CRUD Operation on WordPress Posts From C# over REST API

Sudip Chakraborty [1] & P. S. Aithal [2]
[1] D.Sc. Researcher, Institute of Computer Science and Information Sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in
[2] Professor, Institute of Management & Commerce, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

Area/Section: Computer Science.
Type of the Paper: Experimental Research.
Type of Review: Peer Reviewed as per |C|O|P|E| guidance.
Indexed in: OpenAIRE.
DOI: https://doi.org/10.5281/zenodo.10264407
Google Scholar Citation: IJMTS

How to Cite this Paper:
Chakraborty, S., & Aithal, P. S., (2023). CRUD Operation on WordPress Posts From C# over REST API. *International Journal of Management, Technology, and Social Sciences (IJMTS), 8*(4), 223-231. DOI: https://doi.org/10.5281/zenodo.10264407

International Journal of Management, Technology, and Social Sciences (IJMTS)
A Refereed International Journal of Srinivas University, India.

CrossRef DOI: https://doi.org/10.47992/IJMTS.2581.6012.0320

Received on: 09/11/2023
Published on: 05/12/2023

CRUD Operation on WordPress Posts From C# over REST API

Sudip Chakraborty [1] & P. S. Aithal [2]

[1] D.Sc. Researcher, Institute of Computer Science and Information Sciences, Srinivas University, Mangalore-575 001, India,
OrcidID: 0000-0002-1088-663X; E-mail: sudip.pdf@srinivasuniversity.edu.in

[2] Professor, Institute of Management & Commerce, Srinivas University, Mangalore, India,
OrcidID: 0000-0002-4691-8736; E-Mail: psaithal@gmail.com

ABSTRACT

Purpose: *WordPress is a sound content management system on the web. We can build a website with little effort. Around 40% of internet websites are powered by WordPress. Using the WordPress website, our researcher can store research data inside the website database without spending money on the cloud. We demonstrate how to run CRUD operations on a WordPress-powered website database using WordPress Posts. We observe the essential CRUD operation so that our researcher can integrate it easily into their project with complex procedures. We use REST API for communication between the WordPress and the client application. We created a GUI in Visual Studio using C# language for easy operation. An HTTP client is used for communication. The complete project code is available to download and continue research work.*

Design/Methodology/Approach: *First, we install a "local" application in our working system. Then, we create a brand-new WordPress website. We activated the latest theme, which is now Twenty twenty-four. We make a custom route and add a posts operation handling module inside the theme folder. This is the server-side action. On the other hand, inside the Visual Studio 2022 community edition, using C# language, we created a GUI-based client application to interact with users. We added an HTTP client module to communicate with the server.*

Findings/Result: *We experimented with practical, real-world examples. The overall performance is excellent and helpful in real-world scenarios without any issues. Network latency is sometimes present, creating problems in real-time data processing. The researcher creating a custom route in their online website can experiment. It will not have any adverse effect on keeping the website live.*

Originality/Value: *WordPress uses the database to store post-type data and some WordPress-related configurations. Out of the traditional way, we can also reserve our research data as post-type content according to our project requirements. Keeping our data as the post-type of content from the C# application is the least available documentation on the web. We demonstrate here through Practical examples. Some valuable reference information can be found here.*

Paper Type: *Experimental-based Research.*

Keywords: CRUD operation on WordPress website, HTTP client demonstration in C#, WordPress Post Type demonstration. WordPress CRUD operation from C#.

1. INTRODUCTION :

Nowadays, WordPress is a popular content management system. Around forty percent of the website uses this platform. Using this, any website can be created. The typical scenario is that WordPress uses its database to store the data for its content management. To host our website, we purchase space in the cloud from the provider. Most of the institute's present cloud faces. When researchers need to store their sensor data, they again subscribe to an IoT platform to keep the data available around the clock from anywhere.

Here, we will see that we can keep our sensor data in our website database as posts. For this purpose, we chose a WordPress website database. We started with essential CRUD operation for easy understanding. We didn't have a complex code with error handling, which can be added easily later in the final implementation phase. Use a local server to keep the experiment cost as cheap as possible. Using the "local" application, we install WordPress locally, create a custom route, and add our post-data type handling routine.
The paper is organized as follows: Section 2 provides an overview of related work already done on IoT. Section 3 discusses the objective of the research work. Section 4 highlights the methodology we used for the research work. In section 5, we do the actual experiment. Section 6 provides recommendations for reading or watching videos to understand the research topic better. Finally, Section 7 concludes the paper and offers future research directions.

2. RELATED WORKS :

Anuar, A. et al. explain web application development with integrated records management is essential for Re-CRUD [1]. Leone, Set. al. Integrating component-based web engineering into content management systems in Web Engineering [2]. Bandirmali N. et al. describe a novel memory table-based content management framework for automatic website generation [3]. Prstačić, S. et al. demonstrate Web application frameworks as reusable components [4]. Leone, S et al. Component-based web engineering using shared components and connectors [5]. Murolo, A. et al. Deriving custom post types from digital mock-ups [6]. Chakraborty S. et al. demonstrated how to display the message inside the debug panel [7]. Their paper [8] showed the GIT process using a practical approach. Another report [9] discussed how to build an intelligent IDE. Nowadays, MVVM is the famous architecture of any professional project. In the article, [10] demonstrates how to implement MVVM.

3. OBJECTIVES :

The research aims to provide some reference information on using post-type operations from the C# environment. The researcher needs to store their sensor data on the cloud server. Using the WordPress website, we can keep our sensor data easily. The send or receive data from or to the WordPress website is post-type content. Post-type content has several advantages, such as being easy to implement, read, and display. There are several options are available inside the WordPress to display the post in tabular form. Here, we provide practical examples to experiment with WordPress post types data so researchers can integrate them into their projects efficiently.

4. APPROACH AND METHODOLOGY :

Figure 4.1 depicts the block diagram of the project. The central part of the project is the C# application. It runs in the UI thread. Several modules are needed to operate the main component of the application. The "**WordPress_post.cs**" is a customized module that is project-specific. It has several functions that generate JSON objects to send to the server. According to our needs, it provides different objects for CRUD operation. The "**Global. cs**" module stores additional global variables of the project needed for separate modules for various functions. The "**MSG.cs**" module helps to display inside the list box. The GUI module or main UI thread is used to interact with the user. DataGrid control. It is used to display and edit the fetched data. A list box is used to show the communication status with the server. The button control sends the command to the server, fetches the response, and displays it inside the list box. The textbox receives inputs from the user. The "**WordPress_Connect.cs**" module communicates with the WordPress server. Several modules like "**System.Net.Http**" and "**HttpClient**" are used to execute the module. On the other hand, we are working with the latest theme, twenty-four. We create our module in PHP for the Posts handle. We register a custom route for convenience so that our custom code will not be mixed up with another WordPress module.

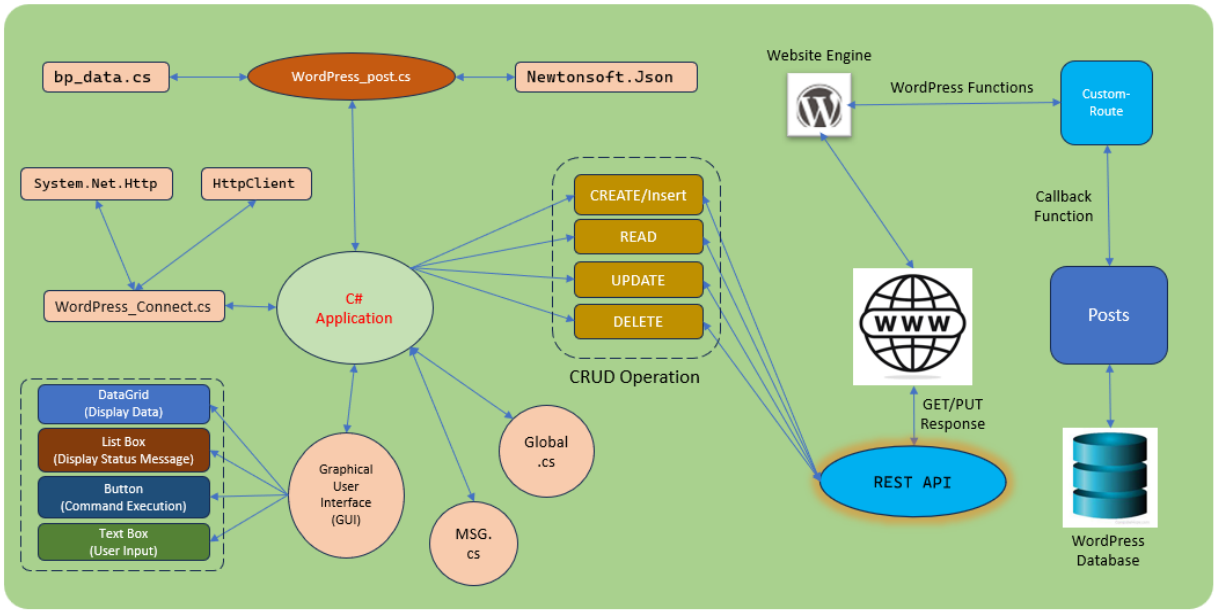
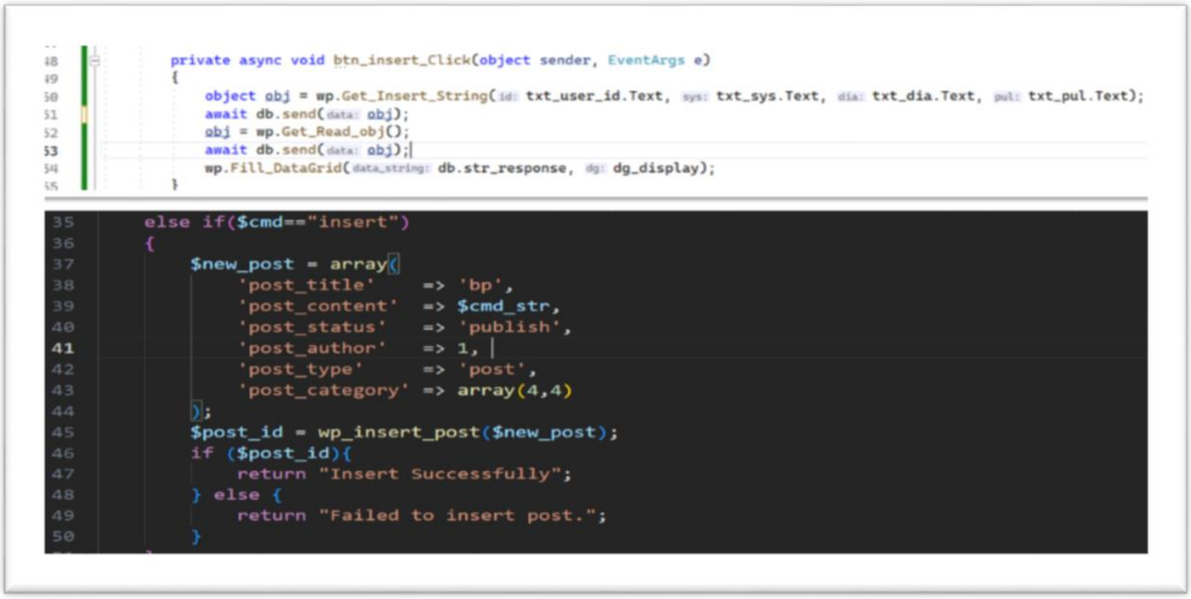


Fig. 1: The project block diagram

Now, let us discuss how our CRUD operation works:

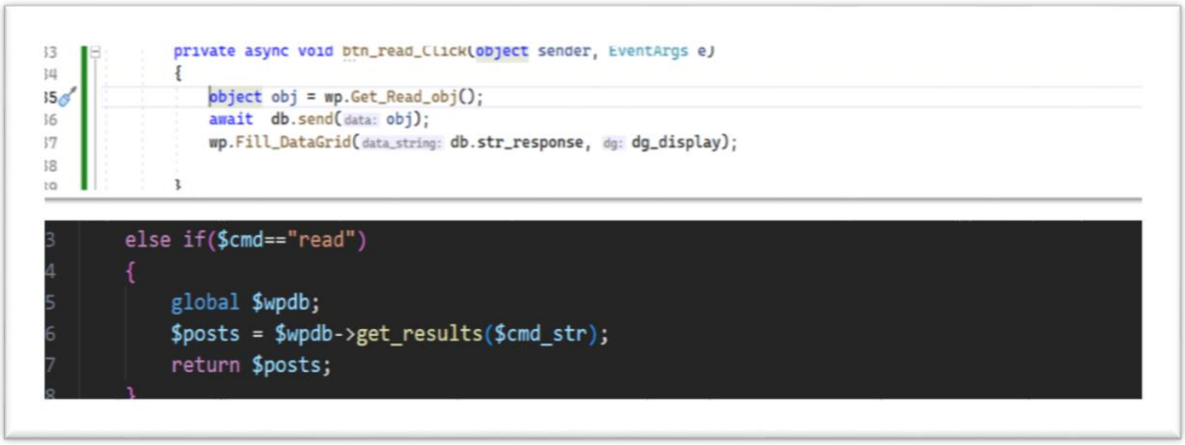


```
48    private async void btn_insert_Click(object sender, EventArgs e)
49    {
50        object obj = wp.Get_Insert_String(id: txt_user_id.Text, sys: txt_sys.Text, dia: txt_dia.Text, pul: txt_pul.Text);
51        await db.send(data: obj);
52        obj = wp.Get_Read_obj();
53        await db.send(data: obj);
54        wp.Fill_DataGrid(data_string: db.str_response, dg: dg_display);
55    }
```

```
35    else if($cmd=="insert")
36    {
37        $new_post = array(
38            'post_title'    => 'bp',
39            'post_content'  => $cmd_str,
40            'post_status'   => 'publish',
41            'post_author'   => 1,
42            'post_type'     => 'post',
43            'post_category' => array(4,4)
44        );
45        $post_id = wp_insert_post($new_post);
46        if ($post_id){
47            return "Insert Successfully";
48        } else {
49            return "Failed to insert post.";
50        }
```

Fig. 2: The insert functions.

Insert Post: figure 4. 2 depicts the insert function. This function adds the record to the WordPress database. The top portion, lines 48 to 54, is added to a button handler. At first, we create an insert string object providing record data. After that, we send it to the WordPress server. After inserting data, we read the post and displayed it to the data. The second part is handled by the word press engine. Once the request reaches the server, it parses the data, creates an associative array, and calls its inbuilt function "wp_insert_post()."
Read Post: figure 3 depicts the read function. In the first part, lines 13 to 19, the code must be added to a button function called "READ." At first, we create a read JSON object, which is sent to the server. Once the server returns the data, parse it and display it inside the data grid.



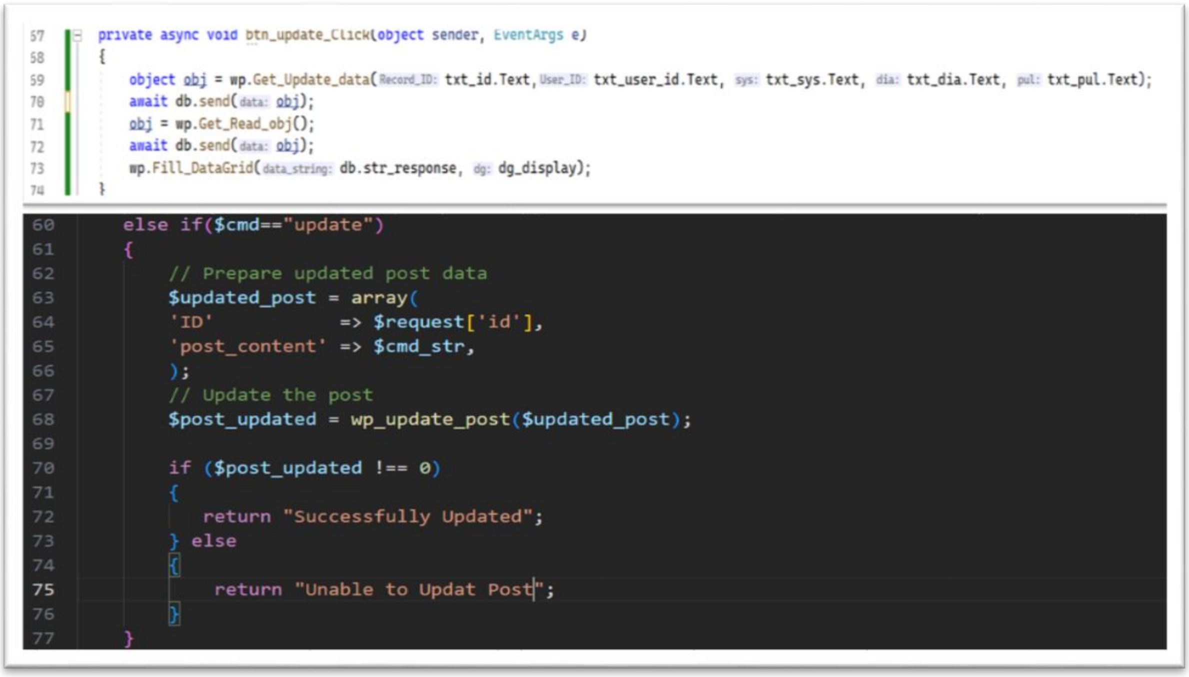
```
private async void btn_read_Click(object sender, EventArgs e)
{
    object obj = wp.Get_Read_obj();
    await  db.send(data: obj);
    wp.Fill_DataGrid(data_string: db.str_response, dg: dg_display);

}
```

```
else if($cmd=="read")
{
    global $wpdb;
    $posts = $wpdb->get_results($cmd_str);
    return $posts;
}
```

Fig. 3: The read function

Update Post: figure 4.4 depicts the update function. The first part is to add a button called “UPDATE” " Using the modified content, we create a JSON object. Then, it is sent to WordPress. Then, we read and display it to the data grid to see if it is updated. In the second part, the code is run by the server. Receiving the data from the client, it creates an associative array and sends it to the WordPress engine using the built-in function “**wp_update_post()**”. If the update succeeds, return ok; otherwise, return an error.

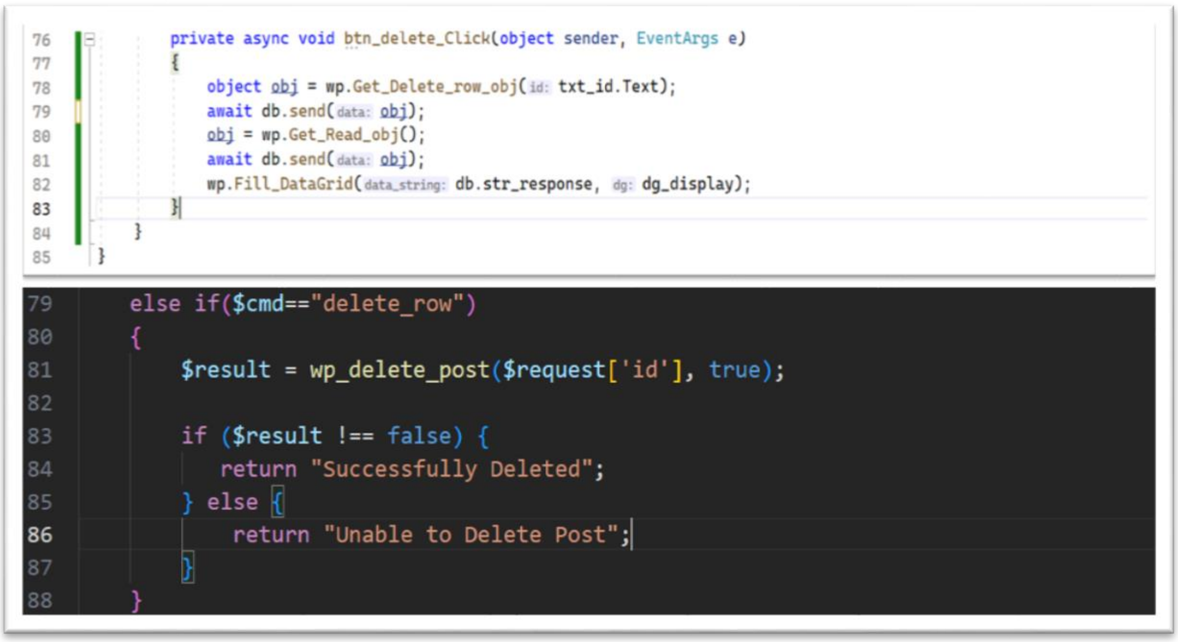
Delete Post: Figure 4.5 depicts the code to delete the post. We can delete any post using the post ID. The first part of the figure line 76 to 85 is the code we add to a button handler code called “DELETE”. Before pressing the delete button, we read the posts from the server. The record will be displayed in the data grid. Now click the row that we want to delete. It will update the textbox data from the grid row data. Now we press the “DELETE” button. the first action is to get a JSON object providing a post ID. That object is sent to the server. The server will delete the post. Then, read the post and display it on the data grid. We observe that the deleted data is not present in the data grid.



```
private async void btn_update_Click(object sender, EventArgs e)
{
    object obj = wp.Get_Update_data(Record_ID: txt_id.Text, User_ID: txt_user_id.Text, sys: txt_sys.Text, dia: txt_dia.Text, pul: txt_pul.Text);
    await db.send(data: obj);
    obj = wp.Get_Read_obj();
    await db.send(data: obj);
    wp.Fill_DataGrid(data_string: db.str_response, dg: dg_display);
}
```

```
else if($cmd=="update")
{
    // Prepare updated post data
    $updated_post = array(
    'ID'            => $request['id'],
    'post_content'  => $cmd_str,
    );
    // Update the post
    $post_updated = wp_update_post($updated_post);

    if ($post_updated !== 0)
    {
        return "Successfully Updated";
    } else
    {
        return "Unable to Updat Post";
    }
}
```

Fig. 4: The update functions.



```
        private async void btn_delete_Click(object sender, EventArgs e)
        {
            object obj = wp.Get_Delete_row_obj(id: txt_id.Text);
            await db.send(data: obj);
            obj = wp.Get_Read_obj();
            await db.send(data: obj);
            wp.Fill_DataGrid(data_string: db.str_response, dg: dg_display);
        }
    }
}
```

```
    else if($cmd=="delete_row")
    {
        $result = wp_delete_post($request['id'], true);

        if ($result !== false) {
            return "Successfully Deleted";
        } else {
            return "Unable to Delete Post";
        }
    }
```

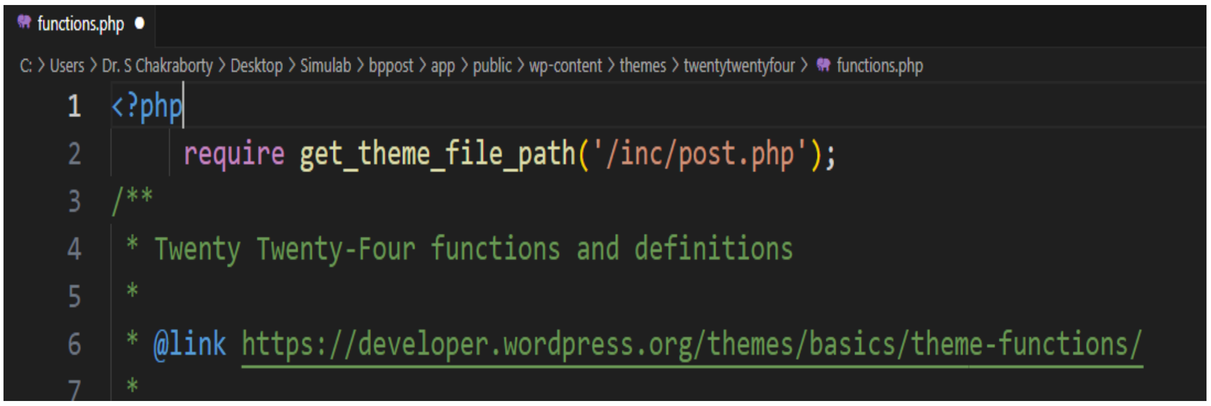
Fig. 5: The delete function.

5. EXPERIMENT :

Now, we can do some practical experiments or hands-on tests. We need to follow the following steps:

WordPress website modification:

1) First, we need to install local software in our system. It is available from "https://localwp.com/". Once it is downloaded, then we can install it. The installation procedure is available in the paper[x][x].
2) Download or clone the project from Github: https://github.com/sudipchakraborty/CRUD-Operation-On-WordPress-Posts-From-C-sharp-Over-REST-API.git.
3) Open local apps. Select the website we created. Click on "**Go to the site folder**."
4) Now copy the "**inc**" folder from the downloaded folder and paste it inside the path below: "C:\Users\xx\Desktop\xxx\xx\app\public\wp-content\themes\twentytwentyfour\"
5) under the above path, we found a file called "**functions.php**." Add the one line depicted in Figure 6, line number 2.


functions.php ●
C: > Users > Dr. S Chakraborty > Desktop > Simulab > bppost > app > public > wp-content > themes > twentytwentyfour > functions.php

```
1  <?php
2      require get_theme_file_path('/inc/post.php');
3  /**
4   * Twenty Twenty-Four functions and definitions
5   *
6   * @link https://developer.wordpress.org/themes/basics/theme-functions/
7   *
```

Fig. 6: "functions.php" file for modification.

Post Category Creation :

We will create a parent category named "**Health_Data**". Under this, we will make "**BP_Record**." That finally brings us to "**Health_Data/ BP_Record**."

1) Open the "**local**" apps interface. Click on the "**WP Admin**" button.

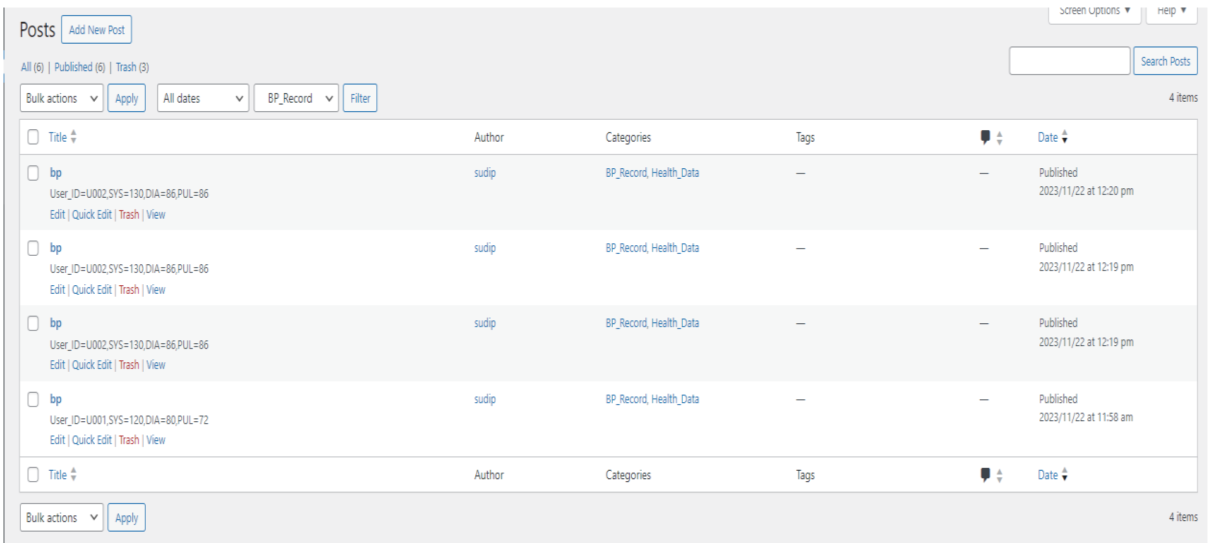
2) From the left side menu, under "**Posts**," click on "**Categories**." Under the "**Add New Category**," set Name: **Health_Data**, Slug: **health_data**, Parent Category: **None**, Description: This is the category for health record….
3) Press the "**Add New Category**" button. The category button will display on the right side.
4) Using the same way, again, set Name: **BP_Record**, Slug: **bp_record**, select Parent Category: **Health_Data**, Description: This is the sub-category for health records to store bp record data.

Manually add/insert data to the created Category:

1) Open the "**WP Admin"** panel. From the left side menu bar, click "**Posts**" under the Posts interface and click the "**Add New Posts**" button.
2) Set title: **bp**, content: User_ID=**U001, SYS=120, DIA=80, PUL=72**
3) from the right side, select **Health_Data** and **BP_Record** under the category.
4) Click the "**Publish**" button from the top right. And again, click on "**Publish**". The post will be saved to the WordPress database.
5) click the "**View Post**" button to view the post.
6) Like the above, a couple of entries may be added.

Display the manually from the WordPress admin panel:

1) Open the WordPress Admin panel.
2) From the left side, click on "**Posts**". All posts will be displayed on the right side.
3) Observe the data is displayed on the right side.
4) To view only our entered data, from the "**All Categories**" dropdown list, select "**BP_Record**" and click the "**Filter**" button.
5) Now, only our entered data will be displayed.

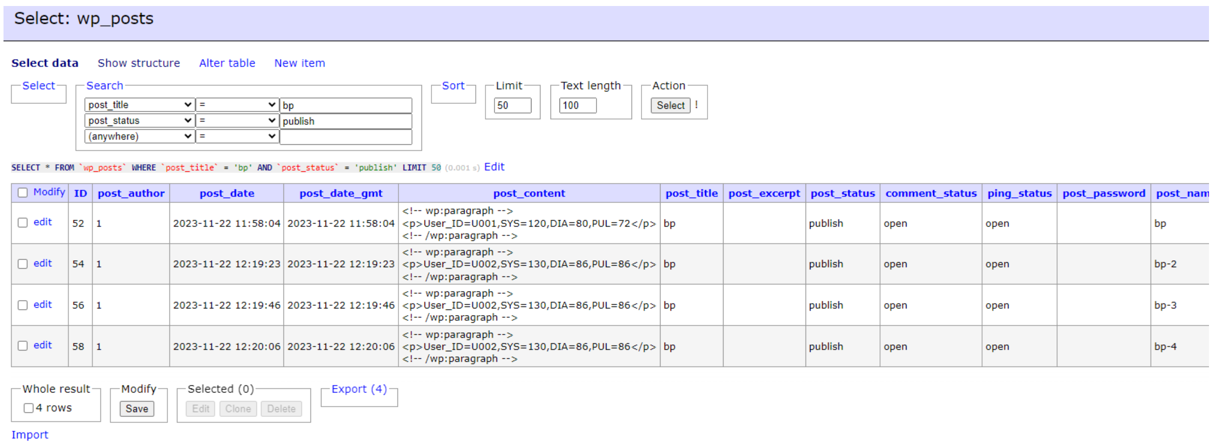


Title	Author	Categories	Tags		Date
bp User_ID=U002,SYS=130,DIA=86,PUL=86 Edit | Quick Edit | Trash | View	sudip	BP_Record, Health_Data	—	—	Published 2023/11/22 at 12:20 pm
bp User_ID=U002,SYS=130,DIA=86,PUL=86 Edit | Quick Edit | Trash | View	sudip	BP_Record, Health_Data	—	—	Published 2023/11/22 at 12:19 pm
bp User_ID=U002,SYS=130,DIA=86,PUL=86 Edit | Quick Edit | Trash | View	sudip	BP_Record, Health_Data	—	—	Published 2023/11/22 at 12:19 pm
bp User_ID=U001,SYS=120,DIA=80,PUL=72 Edit | Quick Edit | Trash | View	sudip	BP_Record, Health_Data	—	—	Published 2023/11/22 at 11:58 am
Title	Author	Categories	Tags		Date

Fig. 7: sample data display from WordPress admin panel

Display the manually from the local database admin panel:

1) From local, click on database.
2) Click on "**Open Adminer**"
3) Click on Select, which is beside the "**wp_post.**"
4) on the right side, all pots will be displayed.
5) Top side click on **search**, set "post_title" = **bp** and "post_status" = **publish.** Press Enter. And observe that only our data is visible now. The test result is depicted in the figure 5.3.
6) The added data will be filtered from entire posts.



post_title =	bp
post_status =	publish
(anywhere) =	
Limit	50
Text length	100

```
SELECT * FROM `wp_posts` WHERE `post_title` = 'bp' AND `post_status` = 'publish' LIMIT 50 (0.001 s) Edit
```

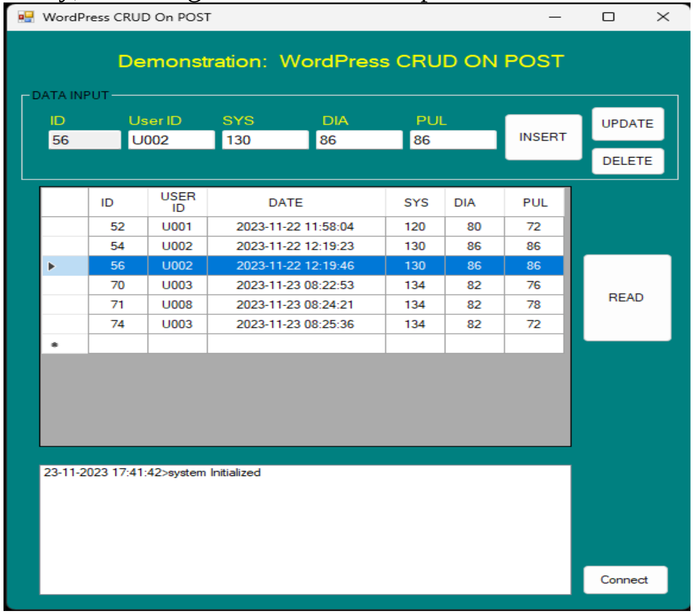
Modify	ID	post_author	post_date	post_date_gmt	post_content	post_title	post_excerpt	post_status	comment_status	ping_status	post_password	post_nam
edit	52	1	2023-11-22 11:58:04	2023-11-22 11:58:04	<!-- wp:paragraph --> <p>User_ID=U001,SYS=120,DIA=80,PUL=72</p> <!-- /wp:paragraph -->	bp		publish	open	open		bp
edit	54	1	2023-11-22 12:19:23	2023-11-22 12:19:23	<!-- wp:paragraph --> <p>User_ID=U002,SYS=130,DIA=86,PUL=86</p> <!-- /wp:paragraph -->	bp		publish	open	open		bp-2
edit	56	1	2023-11-22 12:19:46	2023-11-22 12:19:46	<!-- wp:paragraph --> <p>User_ID=U002,SYS=130,DIA=86,PUL=86</p> <!-- /wp:paragraph -->	bp		publish	open	open		bp-3
edit	58	1	2023-11-22 12:20:06	2023-11-22 12:20:06	<!-- wp:paragraph --> <p>User_ID=U002,SYS=130,DIA=86,PUL=86</p> <!-- /wp:paragraph -->	bp		publish	open	open		bp-4

Whole result: 4 rows

Fig. 8: sample data display from local adminer panel

How to Create a C# application:

1) Download MS Visual Studio and install it.
2) Create a C# Form Application.
3) Add a couple of buttons and the required control depicted in Figure 5.4.
4) Add bp_data.cs, global.cs, MSG.cs, WordPress_post.cs module inside the project, which are available from the downloaded project folder.
5) Using the NuGet package manager, install the package Newtonsoft.Json.
6) Add code to the UI control. Our downloaded project can be the reference.
7) Build the projects and run them. The application should look like the Figure 5.4.
8) Fill some data and press the "INSERT" button.
9) After a couple of data entries, press the "**READ**" button. It fetches the data from the server and displays it inside the data grid.
10) We can the modify entered data also. Click the row that we want to modify. The row data is available inside the input textbox. Then, change the data and press the "**UPDATE**" button. It will update to the server. Instantly, the data grid data will also update. Read from the cloud and display.



ID	User ID	SYS	DIA	PUL
56	U002	130	86	86

ID	USER ID	DATE	SYS	DIA	PUL
52	U001	2023-11-22 11:58:04	120	80	72
54	U002	2023-11-22 12:19:23	130	86	86
56	U002	2023-11-22 12:19:46	130	86	86
70	U003	2023-11-23 08:22:53	134	82	76
71	U008	2023-11-23 08:24:21	134	82	78
74	U003	2023-11-23 08:25:36	134	82	72

```
23-11-2023 17:41:42>system Initialized
```

Fig. 9: The application GUI

6. RECOMMENDATIONS :

- The project code is developed to show the communication flow of the project. For production, we need to add an exception-handling routine.
- Using this project, we can make custom IoT projects easily. Some IoT-related papers are available [11][12][13].

- ➢ If the project is not run as expected, debug according to data flow individually.

7. CONCLUSION :

WordPress is a sound content management system. Using post type, we can store our research data. Here, we demonstrated through practical examples how to do CRUD operation on the WordPress server using post type. Using the GUI written in C# language, we build the HTTP client application responsible for communicating with the WordPress database. We did the research work using a local server. Once work is over, we can deploy online without any trouble.

REFERENCES :

[1] Anuar, A. W., Kama, N., Azmi, A., & Mohd Rusli, H. (2022). Revisiting web application development with integrated records management is essential to using Re-CRUD. *Journal of Information and Knowledge Management (JIKM)*, *12*(1), 31-53. Google Scholar↗

[2] Leone, S., De Spindler, A., Norrie, M. C., & McLeod, D. (2013). Integrating component-based web engineering into content management systems. In Web Engineering: 13th International Conference, ICWE 2013, Aalborg, Denmark, July 8-12, 2013. Proceedings 13 (pp. 37-51). Springer Berlin Heidelberg. Google Scholar↗

[3] Bandirmali, N. (2018). mtCMF: A novel memory table-based content management framework for automatic website generation. *Computer Standards & Interfaces*, *58*(1), 43-52. Google Scholar↗

[4] Prstačić, S. (2021). Web application frameworks as reusable components (Doctoral dissertation, University of Zagreb. Faculty of Electrical Engineering and Computing. Department of Control and Computer Engineering). Google Scholar↗

[5] Leone, S., De Spindler, A., Norrie, M. C., & McLeod, D. (2014). Component-based web engineering using shared components and connectors. *Journal of Web Engineering*, 183-202. Google Scholar↗

[6] Murolo, A., & Norrie, M. C. (2015). Deriving custom post types from digital mockups. In Engineering the Web in the Big Data Era: 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings 15 (pp. 71-80). Springer International Publishing. Google Scholar↗

[7] Chakraborty, S., & Aithal, P. S. (2023). Industrial Automation Debug Message Display Over Modbus RTU Using C#. *International Journal of Management, Technology, and Social Sciences (IJMTS)*, *8*(2), 305-313. Google Scholar↗

[8] Chakraborty, S., & Aithal, P. S., (2022). A Practical Approach to GIT Using Bitbucket, GitHub, and SourceTree. *International Journal of Applied Engineering and Management Letters (IJAEML)*, *6*(2), 254-263. Google Scholar↗

[9] Chakraborty, Sudip, & Aithal, P. S. (2022). A Smart IDE for Robotics Research. *International Journal of Management, Technology, and Social Sciences (IJMTS)*, *7*(1), 513-519. Google Scholar↗

[10] Chakraborty, S., & Aithal, P. S., (2023). MVVM Demonstration Using C# WPF. *International Journal of Applied Engineering and Management Letters (IJAEML)*, *7*(1), 1-14. Google Scholar↗

[11] Chakraborty, S., & Aithal, P. S., (2023). Let Us Create an IoT Inside the AWS Cloud. *International Journal of Case Studies in Business, IT, and Education (IJCSBE)*, *7*(1), 211-219. Google Scholar↗

[12] Chakraborty, S., & Aithal, P. S., (2023). Let Us Create a Physical IoT Device Using AWS and ESP Module. *International Journal of Management, Technology, and Social Sciences (IJMTS)*, *8*(1), 224-233. Google Scholar↗

[13] Chakraborty, S., & Aithal, P. S., (2023). Let Us Create Multiple IoT Device Controller Using AWS, ESP32 And C#. *International Journal of Applied Engineering and Management Letters (IJAEML)*, *7*(2), 27-34. Google Scholar↗

****