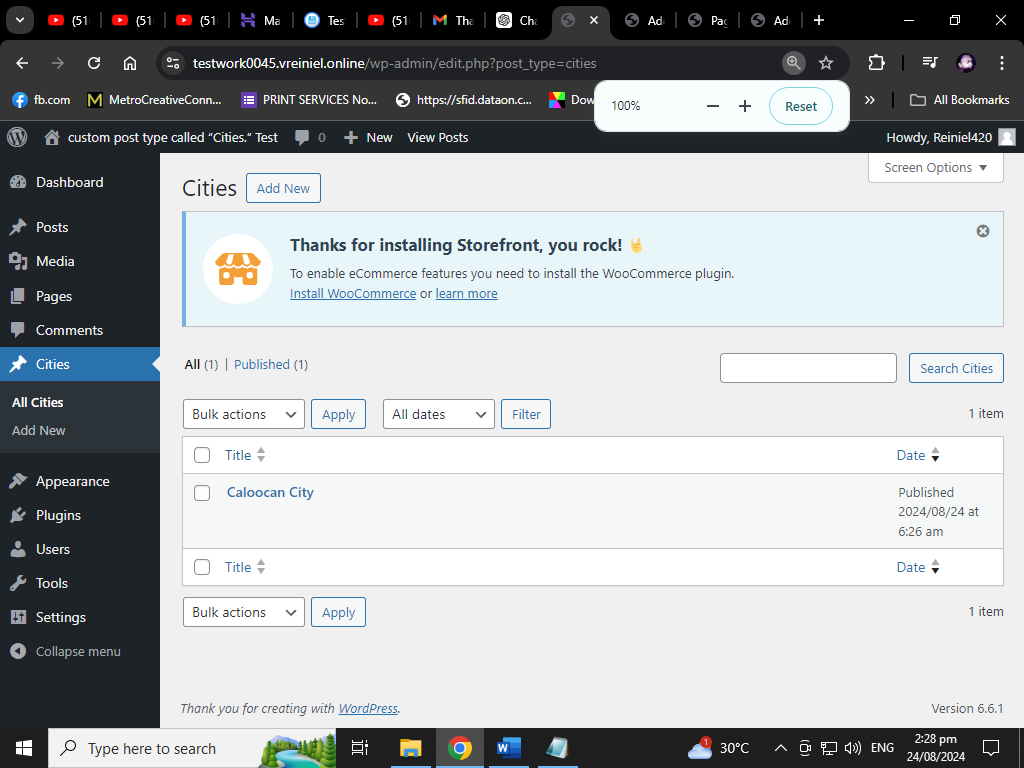
I’d try to add new example Caloocan City



If you Click View City as you can see The Page can’t found



But they were add in our system



**Create a custom post type called “Cities.”**

First, you'll need to register the custom post type called "Cities".

function create\_cities\_cpt() {

$labels = array(

'name' => \_\_('Cities', 'textdomain'),

'singular\_name' => \_\_('City', 'textdomain'),

'menu\_name' => \_\_('Cities', 'textdomain'),

'name\_admin\_bar' => \_\_('City', 'textdomain'),

'add\_new' => \_\_('Add New', 'textdomain'),

'add\_new\_item' => \_\_('Add New City', 'textdomain'),

'new\_item' => \_\_('New City', 'textdomain'),

'edit\_item' => \_\_('Edit City', 'textdomain'),

'view\_item' => \_\_('View City', 'textdomain'),

'all\_items' => \_\_('All Cities', 'textdomain'),

'search\_items' => \_\_('Search Cities', 'textdomain'),

);

$args = array(

'labels' => $labels,

'public' => true,

'has\_archive' => true,

'supports' => array('title', 'editor', 'custom-fields'),

'show\_in\_rest' => true,

);

register\_post\_type('cities', $args);

}

add\_action('init', 'create\_cities\_cpt');

**Function: create\_cities\_cpt()**

This function is responsible for registering a custom post type in WordPress. In this case, the custom post type is called "Cities."

**1. $labels Array**

This array contains the various labels that will be used throughout the WordPress admin interface when working with the "Cities" custom post type.

* **'name'**: The general plural name for the post type, displayed in the admin menu and post overview pages. Here, it's set to "Cities."
* **'singular\_name'**: The singular version of the name, used when a single item is referred to. Here, it's "City."
* **'menu\_name'**: The label for the menu item in the admin sidebar.
* **'name\_admin\_bar'**: The label for the post type that appears in the admin bar when you create a new item.
* **'add\_new'**: The text for the "Add New" button when you want to create a new post of this type.
* **'add\_new\_item'**: The label for the "Add New Item" page title when you’re creating a new city.
* **'new\_item'**: The label used in the title bar when creating a new city.
* **'edit\_item'**: The label used when editing a city.
* **'view\_item'**: The label used when viewing a city.
* **'all\_items'**: The label used for the "All Items" page that lists all cities.
* **'search\_items'**: The label for the search feature in the "Cities" post type.

These labels make the custom post type more user-friendly in the WordPress admin interface.

**2. $args Array**

This array defines the properties and behavior of the custom post type.

* **'labels' => $labels**: Assigns the $labels array to the custom post type.
* **'public' => true**: Makes the custom post type publicly accessible. This means that it can be viewed on the front-end of the website and is visible in the admin panel.
* **'has\_archive' => true**: Enables an archive page for this post type. For example, if you visit yoursite.com/cities/, you’ll see a list of all posts of the "Cities" type.
* **'supports' => array('title', 'editor', 'custom-fields')**: Specifies what features the "Cities" post type supports. In this case:
  + **'title'**: Adds a title field.
  + **'editor'**: Adds the main content editor (the big text area for writing).
  + **'custom-fields'**: Enables the use of custom fields (meta boxes) for additional data.
* **'show\_in\_rest' => true**: Makes the custom post type available in the WordPress REST API, which is useful for building headless WordPress applications or for other integrations.

**3. register\_post\_type('cities', $args);**

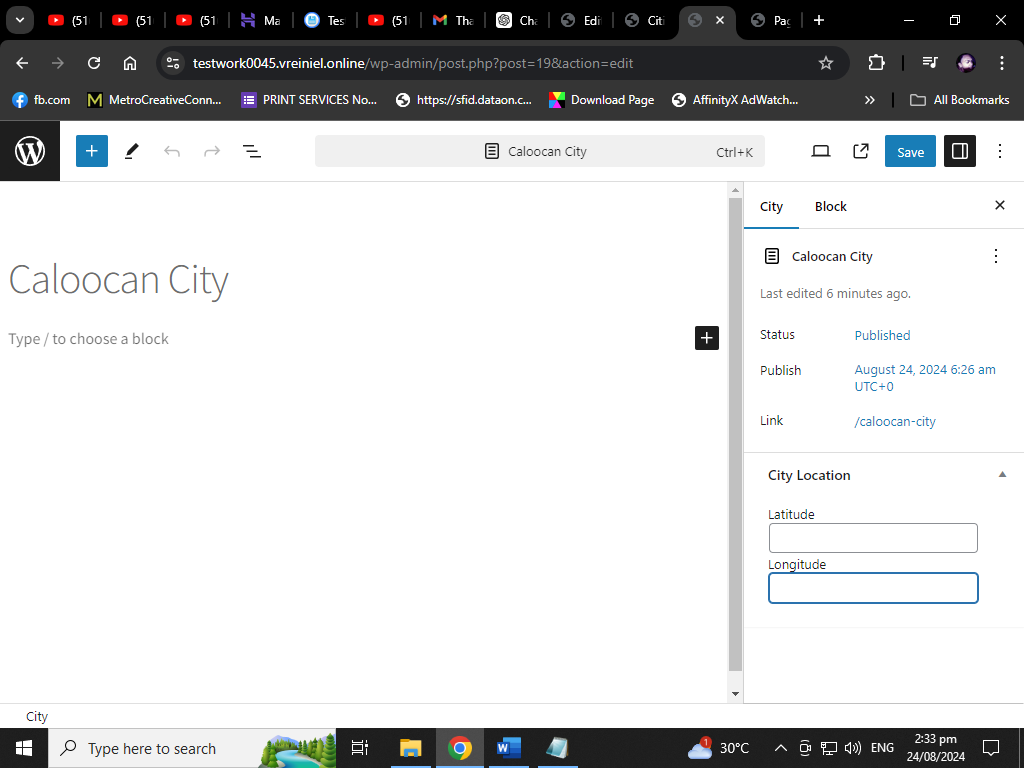
This function registers the "Cities" custom post type with the configuration provided in the $args array. The first parameter ('cities') is the internal name (or slug) for the post type. This is how WordPress identifies and refers to this post type in the code.

**Hook: add\_action('init', 'create\_cities\_cpt');**

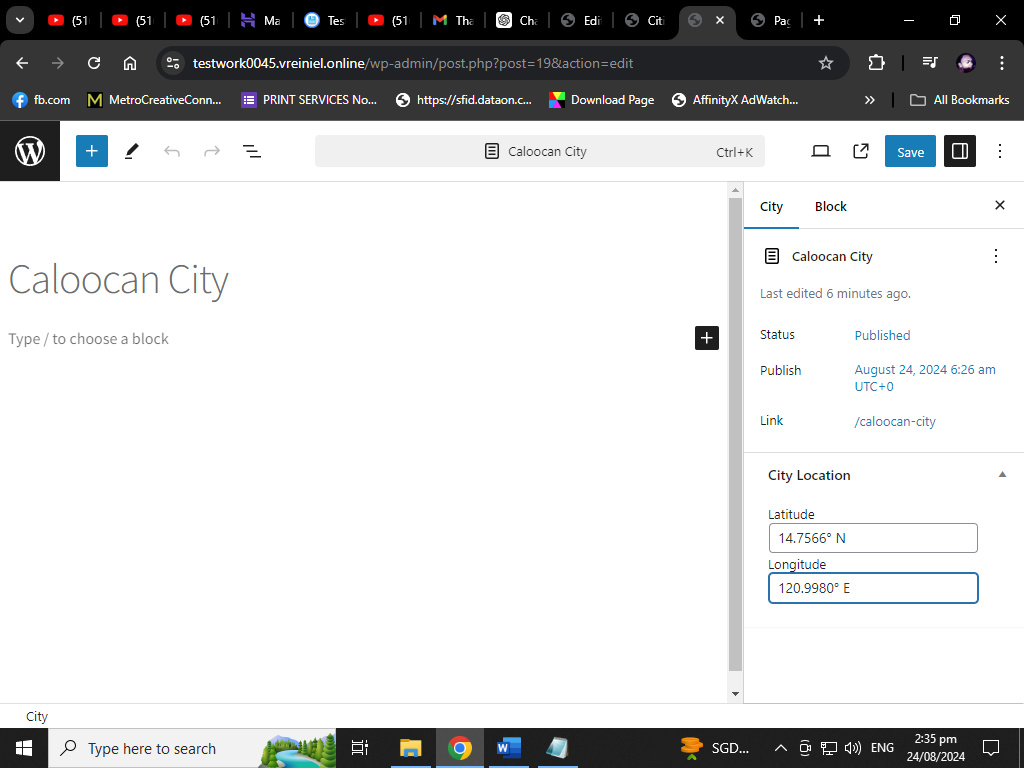
Finally, this line hooks the create\_cities\_cpt function into the init action. The init action is one of the earliest hooks in WordPress, and it's the appropriate place to register custom post types. When WordPress initializes, it will run this function, thereby registering the "Cities" post type.

**Summary**

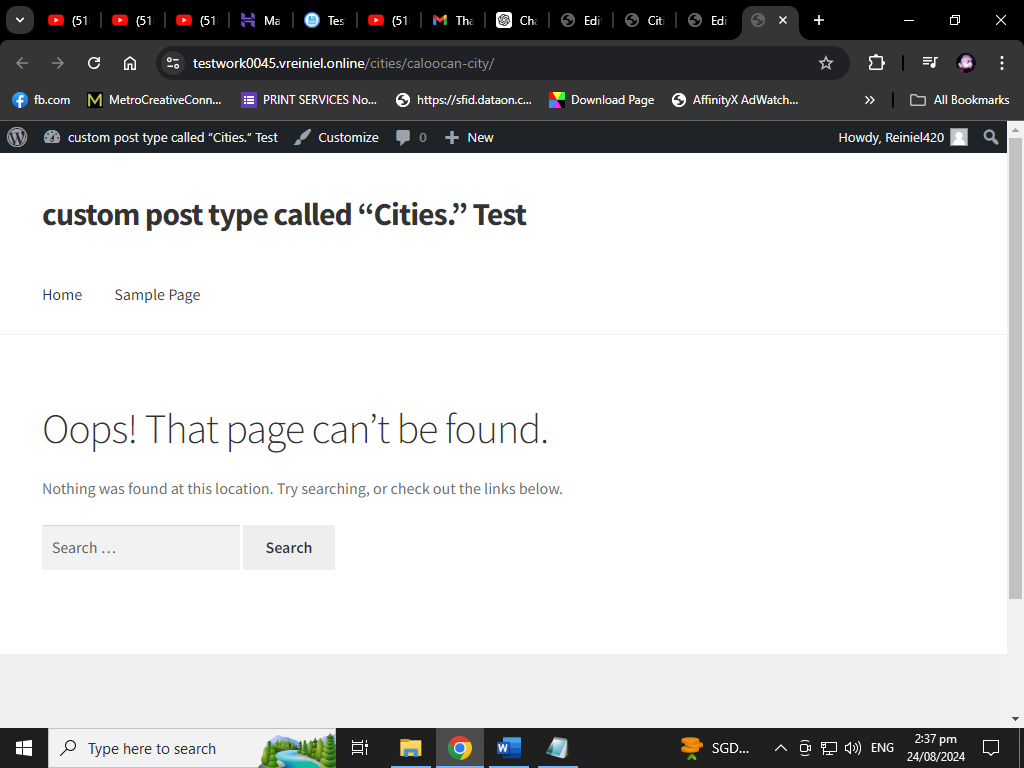
This code creates a new custom post type called "Cities" in WordPress. The post type will have features like a title, content editor, and custom fields. It will be publicly accessible, have its own archive page, and be available through the REST API. The labels help make the admin interface more intuitive when managing cities.

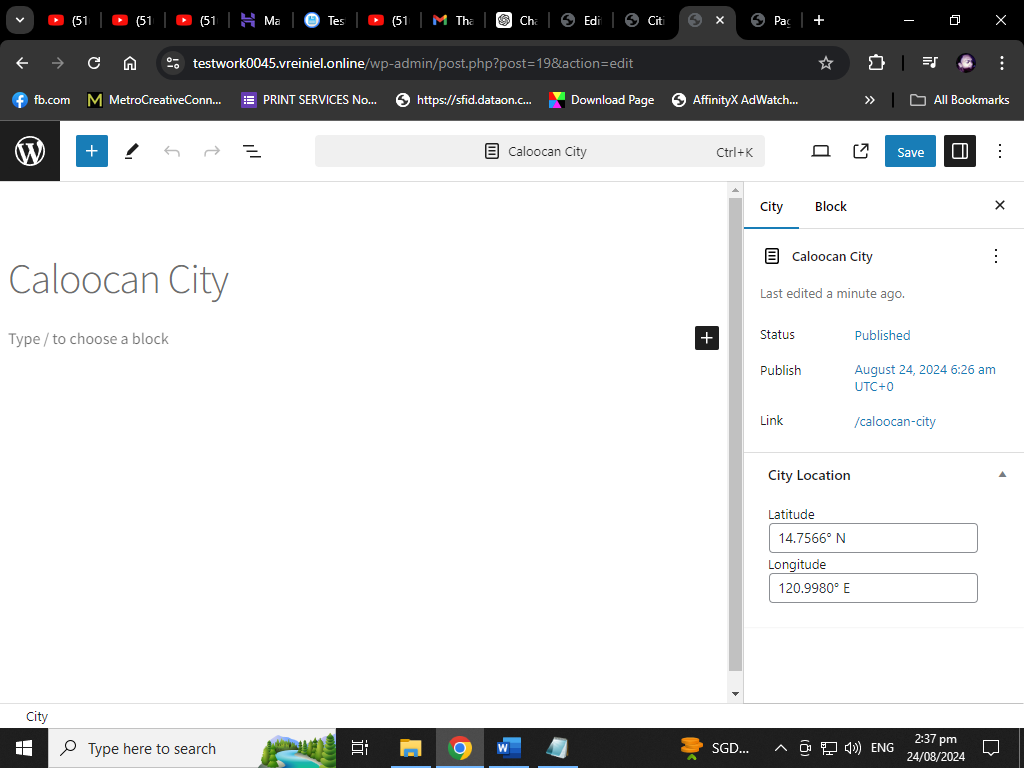


*As you see after I add the code for Meta Box*



*I’d try to add Latitude and Longitude*





*View Page still the same and admin area to view all. but it is saved in editing page where I enter the field*

**Add Meta Box with Custom Fields for Latitude and Longitude**

Next, add a meta box for the "Cities" post type to include latitude and longitude.

function cities\_add\_meta\_box() {

add\_meta\_box(

'cities\_location\_meta\_box',

\_\_('City Location', 'textdomain'),

'cities\_location\_meta\_box\_callback',

'cities',

'side',

'high'

);

}

add\_action('add\_meta\_boxes', 'cities\_add\_meta\_box');

function cities\_location\_meta\_box\_callback($post) {

wp\_nonce\_field('save\_city\_location', 'city\_location\_nonce');

$latitude = get\_post\_meta($post->ID, '\_city\_latitude', true);

$longitude = get\_post\_meta($post->ID, '\_city\_longitude', true);

echo '<label for="city\_latitude">' . \_\_('Latitude', 'textdomain') . '</label>';

echo '<input type="text" id="city\_latitude" name="city\_latitude" value="' . esc\_attr($latitude) . '" size="25" />';

echo '<label for="city\_longitude">' . \_\_('Longitude', 'textdomain') . '</label>';

echo '<input type="text" id="city\_longitude" name="city\_longitude" value="' . esc\_attr($longitude) . '" size="25" />';

}

function save\_city\_location($post\_id) {

if (!isset($\_POST['city\_location\_nonce']) || !wp\_verify\_nonce($\_POST['city\_location\_nonce'], 'save\_city\_location')) {

return;

}

if (defined('DOING\_AUTOSAVE') && DOING\_AUTOSAVE) {

return;

}

if (!current\_user\_can('edit\_post', $post\_id)) {

return;

}

if (isset($\_POST['city\_latitude'])) {

update\_post\_meta($post\_id, '\_city\_latitude', sanitize\_text\_field($\_POST['city\_latitude']));

}

if (isset($\_POST['city\_longitude'])) {

update\_post\_meta($post\_id, '\_city\_longitude', sanitize\_text\_field($\_POST['city\_longitude']));

}

}

add\_action('save\_post', 'save\_city\_location');

Certainly! Let's go through this code step by step. This code adds a meta box to the custom post type "Cities" that allows users to input and save latitude and longitude coordinates for each city.

**1. Function: cities\_add\_meta\_box()**

This function is used to add a meta box to the "Cities" post type.

* **add\_meta\_box()**: This is a WordPress function that adds a meta box to the post editing screen.
  + **'cities\_location\_meta\_box'**: This is the unique ID for the meta box.
  + **\_\_('City Location', 'textdomain')**: This is the title of the meta box that will be displayed in the post editing screen. The \_\_() function is used for translation.
  + **'cities\_location\_meta\_box\_callback'**: This is the callback function that will output the content inside the meta box.
  + **'cities'**: This specifies that the meta box should be added to the "Cities" custom post type.
  + **'side'**: This specifies the location of the meta box on the editing screen. "side" means it will appear in the sidebar.
  + **'high'**: This specifies the priority of the meta box. "high" means it will appear near the top.

This function is hooked into the add\_meta\_boxes action, which triggers it when meta boxes are being added to the post editing screen.

**2. Function: cities\_location\_meta\_box\_callback($post)**

This is the callback function that renders the meta box's content in the post editing screen.

* **wp\_nonce\_field('save\_city\_location', 'city\_location\_nonce')**: This function generates a hidden field with a nonce (a security token) to verify later that the data submission is valid and came from the correct source. The nonce is named 'city\_location\_nonce'.
* **$latitude = get\_post\_meta($post->ID, '\_city\_latitude', true);**: This retrieves the stored latitude value for the current post (if any). The $post->ID gives the current post ID, and get\_post\_meta() is used to fetch the meta value with the key \_city\_latitude. The true parameter means it returns a single value rather than an array.
* **$longitude = get\_post\_meta($post->ID, '\_city\_longitude', true);**: Similarly, this retrieves the stored longitude value for the current post.
* **Output HTML Form Fields**:
  + **<label for="city\_latitude">**: This outputs a label for the latitude input field.
  + **<input type="text" id="city\_latitude" name="city\_latitude" value="' . esc\_attr($latitude) . '" size="25" />**: This creates a text input field for the latitude. It is pre-filled with the previously saved value (if any), escaped for security using esc\_attr().
  + **<label for="city\_longitude">**: This outputs a label for the longitude input field.
  + **<input type="text" id="city\_longitude" name="city\_longitude" value="' . esc\_attr($longitude) . '" size="25" />**: This creates a text input field for the longitude, similarly pre-filled with the saved value.

This function essentially creates the form that allows users to enter or edit the latitude and longitude of a city.

**3. Function: save\_city\_location($post\_id)**

This function handles saving the latitude and longitude data when the post is saved or updated.

* **Security Checks**:
  + **if (!isset($\_POST['city\_location\_nonce']) || !wp\_verify\_nonce($\_POST['city\_location\_nonce'], 'save\_city\_location')) { return; }**: This checks if the nonce field is set and valid. If not, it exits the function without saving.
  + **if (defined('DOING\_AUTOSAVE') && DOING\_AUTOSAVE) { return; }**: This prevents the function from running during an autosave operation.
  + **if (!current\_user\_can('edit\_post', $post\_id)) { return; }**: This checks if the current user has permission to edit the post. If not, it exits the function.
* **Saving Data**:
  + **if (isset($\_POST['city\_latitude'])) { update\_post\_meta($post\_id, '\_city\_latitude', sanitize\_text\_field($\_POST['city\_latitude'])); }**: This checks if the latitude data was submitted. If so, it sanitizes the input (to remove any harmful code) and saves it to the database using update\_post\_meta() with the key \_city\_latitude.
  + **if (isset($\_POST['city\_longitude'])) { update\_post\_meta($post\_id, '\_city\_longitude', sanitize\_text\_field($\_POST['city\_longitude'])); }**: This does the same for the longitude data, saving it with the key \_city\_longitude.

This function ensures that the entered latitude and longitude values are securely saved when the user saves or updates the post.

**4. Hook: add\_action('save\_post', 'save\_city\_location');**

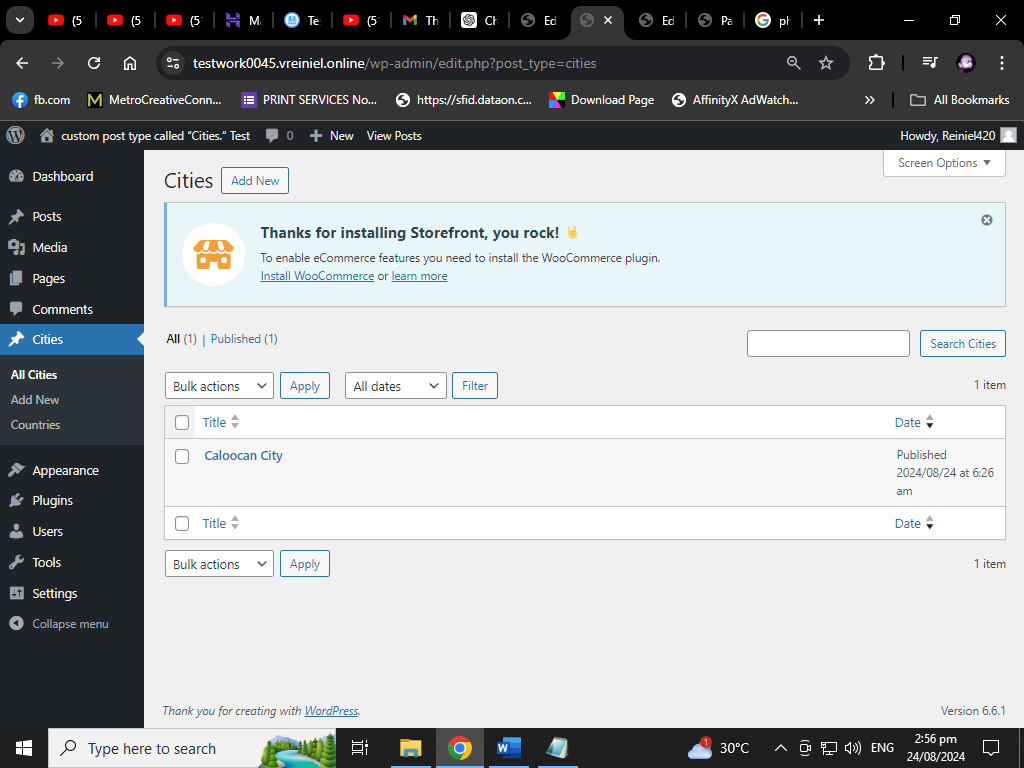
This hook attaches the save\_city\_location() function to the save\_post action, which is triggered whenever a post is saved or updated. This ensures that the latitude and longitude data is saved correctly whenever a city post is saved.

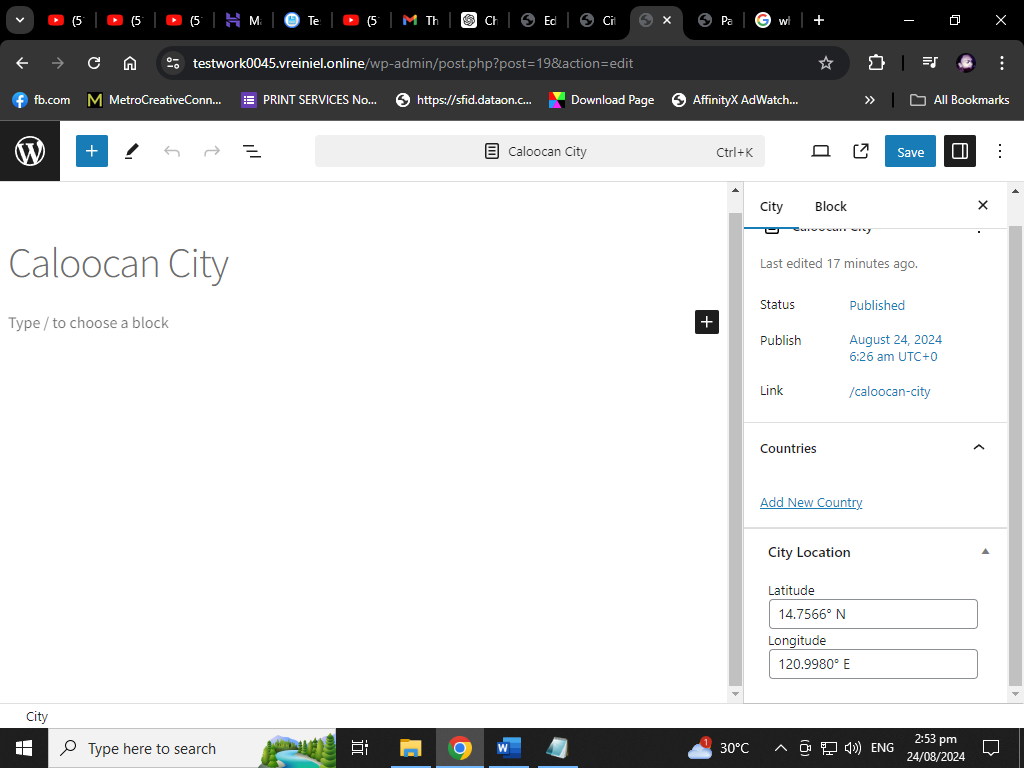
**Summary**

* The code creates a meta box in the "Cities" custom post type where users can input latitude and longitude values for each city.
* The latitude and longitude values are securely saved to the post's meta data when the post is saved.
* The code includes necessary security checks to ensure that only authorized users can save data and that the data is safe to store.

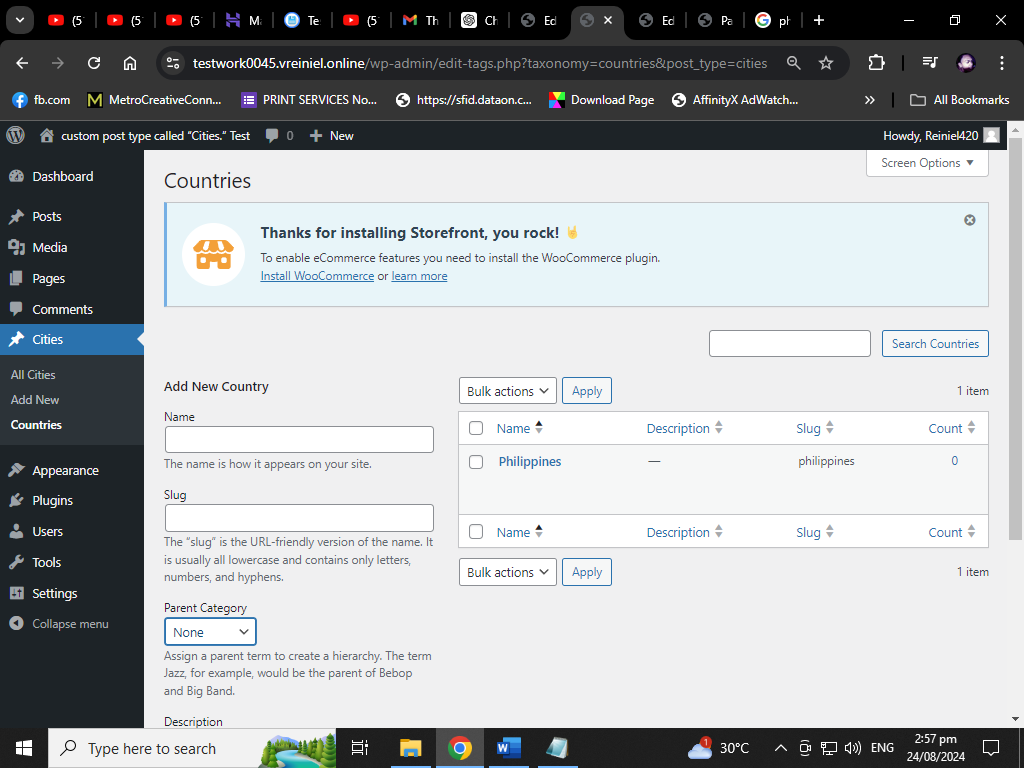
**Create a Custom Taxonomy: "Countries"**

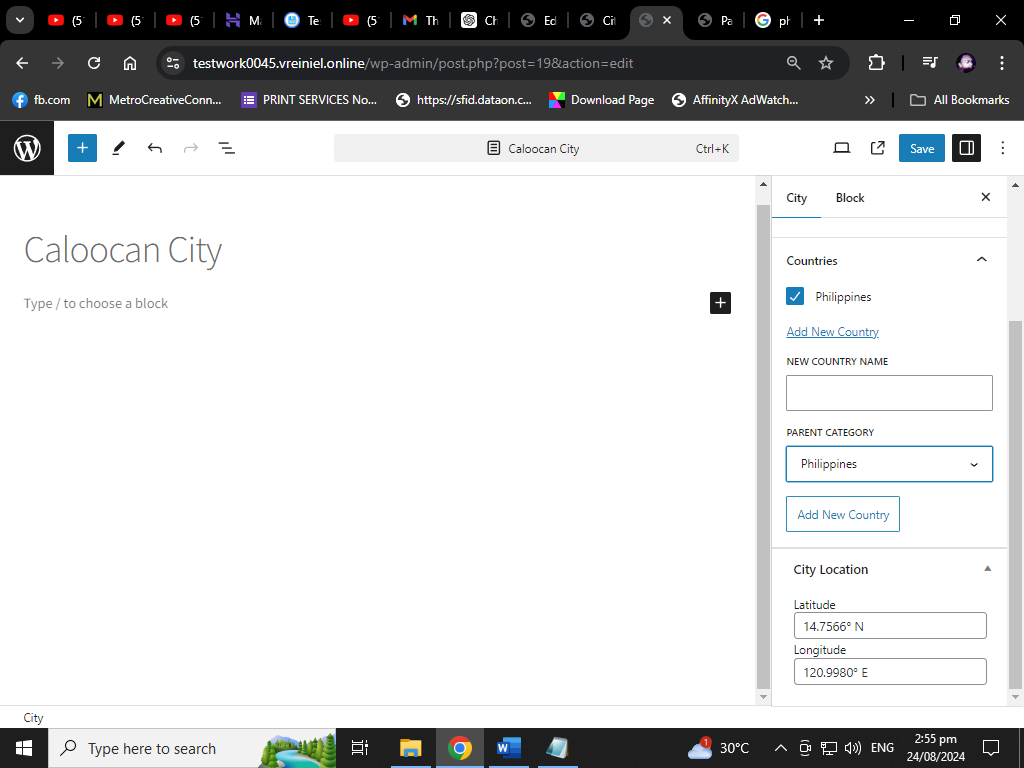
Attach the "Countries" taxonomy to the "Cities" post type.





*After I add taxonomy code they add Countries and I’d try to add.*





*After I add. They show Parent Category as Countries*

*function create\_countries\_taxonomy() {*

*$labels = array(*

*'name' => \_x('Countries', 'taxonomy general name', 'textdomain'),*

*'singular\_name' => \_x('Country', 'taxonomy singular name', 'textdomain'),*

*'search\_items' => \_\_('Search Countries', 'textdomain'),*

*'all\_items' => \_\_('All Countries', 'textdomain'),*

*'edit\_item' => \_\_('Edit Country', 'textdomain'),*

*'update\_item' => \_\_('Update Country', 'textdomain'),*

*'add\_new\_item' => \_\_('Add New Country', 'textdomain'),*

*'new\_item\_name' => \_\_('New Country Name', 'textdomain'),*

*);*

*$args = array(*

*'labels' => $labels,*

*'hierarchical' => true,*

*'show\_in\_rest' => true,*

*);*

*register\_taxonomy('countries', array('cities'), $args);*

*}*

*add\_action('init', 'create\_countries\_taxonomy');*

***1. Function: create\_countries\_taxonomy()***

*This function registers the "Countries" taxonomy and attaches it to the "Cities" custom post type.*

***1.1 $labels Array***

*This array defines the various labels used in the WordPress admin interface when managing the "Countries" taxonomy.*

* ***'name' => \_x('Countries', 'taxonomy general name', 'textdomain')****: The general name for the taxonomy, displayed in the WordPress admin menu. The \_x() function is similar to \_\_() but allows for context-specific translation.*
* ***'singular\_name' => \_x('Country', 'taxonomy singular name', 'textdomain')****: The singular version of the taxonomy name, used when referring to a single item.*
* ***'search\_items' => \_\_('Search Countries', 'textdomain')****: The label for the search box when searching for countries in the admin interface.*
* ***'all\_items' => \_\_('All Countries', 'textdomain')****: The label for the page that lists all countries.*
* ***'edit\_item' => \_\_('Edit Country', 'textdomain')****: The label used when editing a specific country.*
* ***'update\_item' => \_\_('Update Country', 'textdomain')****: The label used when updating a country.*
* ***'add\_new\_item' => \_\_('Add New Country', 'textdomain')****: The label for the link or button to add a new country.*
* ***'new\_item\_name' => \_\_('New Country Name', 'textdomain')****: The label for the input field where the name of a new country is entered.*

*These labels help to make the WordPress admin interface user-friendly and intuitive when dealing with the "Countries" taxonomy.*

***1.2 $args Array***

*This array defines the properties and behavior of the "Countries" taxonomy.*

* ***'labels' => $labels****: Assigns the $labels array to the taxonomy, which provides all the necessary text strings for the admin interface.*
* ***'hierarchical' => true****: This option makes the taxonomy hierarchical, meaning that it behaves like categories (where you can have parent and child terms). If set to false, it would behave like tags (non-hierarchical).*
* ***'show\_in\_rest' => true****: This makes the taxonomy available in the WordPress REST API, which is useful for headless WordPress applications or other integrations.*

***1.3 register\_taxonomy('countries', array('cities'), $args);***

*This function registers the "Countries" taxonomy with the settings defined in the $args array.*

* ***'countries'****: This is the internal name (slug) for the taxonomy. It’s how WordPress identifies this taxonomy in the code.*
* ***array('cities')****: This specifies which post types the taxonomy should be associated with. In this case, it attaches the "Countries" taxonomy to the "Cities" custom post type.*
* ***$args****: This array defines how the taxonomy behaves and how it appears in the WordPress admin interface.*

***2. Hook: add\_action('init', 'create\_countries\_taxonomy');***

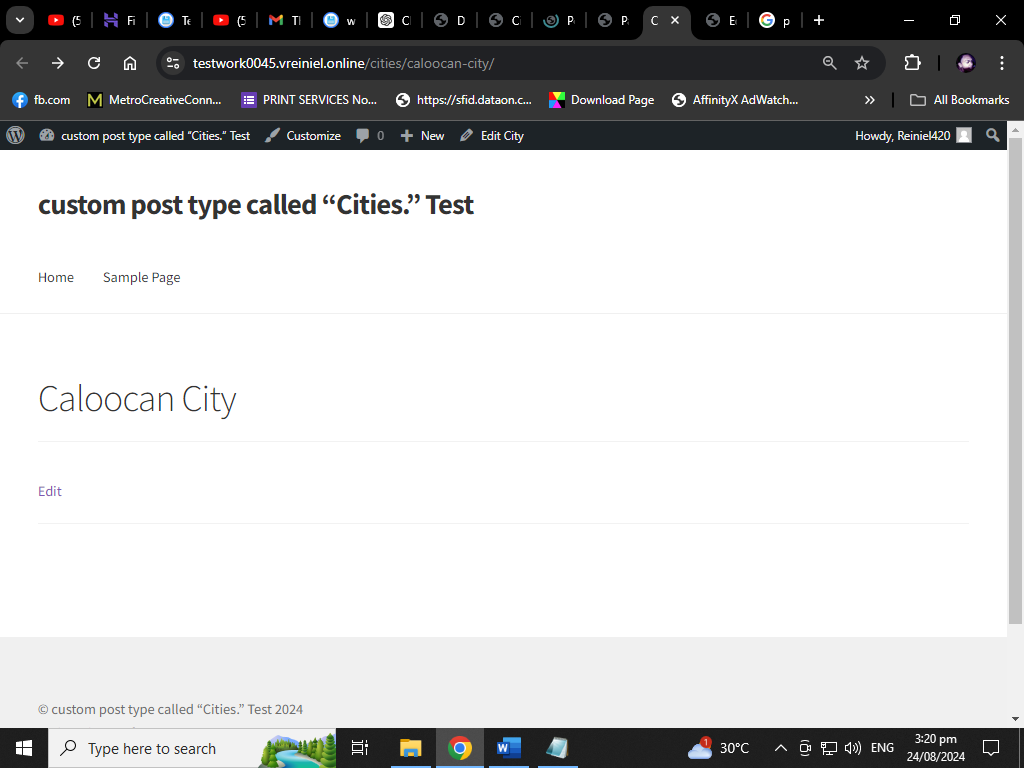
*This line hooks the create\_countries\_taxonomy() function into the init action. The init action is triggered early in WordPress's execution cycle, making it an appropriate place to register custom taxonomies and post types.*

***Summary***

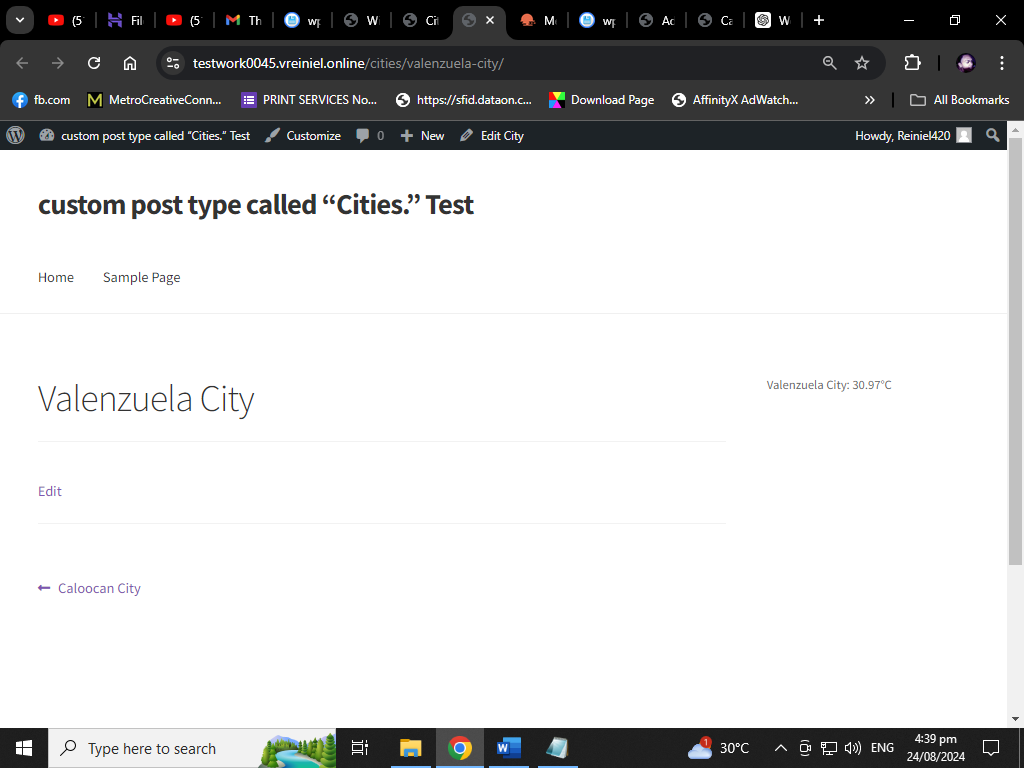
*This code creates a custom taxonomy called "Countries" and associates it with the "Cities" custom post type. The taxonomy is hierarchical (like categories), so you can create parent and child terms, and it’s also available in the REST API. The labels defined in the $labels array make the taxonomy user-friendly in the WordPress admin interface, ensuring that users can easily manage countries associated with cities.*

**4. Create a Widget to Display City Name and Current Temperature**

*Create a custom widget that retrieves and displays the city’s name and current temperature using an external API.*



*After Permalinks change to default the page is see the input city*



*After I add the Widget and Paste the API Keys*

*class City\_Temperature\_Widget extends WP\_Widget {*

*function \_\_construct() {*

*parent::\_\_construct(*

*'city\_temperature\_widget',*

*\_\_('City Temperature Widget', 'text\_domain'),*

*array('description' => \_\_('Displays the temperature of a city', 'text\_domain'))*

*);*

*}*

*public function widget($args, $instance) {*

*echo $args['before\_widget'];*

*$city\_id = !empty($instance['city\_id']) ? $instance['city\_id'] : 0;*

*if ($city\_id) {*

*$city\_name = get\_the\_title($city\_id);*

*$latitude = get\_post\_meta($city\_id, '\_city\_latitude', true);*

*$longitude = get\_post\_meta($city\_id, '\_city\_longitude', true);*

*// API call to OpenWeatherMap*

*$api\_key = 'your\_openweathermap\_api\_key';*

*$weather\_data = wp\_remote\_get("https://api.openweathermap.org/data/2.5/weather?lat=$latitude&lon=$longitude&units=metric&appid=$api\_key");*

*if (is\_wp\_error($weather\_data)) {*

*echo 'Unable to retrieve weather data.';*

*} else {*

*$body = wp\_remote\_retrieve\_body($weather\_data);*

*$weather = json\_decode($body);*

*if ($weather && $weather->main->temp) {*

*echo "<p>$city\_name: " . $weather->main->temp . "°C</p>";*

*} else {*

*echo 'Weather data not available.';*

*}*

*}*

*}*

*echo $args['after\_widget'];*

*}*

*public function form($instance) {*

*$city\_id = !empty($instance['city\_id']) ? $instance['city\_id'] : '';*

*?>*

*<p>*

*<label for="<?php echo $this->get\_field\_id('city\_id'); ?>">City:</label>*

*<select class="widefat" id="<?php echo $this->get\_field\_id('city\_id'); ?>" name="<?php echo $this->get\_field\_name('city\_id'); ?>">*

*<?php*

*$cities = get\_posts(array('post\_type' => 'cities', 'numberposts' => -1));*

*foreach ($cities as $city) {*

*echo '<option value="' . $city->ID . '" ' . selected($city\_id, $city->ID, false) . '>' . $city->post\_title . '</option>';*

*}*

*?>*

*</select>*

*</p>*

*<?php*

*}*

*public function update($new\_instance, $old\_instance) {*

*$instance = array();*

*$instance['city\_id'] = (!empty($new\_instance['city\_id'])) ? strip\_tags($new\_instance['city\_id']) : '';*

*return $instance;*

*}*

*}*

*function register\_city\_temperature\_widget() {*

*register\_widget('City\_Temperature\_Widget');*

*}*

*add\_action('widgets\_init', 'register\_city\_temperature\_widget');*

**1. Class Definition: City\_Weather\_Widget**

* **\_\_construct()**: This constructor method initializes the widget, setting its name ("City Weather Widget") and description. It also calls the parent constructor to ensure the widget is properly set up.
* **widget($args, $instance)**: This method is responsible for rendering the widget on the front end.
  + **City Selection**: It first retrieves the selected city ID ($city\_id) from the widget instance.
  + **City Data Retrieval**: The city’s name, latitude, and longitude are fetched using WordPress functions like get\_the\_title() and get\_post\_meta().
  + **API Call**: The widget constructs a URL to the OpenWeatherMap API using the city’s latitude and longitude. It then sends a request to the API and retrieves the current temperature.
  + **Display**: The city name and temperature are displayed within the widget.
* **form($instance)**: This method creates the form that appears in the widget settings area within the WordPress admin. It allows the user to select a city from the "Cities" custom post type.
  + **City Dropdown**: The form includes a dropdown menu populated with the titles of all posts in the "Cities" custom post type.
* **update($new\_instance, $old\_instance)**: This method handles saving the widget settings. It updates the selected city ID when the widget settings are saved.

**2. Widget Registration: register\_city\_weather\_widget()**

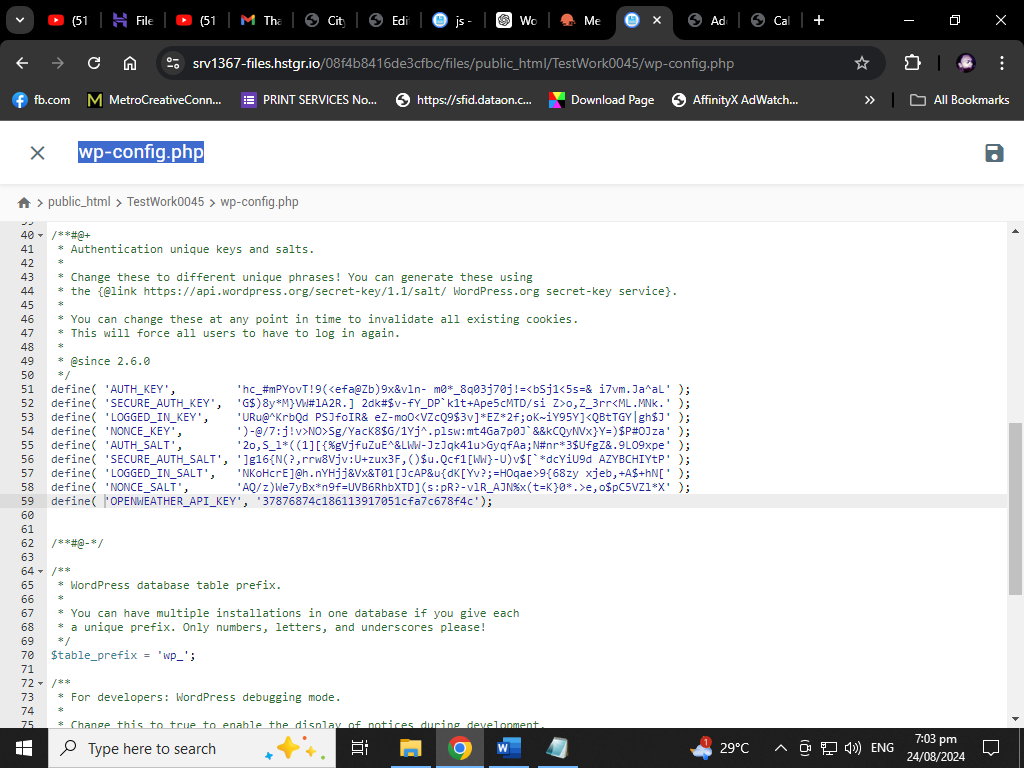
* This function registers the City\_Weather\_Widget with WordPress, making it available for use in the Widgets area under "Appearance > Widgets."

**3. Usage**

* After adding this code to your theme’s functions.php file or a custom plugin, you can use the widget by:
  + Navigating to **Appearance > Widgets** in the WordPress admin area.
  + Dragging and dropping the "City Weather Widget" into a widget area (e.g., a sidebar or footer).
  + Selecting a city from the dropdown in the widget settings and saving the widget.

**API Key**

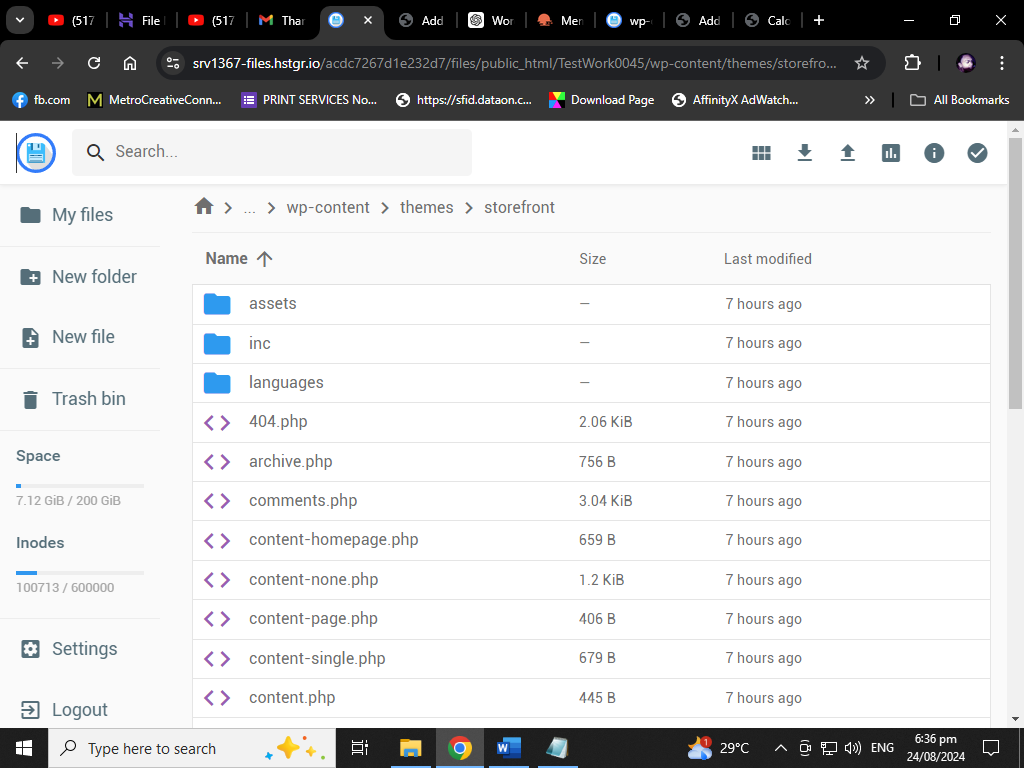
* **Note**: Replace 'your\_api\_key' with your actual OpenWeatherMap API key to fetch the temperature data.



**Add this code in Wp-config.php**

**Summary**

This widget allows users to display the current temperature of a selected city from the custom post type "Cities" on their WordPress site. The temperature data is retrieved using the OpenWeatherMap API based on the city’s latitude and longitude.



*Go to the directory of your active theme. This is typically located in wp-content/themes/your-theme-name/*

**Create a New File:** In your theme directory, create a new PHP file for your custom page template. For example, you might name it page-city-temperatures.php.

**Add Template Header Information:** Open the new PHP file and add a comment at the top to define it as a page template. This is essential for WordPress to recognize it.

**Add Your Template Code:** Below the template header, you can start adding your custom PHP, HTML, and WordPress template tags.

File: Page-Cities.php

<?php

/\*

Template Name: Cities Table

\*/

get\_header();

// Custom action hook before the table

do\_action('before\_cities\_table');

?>

<div id="cities-search">

<input type="text" id="city-search-input" placeholder="Search for a city...">

</div>

<table id="cities-table">

<thead>

<tr>

<th>Country</th>

<th>City</th>

<th>Temperature (°C)</th>

</tr>

</thead>

<tbody id="cities-table-body">

<?php

global $wpdb;

$results = $wpdb->get\_results("

SELECT posts.ID, posts.post\_title, meta\_lat.meta\_value as latitude, meta\_lon.meta\_value as longitude, term\_tax.term\_id as country\_id, terms.name as country\_name

FROM $wpdb->posts as posts

JOIN $wpdb->postmeta as meta\_lat ON posts.ID = meta\_lat.post\_id AND meta\_lat.meta\_key = '\_city\_latitude'

JOIN $wpdb->postmeta as meta\_lon ON posts.ID = meta\_lon.post\_id AND meta\_lon.meta\_key = '\_city\_longitude'

JOIN $wpdb->term\_relationships as term\_rel ON posts.ID = term\_rel.object\_id

JOIN $wpdb->term\_taxonomy as term\_tax ON term\_rel.term\_taxonomy\_id = term\_tax.term\_taxonomy\_id AND term\_tax.taxonomy = 'countries'

JOIN $wpdb->terms as terms ON term\_tax.term\_id = terms.term\_id

WHERE posts.post\_type = 'cities' AND posts.post\_status = 'publish'

");

foreach ($results as $city) {

// Fetch temperature from OpenWeatherMap API

$api\_key = '37876874c186113917051cfa7c678f4c';

$weather\_url = "http://api.openweathermap.org/data/2.5/weather?lat={$city->latitude}&lon={$city->longitude}&appid={$api\_key}&units=metric";

$weather\_data = wp\_remote\_get($weather\_url);

$temperature = 'N/A';

if (!is\_wp\_error($weather\_data)) {

$body = wp\_remote\_retrieve\_body($weather\_data);

$data = json\_decode($body);

$temperature = isset($data->main->temp) ? $data->main->temp : 'N/A';

}

echo "<tr data-city-id='{$city->ID}'>";

echo "<td>{$city->country\_name}</td>";

echo "<td>{$city->post\_title}</td>";

echo "<td>{$temperature}</td>";

echo "</tr>";

}

?>

</tbody>

</table>

<?php

// Custom action hook after the table

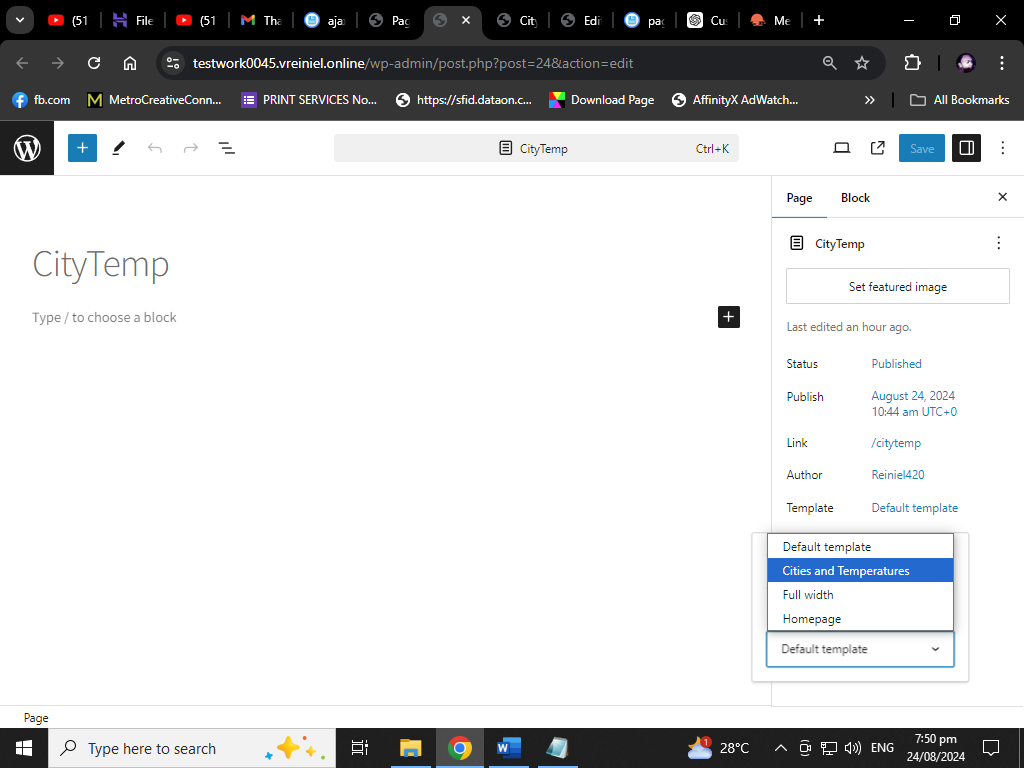
do\_action('after\_cities\_table');

get\_footer();

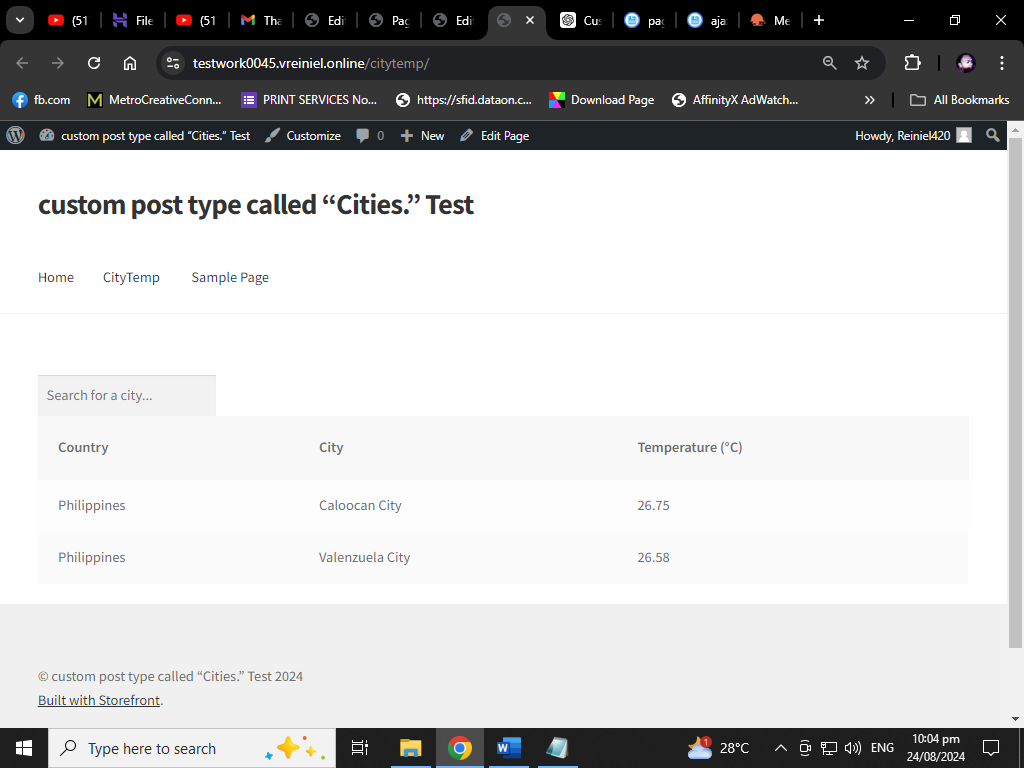
**Save the File**

**Create a New Page in WordPress**

1. **Log In to WordPress Admin: Go to your WordPress admin dashboard.**
2. **Add a New Page: Navigate to Pages > Add New.**
3. **Select Your Template: On the right side of the page editor, you’ll find the Page Attributes box. In the Template dropdown, select your new template (e.g., "City Temperatures").**



1. **Publish the Page: Add any content you want to the page, then click Publish to make the page live.**



**Add WP Ajax Search for Cities**

**To enable searching for cities via AJAX, add the following JavaScript in your theme or enqueue it properly:**

**Add the new file JS file in theme path**

**And add this code**

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

$('#city-search-input').on('input', function() {

var searchQuery = $(this).val();

$.ajax({

url: ajax\_data.ajaxurl, // Correct usage of localized script object

type: 'POST',

data: {

action: 'search\_cities', // AJAX action name

search: searchQuery

},

success: function(response) {

$('#cities-table-body').html(response); // Update the table body with response

},

error: function(xhr, status, error) {

console.log('AJAX Error: ' + error); // Log any AJAX errors

}

});

});

});

**Enqueue the Script:**

**Add the following to your theme’s functions.php file to enqueue the script.**

function enqueue\_search\_cities\_script() {

wp\_enqueue\_script('search-cities', get\_template\_directory\_uri() . '/assets/js/search-cities.js', array('jquery'), null, true);

// Localize the script with the AJAX URL

wp\_localize\_script('search-cities', 'ajax\_data', array(

'ajaxurl' => admin\_url('admin-ajax.php')

));

}

add\_action('wp\_enqueue\_scripts', 'enqueue\_search\_cities\_script');

**/\*\* Handle the AJAX Request \*/**

function search\_cities() {

global $wpdb;

$search = isset($\_POST['search']) ? sanitize\_text\_field($\_POST['search']) : '';

// Debugging

error\_log('Search Query: ' . $search);

$results = $wpdb->get\_results($wpdb->prepare("

SELECT posts.ID, posts.post\_title, meta\_lat.meta\_value as latitude, meta\_lon.meta\_value as longitude, term\_tax.term\_id as country\_id, terms.name as country\_name

FROM $wpdb->posts as posts

JOIN $wpdb->postmeta as meta\_lat ON posts.ID = meta\_lat.post\_id AND meta\_lat.meta\_key = '\_city\_latitude'

JOIN $wpdb->postmeta as meta\_lon ON posts.ID = meta\_lon.post\_id AND meta\_lon.meta\_key = '\_city\_longitude'

JOIN $wpdb->term\_relationships as term\_rel ON posts.ID = term\_rel.object\_id

JOIN $wpdb->term\_taxonomy as term\_tax ON term\_rel.term\_taxonomy\_id = term\_tax.term\_taxonomy\_id AND term\_tax.taxonomy = 'countries'

JOIN $wpdb->terms as terms ON term\_tax.term\_id = terms.term\_id

WHERE posts.post\_type = 'cities' AND posts.post\_status = 'publish' AND posts.post\_title LIKE %s

", '%' . $wpdb->esc\_like($search) . '%'));

if ($results) {

foreach ($results as $city) {

// Debugging

error\_log('Found City: ' . $city->post\_title);

$api\_key = '37876874c186113917051cfa7c678f4c';

$weather\_url = "http://api.openweathermap.org/data/2.5/weather?lat={$city->latitude}&lon={$city->longitude}&appid={$api\_key}&units=metric";

$weather\_data = wp\_remote\_get($weather\_url);

$temperature = 'N/A';

if (!is\_wp\_error($weather\_data)) {

$body = wp\_remote\_retrieve\_body($weather\_data);

$data = json\_decode($body);

$temperature = isset($data->main->temp) ? $data->main->temp : 'N/A';

}

echo "<tr data-city-id='{$city->ID}'>";

echo "<td>{$city->country\_name}</td>";

echo "<td>{$city->post\_title}</td>";

echo "<td>{$temperature}</td>";

echo "</tr>";

}

} else {

echo "<tr><td colspan='3'>No results found</td></tr>";

}

wp\_die();

}

add\_action('wp\_ajax\_search\_cities', 'search\_cities');

add\_action('wp\_ajax\_nopriv\_search\_cities', 'search\_cities');

**5. Add Custom Action Hooks**

In your template, you’ve already included custom action hooks before\_cities\_temperatures\_table and after\_cities\_temperatures\_table. To utilize these, you can hook custom functions to these actions in your functions.php file.

// Function to display content before the table

function before\_cities\_table\_content() {

echo '<div class="before-table-message">Here is some information before the table.</div>';

}

add\_action('before\_cities\_table', 'before\_cities\_table\_content');

// Function to display content after the table

function after\_cities\_table\_content() {

echo '<div class="after-table-message">Here is some information after the table.</div>';

}

add\_action('after\_cities\_table', 'after\_cities\_table\_content');

