CRUD Operation on WordPress Database Using C# And 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

Subject Area: 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.10197134
Google Scholar Citation: IJAEML

How to Cite this Paper:
Chakraborty, S., & Aithal, P. S., (2023). CRUD Operation On WordPress Database Using C# And REST API. *International Journal of Applied Engineering and Management Letters (IJAEML), 7*(4), 130-138. DOI: https://doi.org/10.5281/zenodo.10197134

International Journal of Applied Engineering and Management Letters (IJAEML)
A Refereed International Journal of Srinivas University, India.

Crossref DOI: https://doi.org/10.47992/IJAEML.2581.7000.0197

Received on: 15/10/2023
Published on: 23/11/2023

CRUD Operation on WordPress Database Using C# And 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: *IoT is famous for research with sensor data. The standard way is to keep sensor data inside IoT platforms like AWS, Google, and Microsoft Azure cloud. We must pay to keep or exchange the data among different systems. Here, we provide a different approach to keep our research data inside the cloud. We created a communication channel using the C# application and WordPress website database over REST API, which is now widespread. Our data format is JSON, which is the industry standard. Upload data to the website database using an HTTP client from the C# application. The uploaded data can be accessible from anywhere among the various systems. Our website runs round the clock, so our data is always available. This procedure might be cheaper than the available solution due to the use of our existing website database. Through the practical approach, we demonstrate how to do that. This procedure can be helpful for the researcher trying to keep their research data at a cheap cost inside the cloud. The complete project code is available to download, and the value addition to this work.*

Design/Methodology/Approach: *We first install "Local" software inside our system. On that, we installed a WordPress website. Then, we create a custom route for our experiment to differentiate from existing website content inside the server using PHP. Then, we added our module, which provides CURD operation. This module receives JSON-structured content, which is converted by the WordPress engine. The module parses the command and processes it accordingly. The response is returned to the client application. We created a C# application with a couple of GUI elements to interact with the user. We added a couple of modules using the Nuget package manager for REST API. After the application opens, the first action is to connect with the database. Once the database is available, CRUD operation can be executed.*

Findings/Result: *We tested the entire project using practical deployment. The performance is pretty good. A small quantity of network latency is present in our observation. The content updation time depends on website bandwidth, network traffic, the system's specification, and the number of running applications.*

Originality/Value: *Over the net, there is some practical documentation on C# REST API on the WordPress Website. Sometimes, researchers working on WordPress websites want to use database operation over REST API. Here, we demonstrate this through practical examples. So, it is a slightly different approach for quickly understanding the communication flow.*

Paper Type: *Experimental-based Research.*

Keywords: REST API in WordPress and C#, REST API Demonstration, CRUD Operation in WordPress Website. Database operation using REST API.

1. INTRODUCTION :

We research in various fields. In all research projects, we process data. Some research must store processed and unprocessed data inside the cloud to access other team members or provide global data. In this scenario, The general trend is to subscribe to the cloud space from well-known cloud providers like AWS, Microsoft, Google, etc., which are efficient and easy to deploy. The only thing is to pay for the cloud server. Generally, the cost varies due to data traffic and storage space. Sometimes, this cost

becomes overhead for the researcher. They keenly search for an alternate way to do the task or execute the research work. There is another way to do the same job without paying money. We can use that website database to store our data if we have our website.

Here, we demonstrate how to store our research data inside the WordPress website database. It is simpler than other available procedures, and we have complete control over the database and communication flow. The same procedure can also be applied to another website with little change. Most research institutes maintain a website that displays the various institutional research content. In the same database, we can use to store our data—the real-time data we can store without issues. The benefit of this process is to minimize the research cost. The deployment is also easy. The recurring fee is nill. Deployment and maintenance is also easy. We use local software for basic testing and code development. It is free and easy to use. Once the product development phase is over, we can deploy it inside the online website.

We create a data table inside the database programmatically from our application. We can also do a direct Admin panel provided by the website. Then, we communicate with the server from our C# REST client application. We need to provide server credentials like ID and Password When the client tries to communicate with the server. Once the connection is successful, we can operate several operations, Like CRUD operation, which are CRETE, READ, UPDATE, and DELETE.

2. RELATED WORKS :

Walker, C. et. Al. introduces the concept of a "Personal Data Lake," which serves as a unified storage facility for storing, analyzing, and querying personal data. It emphasizes the importance of data gravity pulling in data lakes to prevent them from becoming data swamps [1]. Kornienko D. et al. discuss the Single Page Application (SPA) architecture in the context of developing secure web services [2]. Luo, Y. et al. present insights into low-code development from a practitioner's perspective. It delves into the characteristics and challenges of low-code software engineering development [3]. Al Mahruqi, R. S., et. Al. presents a semi-automated framework for migrating web applications from SQL databases to document-oriented NoSQL databases [4]. Resceanu, I. C. et al. discuss the development of a framework for websites intended for an international audience [5]. Ferry E. et al. comprehensively evaluate the security aspects of the OAuth 2.0 framework [6]. Lemos, A. L. et al. surveyed techniques and tools used in web service composition [7]. Kaewprathum, T. P.'s paper analyzes the architectural aspects of retail omnichannel systems and the integration of cash IT point-of-sale software with e-commerce platforms [8]. Gagliardi, V. introduces the concept of decoupled Django architectures for building web applications with separate front-end and back-end components. It explains the principles and advantages of decoupled web development [9]. Gibbons K. et al. evaluate the security aspects of the OAuth 2.0 framework, focusing on its role in managing access and identity. It assesses the framework's security features and potential vulnerabilities [10]. Prstačić, S. explores the concept of web application frameworks as reusable components [11].

3. OBJECTIVES :

This research work aims to provide reference information on the WordPress website database used for data storage for our research work. Several documents are available over the net. Here, we provide practical examples so the researcher can integrate the WordPress database easily with little effort.

4. APPROACH AND METHODOLOGY :

Figure 1 depicts the complete block diagram of the projects. The central part of the architecture is the C# application. It coordinates the different functional blocks. The four module is connected to the main module. "**WordPress_REST.cs**" is used to communicate with the WordPress database. Inside the module, there are Separate functions for each activity. CREATE, READ, UPDATE, and DELETE. The "**project_BP.cs**" module is for the project-specific JSON object builder module. This module only varies from project to project. The rest of the module is the same for most of the projects. The "**Global. cs**" module is used to store the variable which is used across the different modules. The **GUI element** is used to interact with different commands. We use Datagrid to display fetched data in tabular format. It is also used to edit the data. The list box is used to display the status of different messages. The button is used to execute the command. The textbox is used to receive input from the user.

WordPress website: We worked with the default theme. However, we will use the existing website theme in a real website, which might not be an issue. The main point is registering a custom route with WordPress by adding a callback function. When our website gets a request, it triggers the callback function. Inside the callback function, we write our code, which handles the request of various database handling functions.

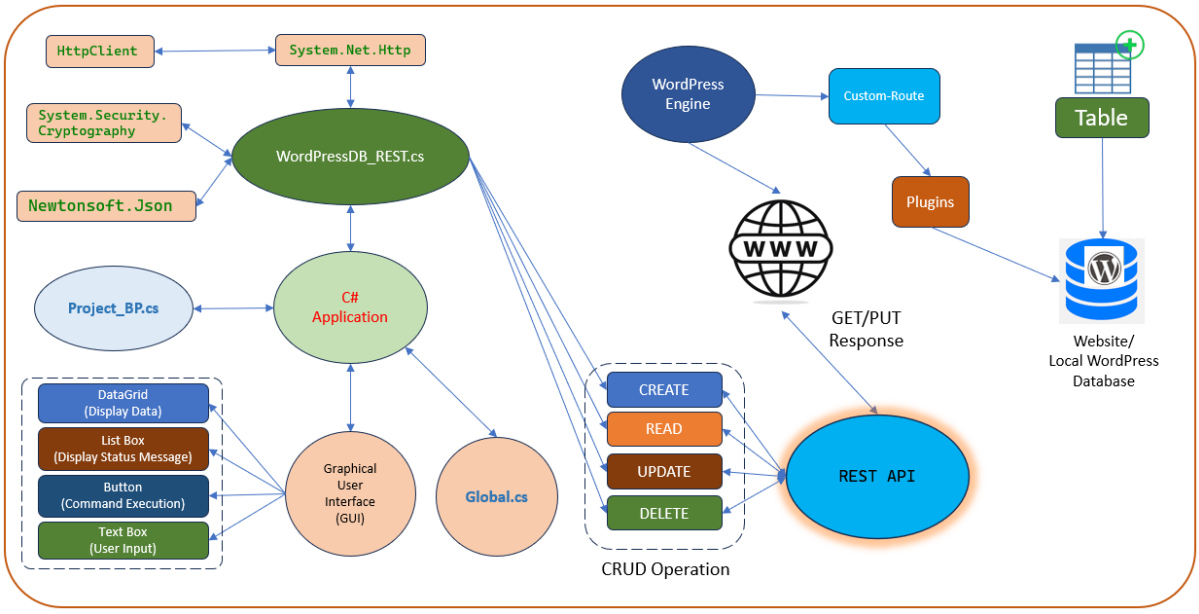

Fig. 1: Project block diagram

Table: The database table stores the data for our research work. We can create the table in different ways. The most common forms are either statically or programmatically. Statically means using the WordPress Adminer panel to create a table using variables from GUI control. Another good way is to create a table from the program. It is also called dynamic table creation. It has advantages, like changing table structure or variable type without logging into the website. We can delete or drop the table at any time.

The complete experiment we will do in local software. Everything development can be done using a local WordPress database. Once the experiment is finalized, we can take the website backup and restore it to the remote website. Several free plugins are available to create database backup and restore quickly without any issues. When the C# client requests some data. The query reaches the callback functions, and according to the function, the server replies to the client using JSON format.

5. EXPERIMENT :

Now, we will do some experiments for our experience. We need to follow the following steps:

Install WordPress: Open the link https://localwp.com/ in a browser. Click on the "**Download for free**" button. Select the platform, complete the form, and click "**GET IT NOW!**". One file will download; it looks like ***local-8.0.1-windows***. Double-click on the setup file and install the software.

Once the installation is finished, Click the "**WP Admin**" button from the local dashboard. From the left side menu, under the "**Appearance**," click on "**Themes**." We work with the 2023 theme. So make sure to "**Active: Twenty Twenty-Three.**"(Another theme is also pretty fine).

Create a file "***functions.php***" inside the C:\Users\..\LocalSites\xxxx\app\public\wp-content\themes\twentytwentythree\. Add the below code inside the "**functions.php,**" which is depicted in Figure 2.

C: > Users > sudip > Local Sites > simulab > app > public > wp-content > themes > twentytwentythree > functions.php

```php
<?php
    require get_theme_file_path('/inc/custom-route.php');

```

Fig. 2: Code example for functions.php

Create a folder "**inc**" inside the path: C: \Users \xxxx\ Local Sites\ xxxxx \ app \ public \ wp-content \themes\ twentytwentythree: Inside the "**inc**" folder, create a file "**custom.php**" add the code depicted in Figure 3: Figure 3 depicts the execution code.

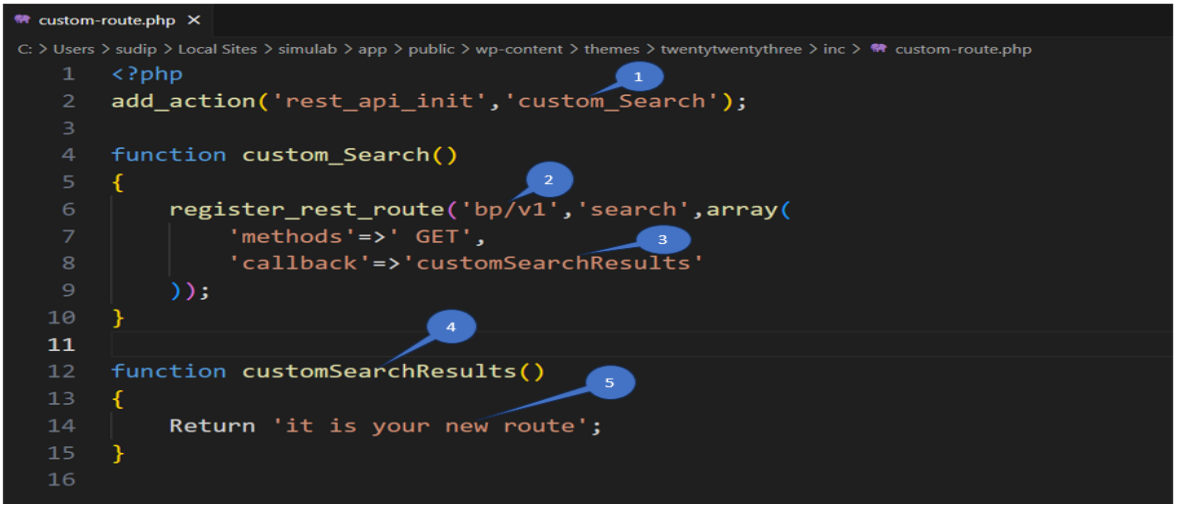

Fig. 3: Code example for custom-route.php

Now, we see the meaning of the code, which is marked in Figure 3.

1) This is the action function needed to register with WordPress.
2) This is the custom route we created.
3) the callback function name must also be registered with the WordPress engine.
4) This is the callback function. The callback function fetches when the server gets called.
5) This is where we place our actions to process the client's request.

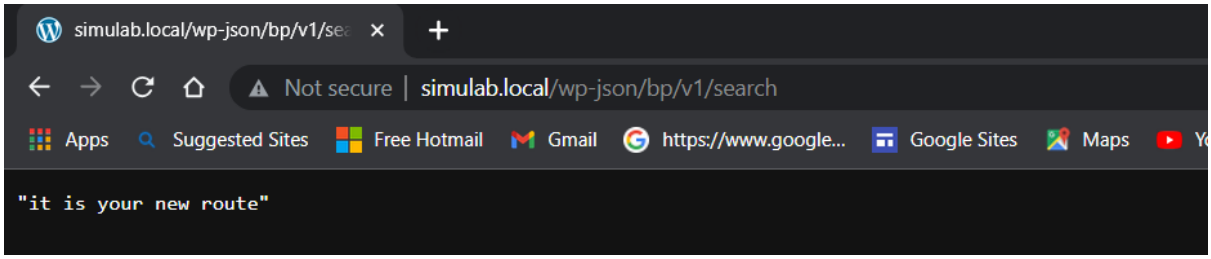

Fig. 4: Test code in the browser

After writing the code, save and close. Open the browser and type: http://xxxxx.local/wp-json/bp/v1/search. xxxxx is the website name. Figure 4 depicts the search result. Figure 4 gives the search result showed inside the browser.

Figure 5 depicts the code for the "connect" button. When we press the "connect," it will call DB. Connect (), which indicates in line number 35 to 40. Line number 59 to 81 indicates the code to connect with the server. At first, we are creating an HTTP client. Then it tries to authorization with the server. It was then using API url string connecting with the server. The server parses the command and responds, "OK". Receive the response code and verify the string. If it is "OK," it shows "connected."

```
35  private async void btn_connect_Click(object sender, EventArgs e)
36  {
37      await  db.Connect("http://simulab.local/wp-json/db/connect");
38      if (db.Connected) msg.push("Connected With Database OK");
39      else msg.push("Error!!! Not able to connect");
40  }
```

```
52  /// <summary>
53  /// @brief This function is used to check the existence of the server. it sends a string "connect" to the server.
54  /// if the server running, response with "ok" message.
55  /// @param apiUrl=API url string
56  /// @return void
57  /// </summary>
58  ///
    1 reference
59  public async Task  Connect(string apiUrl)
60  {
61      if (Connected) return;
62      HttpClient client = new HttpClient();
63      string base64Auth = Convert.ToBase64String(Encoding.ASCII.GetBytes($"{Uid}:{Pwd}"));
64      client.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Basic", base64Auth);
65      HttpResponseMessage response = await client.GetAsync(apiUrl);
66
67      Task.WaitAll(response.Content.ReadAsStreamAsync());
68      // Check if the request was successful
69      if (response.IsSuccessStatusCode)
70      {
71          // Read and display the response content
72          string content = await response.Content.ReadAsStringAsync();
73          content=content.Trim(new char[] { '\r', '\n', '"' });
74          if (content =="ok") Connected=true;
75          else                Connected=false;
76      }
77      else
78      {
79          Connected=false;
80      }
81  }
```

Fig. 5: the example code for connect button

```
57  private async void btn_create_table_Click(object sender, EventArgs e)
58  {
59      await db.send(bp.Get_Create_Table_obj());
60      if (db.str_response=="<Table Created>") msg.push("Table Created");
61      else msg.push("Error!!! Unable to create Table");
62  }
```

```
    1 reference
28  public bp_data Get_Create_Table_obj()
29  {
30      string createTableQuery =
31        "CREATE TABLE IF NOT EXISTS "+table+"( "+
32        "id mediumint(11) NOT NULL AUTO_INCREMENT, "+
33        "User_ID varchar(10) NOT NULL, "+
34        "DateTime datetime NOT NULL, "+
35        "SIS INT(3) unsigned NOT NULL, "+
36        "DIA INT(3) unsigned NOT NULL, "+
37        "PUL INT(3) unsigned NOT NULL, "+
38        "PRIMARY KEY id(id) )";
39
40      bp_data bp_Data = new bp_data();
41      bp_Data.cmd="create";
42      bp_Data.cmd_string=createTableQuery;
43      return bp_Data;
44  }
```

Fig. 6: The example code for programmatically Table creation

Create Operation: Figure 6 depicts the example code for the Table Create operation. To create a table dynamically, we added a button on the GUI called “CREATE” and added a few lines inside the button handler, which showed lines 57 to 62. At first, we build a JSON object for table creation, which resides in the **Project_BP.cs** module. Once we get the object, send it to the server. Line numbers 28 to 44 show how we build the Table creation object. The **bp_data** is the JSON object class that returns object data. Once the Table is created successfully, The server responds by “<Table Created>, " defined inside the WordPress server response handler php file.

```
99
        1 reference
100     private async void btn_read_all_Click(object sender, EventArgs e)
101     {
102         Check_Connection();
103         if (db.Connected)
104         {
105             await db.send(bp.Get_Read_obj());
106             bp.Fill_DataGrid(db.str_response, dg_display);
107             lbl_record_total.Text="Total Record Count="+dg_display.Rows.Count.ToString();
108         }
109     }
```

```
43      /// <summary>
44      /// @brief This asynchronous function is used to send the Data to the server and read the response from the server
45      /// @param data = json structured to send
46      /// @return the server response is stored inside the  "str_response" string veriable
47      /// </summary>
        10 references
48      public async Task send(object data)
49      {
50          HttpClient client = new HttpClient();
51          string base64Auth = Convert.ToBase64String(Encoding.ASCII.GetBytes($"{Uid}:{Pwd}"));
52          client.DefaultRequestHeaders.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("Basic", base64Auth);
53
54          //Tranform it to Json object
55          string jsonData = Newtonsoft.Json.JsonConvert.SerializeObject(data);
56
57          // Create the HTTP content from the JSON data
58          var StringContent content = new StringContent(jsonData, Encoding.UTF8, "application/json");
59
60          // Make the POST request
61          HttpResponseMessage response = await client.PostAsync(route, content);
62          // Check if the request was successful (HTTP status code 201 indicates success)
63          if (response.IsSuccessStatusCode)
64          {
65              str_response = await response.Content.ReadAsStringAsync();
66              str_response=str_response.Trim(new char[] { '\r', '\n', '"' });
67          }
68          else
69          {
70              str_response="Error";
71          }
72      }
```

Fig. 7: The Example code for read operation

Read Operation: Figure 7 depicts the read operation from the server. We added a button named "READ" and added code depicting line numbers 100 to 109. It first checks the connection. If the server is not connected, it will try to connect with the server by sending the string "**connect**." If the server responds with "OK," it is considered connected. Then, we get the JSON object for reading. Once the response is available, we parse the response and fill the data using the "**Fill_DataGrid**" function. Display the record count using the data grid row count. In the second part of the figure is the send function, which is asynchronous. At first, It creates an HTTP client. Then, it generates an authentication header. After that, Convert JSON content to HTTP content. Then, post the content using the async "POST" method.

```
        1 reference
77      private async void txt_insert_Click(object sender, EventArgs e)
78      {
79          DateTime myDateTime = DateTime.Now;
80          string sqlFormattedDate = myDateTime.ToString("yyyy-MM-dd HH:mm:ss");
81
82          bp_data b = bp.Get_Insert_String(txt_user_id.Text, sqlFormattedDate, txt_sys.Text, txt_dia.Text, txt_pul.Text);
83          await db.send(b);
84          if (db.str_response=="Failed to Insert data")
85          {
86              msg.push("Failed to Insert data");
87              return;
88          }
89          else
90          {
91              await db.send(bp.Get_Read_obj());
92              bp.Fill_DataGrid(db.str_response, dg_display);
93              dg_display.ClearSelection();
94              dg_display.Rows[dg_display.RowCount - 2].Selected = true;
95              dg_display.CurrentCell = dg_display.Rows[dg_display.RowCount - 2].Cells[0];
96              lbl_record_total.Text="Total Record Count="+dg_display.Rows.Count.ToString();
97          }
98      }
```

```
        1 reference
53      public bp_data Get_Insert_String(string id, string dt, string sys, string dia, string pul)
54      {
55
56          bp_data bp_Data = new bp_data();
57
58          bp_Data.cmd="insert";
59          bp_Data.cmd_string="";
60          bp_Data.tbl_name=table;
61          bp_Data.User_ID=id;
62          bp_Data.DateTime=dt;
63          bp_Data.SIS=sys;
64          bp_Data.DIA=dia;
65          bp_Data.PUL=pul;
66
67          return bp_Data;
68      }
69
```

Fig. 8: The Example code for Insert operation.

Insert Operation: Figure 8 depicts the function of inserting data into the database. We create a button called "INSERT" and add a couple of codes, shown in lines 77 to 98. At first, we are preparing SQL formatted date. Then, we send the insert data as a parameter to the **Get_Insert_String** function. In return, we get one JSON object. It is sent to the server. If data insertion is successful, it returns OK. After that, we read back and display the data to the data grid to confirm that the data is successfully inserted.

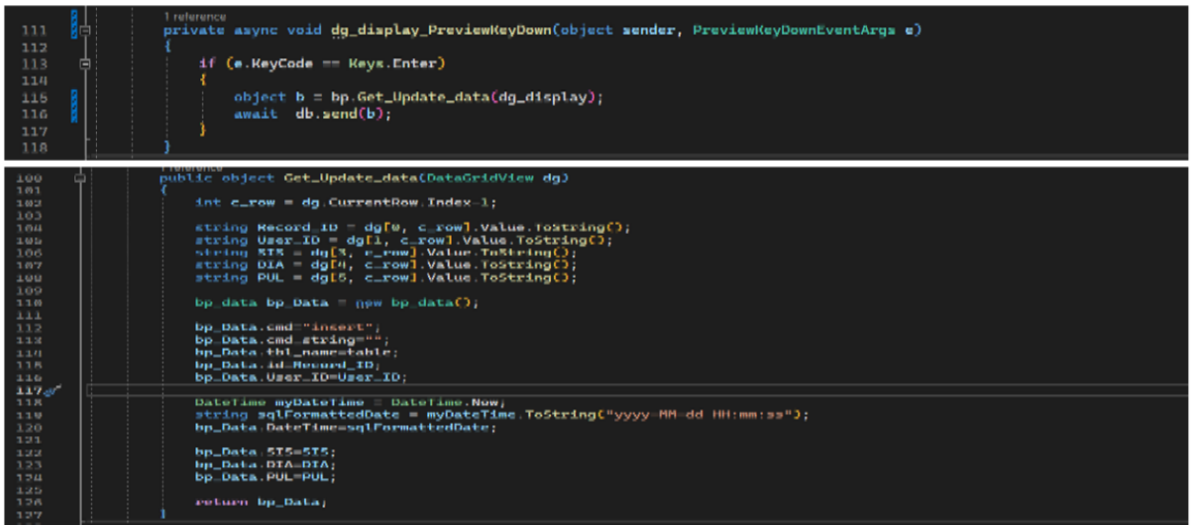

Fig. 9: the code for the database update operation

Update Operation: Figure 9 depicts the update operation. When we change some parameters inside the data grid and press enter, Datagrid **PreviewKeyDown** is triggered. At first, we get an updated JSON object from the selected DataGrid row. The second part of the figure shows the parsing of the data from the data grid and the creation of the JSON update object. Once we get the object, send it to the server.

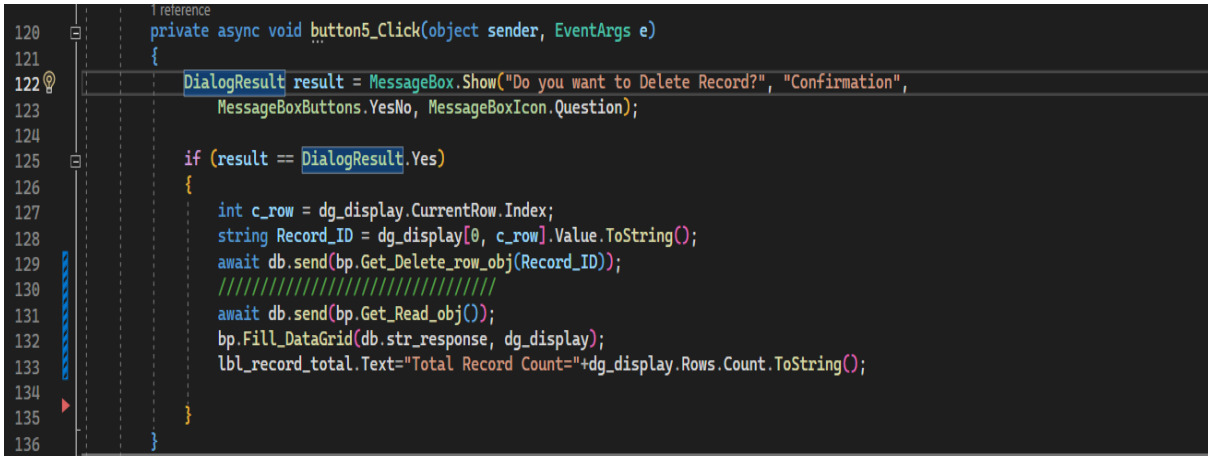

Fig. 10: the code for delete operation

Delete Operation: Figure 10 depicts the delete operation. When we want to delete any record, select the row in the data grid and press the delete record button. We added a couple of codes inside the button handler. At first, we display a confirmation message. Once confirmation is received from the user, create a JSON object using the record ID. Then, the complete structure is sent to the server for record deletion. After deletion. We read the record and display it to the data grid so that we can see that our record is deleted.

6. RECOMMENDATIONS :

- The project is available: https://github.com/sudipchakraborty/CRUD-Operation-On-WordPress-Database-Using-C-sharp-And-REST-API.git

- The code is not up to the production-worthy. Need to implement error handling. Due to keeping code simplicity, we did not include the try-catch block.

7. CONCLUSION :

REST API is popular nowadays. It is excellent and easy to implement. Using this API, we communicate with the WordPress website server and perform CRUD operations. The researcher who wants to integrate a database operation over REST API can get practical reference information here. The code is available for easy implementation.

REFERENCES :

[1] Walker, C., & Alrehamy, H. (2015, August). Personal data lake with data gravity pull. In *2015 IEEE Fifth International Conference on Big Data and Cloud Computing* (pp. 160-167). IEEE. Google Scholar↗

[2] Kornienko, D. V., Mishina, S. V., & Melnikov, M. O. (2021, November). The Single Page Application architecture when developing secure Web services. In *Journal of Physics: Conference Series* (Vol. 2091, No. 1, p. 012065). IOP Publishing. Google Scholar↗

[3] Luo, Y., Liang, P., Wang, C., Shahin, M., & Zhan, J. (2021, October). Characteristics and challenges of low-code development: the practitioners' perspective. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 1-11). Google Scholar↗

[4] Al Mahruqi, R. S., Alalfi, M. H., & Dean, T. R. (2019, November). A semi-automated framework for migrating web applications from SQL to document-oriented NoSQL database. In *CASCON* (pp. 44-53). Google Scholar↗

[5] Resceanu, I. C., Resceanu, C. F., & Simionescu, S. M. (2013, October). Developing a framework for websites with international audiences. In *2013 17th International Conference on System Theory, Control and Computing (ICSTCC)* (pp. 453-458). IEEE. Google Scholar↗

[6] Ferry, E., O Raw, J., & Curran, K. (2015). Security evaluation of the OAuth 2.0 framework. *Information & Computer Security*, *23*(1), 73-101. Google Scholar↗

[7] Lemos, A. L., Daniel, F., & Benatallah, B. (2015). Web service composition: a survey of techniques and tools. *ACM Computing Surveys (CSUR)*, *48*(3), 1-41. Google Scholar↗

[8] Kaewprathum, T. P. (2018). Architectural analysis of Retail Omni-channel and Cash IT Point-of-Sale software integration with E-commerce platform. Google Scholar↗

[9] Gagliardi, V. (2021). Introduction to the decoupled world. In *Decoupled Django: Understand and Build Decoupled Django Architectures for JavaScript Front-ends* (pp. 1-15). Berkeley, CA: Apress. Google Scholar↗

[10] Gibbons, K., Raw, J. O., & Curran, K. (2014). Security evaluation of the OAuth 2.0 framework. *Information Management and Computer Security*, *22*(3), 01-23. Google Scholar↗

[11] 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↗

****