How to Implement Cross-Origin Resource Sharing on CherryPy Platform of Python

Wasun Khan-Am[1]

[1]dept. of Information Systems
Faculty of Business Administration, RMUTT
Phatum Thani, Thailand
e-mail: wasun_k@rmutt.ac.th

Abstract— This article presents how to implement Cross-Origin Resource Sharing, aka CORS, on CherryPy platform of Python. This experiment is under the concept of RESTful. The tools of this experiment are python 3, CherryPy Framework version 18, SQLite 3, USBWebserver version 8.0, JQuery version 1.12, AJAX approach, and use Mozilla Firefox as a web browser. The hardware tool is a laptop computer. Four methods of HTTP methods, namely GET, POST, PUT, and DELETE; based on the concept of RESTful. Those methods were built on CherryPy platform before being tested in the next order. There are three outputs of this experiment that has been shown in this article. Firstly, the RESTful concept has been able to apply by using a Python 3 on CherryPy platform. Secondly, two methods, such as GET and POST, have been able to directly implement under CORS concept with a simple request. Thirdly, the rest method, PUT and DELETE, have been able to apply under CORS concept by using a preflight request.

Keywords: Cross-Origin Resource Sharing; WebAPI; RESTful; CherryPY; CORS

I. INTRODUCTION

Nowadays, Web is very famous, well-known and dispread around the world, there are several kinds of websites and several kinds of application relevance web. One kind to classify web application is use or non-use web server, then if the application uses a web server, it is called application on the web. If the application doesn't need a web server, it is called a web application.

According to the previous paragraph, web application and WebAPI is the most popular approach to implement web. There is a lot of programmer switch to a Web application, using languages such as nodejs, python, and Ruby. Some of them change themselves to create WebAPI. A WebAPI [1] is an application programming interface that provides a function to execute for the web server, web browser, application on the web, and web application. The well-known WebAPI is a RESTful WebAPI, this is a way for accessing a web-resources over the Internet [2].

The python language is one of popular language in the world, created by Guido van Rossum in 1991[3]. It is ease of use; its learning curve is low. It can use for creating a standalone application, both of text and window application; application for web, an application which runs on the web server; web application, an application as a web and running without the web server.

There is a lot of method and framework to create a web application in python. The easy one is using CherryPy [4] framework which declared itself as a minimalist python web framework. In addition, The CherryPy is a pythonic, object-oriented web framework that enables python programmers to create web applications as a similar way as they created the object-oriented Python program. Working with CherryPy Framework, a programmer will get a great result by using the smaller source code and less time.

The Representational State Transfer, in short REST [5], is a software architectural style which defines a new standard for creating web service that was developed by Roy Fielding. The web services implement REST architectural is called RESTful Web services or RWS an as known as a RESTful. The core concept of RESTful is to create a standard method of interoperability between server and client computers on the Internet by using common HTTP methods. A web programmer will be able to access web resources on the internet with four standard methods [6]. The four resource method was used to perform the desired transition are:

- GET for retrieving some information such as read one user or many users.
- POST for creating new information unit such as create one user.
- PUT for update some information such as update user information.
- DELETE for delete the information unit such as delete one user.

Notice that, the main concept of RESTful is involved with the way to manipulate an information unit such as create, read, update, and delete information that was stored in the database. Therefore, the application that implemented RESTful need to had a database to operate also.

SQLite [7] is a database including very small, great powerful, free of charge, SQL embedded, famous, and

easy to use. It is quite pretty good for use in a small and non-complicated project. It also is very suitable for use in test projects such as this experiment.

Cross-Origin Resource Sharing [8, 9], in short CORS, is a mechanism for working in crossing domain for web browsers, additional HTTP Header that let browser running an application at one origin (domain) have permission to access targeted resources from other servers at a different origin (different domain, protocol, and port) than its own origin. The word 'Crossing Domain' or 'cross-origin' was including any kind of host connection such as changing port with the same host, changing host in sending a request. The CORS was developed by World Wide Web (WWW) consortium [9] for web security reason.

The user agent processing model [9] has been described with two major algorithms such as a simple cross-origin request and a cross-origin request with preflight. The generic cross-origin request algorithm defined as *fetch the request URL from origin source origin using referrer source as override referrer source with the manual redirect flag set, and the block cookies flag set if the omit credentials flag is set. Use method request method, entity body request entity body, including the author request headers and include user credentials if the omit credentials flag is unset.* [9]

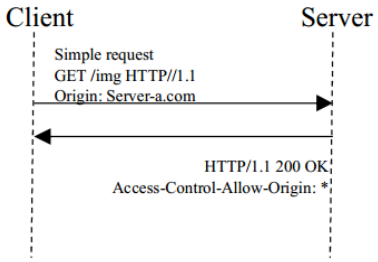

Figure 1. Simple Request.
Adapted from [8]

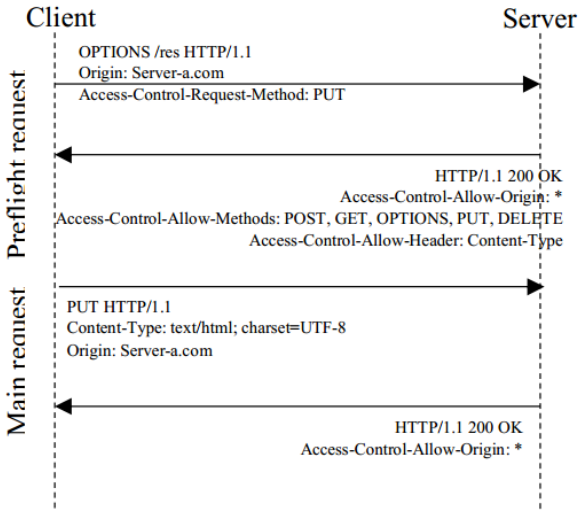

Figure 2. Preflight Request.
Adapted from [8]

The simple request [8] is the one that meets the method including GET, HEAD, and POST. The diagram of the simple request shown in figure 1.

The preflight request [8] is starting by sending an HTTP request with OPTIONS method to the resource on the other source, in order to determine whether the actual request is safe for sending. This diagram were presented in figure 2

In recent years, AJAX [10, 11], stand for **A**synchronous **J**avaScript **A**nd **X**ML, is a very popular approach that is implemented in client-site. The core benefit of AJAX is to make a request to the server without reloading the page. The term AJAX was coined by Jesse James Garrett [10], which is use a number of existing technologies together including: HTML, XHTML, CSS, JavaScript, DOM, XML, XSLT and the most important is the XMLHttpRequest Object. The shorten-pathway for implementing AJAX is using a JQuery [12]. JQuery is a JavaScript Library that provided a lot of functions to utilize programming in JavaScript Language.

According to the previous paragraph, this article set up an experiment for finding out how to implement Cross-Origin Resources Sharing on CherryPy framework of Python in a lot of causes such as how to create RESTful method and how to configure a CherryPy; and shows the result of an experiment by using web console of Mozilla Firefox.

II. Experiment Design

The procedure of this experiment consists of eight steps as following.

Step 1) Design table to store user data named userinfo.

Step 2) applies CherryPy tutorial 9 entitle "Data is all my life", to create source code with RESTful API including GET, POST, PUT and DELETE.

Step 3) create an index.html file with AJAX to apply HTTP RESTful.

Step 4) testing the program using the Mozilla Firefox web browser as a tool until it runs correctly.

Step 5) move an HTML file from CherryPy framework to root directory of USBWebserver

Step 6) test the new HTML file for calling RESTful API in step 3.

Step 7) if any problem, debug them and retest step 6 again until no problem occurrence.

Step 8) Document and create a report.

III. Result

The results of this experiment are divided into four parts: the design of the data table, server programming, client programming, and program test results

A. Table Design

In programming, table userinfo was used for developing the application of the RESTful method. The result of table userinfo design shows as:

TABLE I. USERINFO SCHEME

Column Name	Data type	Size	Note
User_name	TEXT	20	PK
Pass_word	TEXT	20	
Full_name	TEXT	50	
Level	INTEGER		

Table I shows the design of the table named 'userinfo' that consists of four columns including User_name, Pass_word, Full_name, and Level. The data types of each column are TEXT, TEXT, TEXT, and INTEGER respectively.

B. WebAPI programming

In this experiment, there are four methods was created with the CherryPy framework: GET, POST, PUT, and DELETE. The details of each method are shown as follows.

POST Method

```
def POST(self, un, pw, fn, ad):
    with sqlite3.connect(DB_STRING) as c:
        c.execute("INSERT INTO userinfo VALUES (?, ?, ?, ?)", [un, pw, fn, ad])
        print(str(c.total_changes), "record inserted.")
        return json.dumps({"insert_row": c.total_changes}).encode('utf-8')
```

The POST method was worked by taking all data from the user and inserts them into the table.

PUT Method

```
def PUT(self, un, pw, fn, ad):
    with sqlite3.connect(DB_STRING) as c:
        print ("Use PUT method")
        c.execute("UPDATE userinfo SET pass_word=?, full_name=?, level=? WHERE user_name=?",
          [pw, fn, ad, un])
        print(str(c.total_changes), "record updated.")
        return json.dumps({"update_row": c.total_changes}).encode('utf-8')
```

The PUT method work as takes all data from the user then use SQL update command for updating the table with the matched given a username.

GET Method

```
def GET(self, un="no_data_coming" ):
    with sqlite3.connect(DB_STRING) as c:
        if ( un == "no_data_coming"):
            r = c.execute("SELECT * FROM userinfo")
            myresult = r.fetchall()
            m = []
            for x in myresult:
                m.append({'user_name' : x[0], 'pass_word': x[1],
                  'full_name': x[2], 'level': x[3]})
            return json.dumps(m).encode('utf-8')
        else:
            sql ="SELECT * FROM userinfo WHERE user_name='"+un+"'"
            r = c.execute(sql)
            x    = r.fetchone()
            return json.dumps({'user_name' : x[0], 'pass_word': x[1],
                'full_name': x[2], 'level': x[3]}).encode('utf-8')
```

The GET method was created to carry out two major works. The first issue is to display all the data from the table. Another issue is to receive one parameter as a user name, the primary key of the table, and use that information to search the matching row in the table. Then send search results back to users.

DELETE Method

```
def DELETE(self, un=None):
    with sqlite3.connect(DB_STRING) as c:
        un = cherrypy.request.headers['id']
        print({"delete" : un})
        c.execute("DELETE FROM userinfo WHERE user_name='"+un+"'")
        print(str(c.total_changes), " deleted")
        return json.dumps({"delete" : un,
          "number_change" : c.total_changes}).encode("utf-8")
```

The DELETE method starting from takes a username that user wants to delete. Then delete the rows that match given a username.

OPTIONS Method

```
def OPTIONS(self, un=None):
    return json.dumps({}).encode('utf-8')
```

The OPTION method in CherryPy is an extra-method that use to response the HTTP request with the OPTIONS method for the preflight request. The objective of this method is to send an allow method to the client.

The next code is the configuration of /user in CherryPy.

```
'/user': {
    'request.dispatch': cherrypy.dispatch.MethodDispatcher(),
    'tools.response_headers.on': True,
    'tools.response_headers.headers': [('Content-Type', 'application/json'),
     ('Access-Control-Allow-Origin','*'),
     ('Access-Control-Allow-Methods', 'DELETE, GET, HEAD, OPTIONS, POST, PUT'),
     ('Access-Control-Allow-Headers', 'crossdomain,withcredentials,Content-Type,id')],
    'tools.encode.on': True,
    'tools.encode.encoding': 'utf-8'
},
```

The previous configured code shown how a server is controlled for incoming requests such as the allowed domains, the accepted methods, and the accepted header. All controlled data will be sent to the client with the following header include control-allow-origin, control-allow-method, and control-allow-header. The meanings of the controlled data as shown in the above configuration code are: the server accepts a request from all domain (*), there are six methods allow including DELETE, GET, HEAD, OPTIONS, POST, and PUT; there are four headers allow including crossdomain, withcredentials, Content-Type, id.

C. *Client Programming*

For simple request, the client programing was developed by using ajax approach. The part of program shows in the next paragraph.

```
$.ajax({
    "url" : "http://localhost:8080/user",
    type: 'GET',
    crossDomain: true,
    beforeSend: function(xhr){
        xhr.withCredentials = true;
    }
})
.done(function(string) {
    a = string.length;
    alert('Object: '+a+'Record');
    string_1 = '';
    for (i=0; i< a; i++) {
        line = string[i].username +'|'+string[i].passwd+'|'+
               string[i].fulname+'|'+string[i].address;
        line = line +'<br/>';
        string_1 = string_1 + line;
    }
    document.getElementById('showUser').innerHTML = string_1;
});
```

In previous code, the code implements ajax() function of jQuery with four parameters including url with *http://localhost:8080/user* value , type with *get* value, crossDomain with *true* value, and running function for beforeSend parameter.

For preflight request, the client program was created by using ajax approach also. This program part show in next paragraph.

```
$("#doupdate").click(function(e) {
    var xhr = new XMLHttpRequest();
    xhr.open('PUT', 'http://localhost:8080/user');
    xhr.setRequestHeader("crossDomain",true);
    xhr.setRequestHeader("withCredentials",true);
    xhr.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded; charset=UTF-8");
    xhr.onreadystatechange = function() {
        if (this.status == 200 && this.readyState == 4) {
            console.log('response: '+ JSON.stringify(responseText));
        }
    };
    body = "un="+$("input[name='uun']").val()+"&pw="+
           $("input[name='upw']").val()+"&fn="+
           $("input[name='ufn']").val()+"&ad="+
           $("input[name='uad']").val();
    console.log(body);
    xhr.send(body);
})
```

Due to preflight request programming, the server needs to be contacted twice, the first for the OPTIONS request, and the second for the PUTS request. Programming has changed to use. XMLHttpRequest object instead.

D. *CORS Tesing*

The CORS test operates in two type of request: a simple request testing, including GET and POST methods, and a preflight request testing for the PUT and DELETE methods. The result of simple request testing showed in next figure.

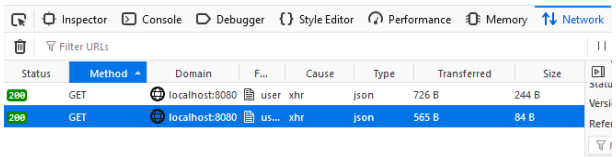

Figure 3. Monitor of Simple Request.

Figure 3 shows that the user agent sends the GET method to a different origin of the current webpage, localhost with port 8080 that is a default port for CherryPy, then gets the response back with status 200, OK status. The request header that was sent by XHR shows in the next figure.

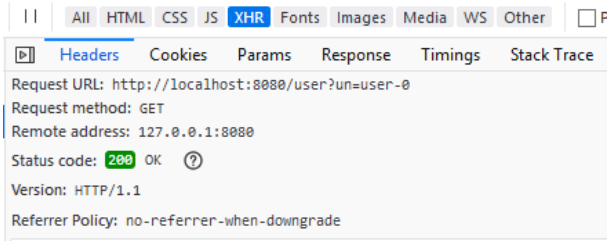

Figure 4. Request Header that was sent by XHR

Figure 4 shows that the detail of the request header are: the URL is http://localhost:8080/user?un=user-0, the method is GET, and the version of HTTP is HTTP/1.1. And the request was sent by XHR.The response header corresponded with this request showed in the next figure.

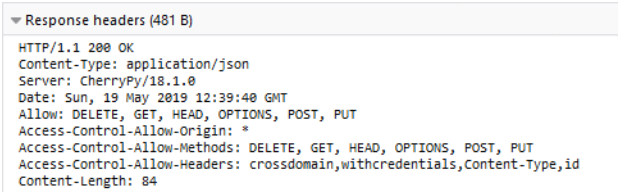

Figure 5. Response header corresponed to simple request.

The response header shown in Figure 5 provides information about the allowed rule of the server. In figure about, server accessing is allowed from every domain (*), the DELETE, GET, HEAD, OPTIONS, POST, and PUT are allowed to use, and the request header including crossdomain, withcredentials, Content-Type, and id are allowed and accepted.

In preflight request testing, the request with the PUT method was selected as an instance of the test. The test output is presented in the next figure.

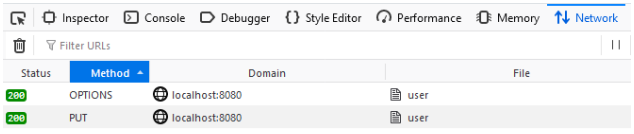

Figure 6. Monitor of Preflight Request.

In figure 6, the user agent sent a request with OPTIONS and PUT method to a different origin of the current webpage then the server sent the response back to a user agent with header contents status 200 or OK status.

The XHR header of request with OPTIONS method showed in next figure.

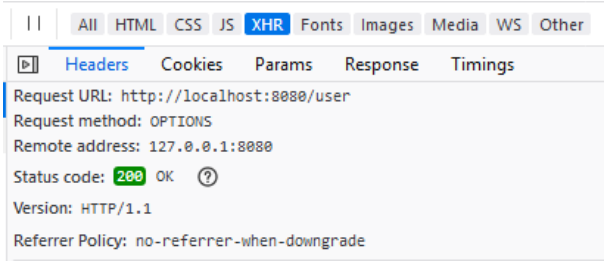

Figure 7. OPTIONS request header.

The request header with OPTIONS method that composes with URL, method, address, version and status code. After the server receives a request, the response is sent back to a user agent with the following header.

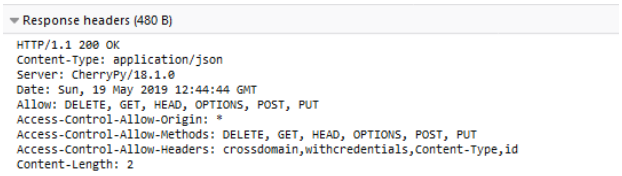

Figure 8. OPTIONS response header.

When the user agent received a response with status OK back from the server, the user agent must check an allowed domain list first. if the current domain is not in the list, the user agent must stop sending any request to the server and raise an error. after domain validation, the method list validation was conducted. By verification allowed method list, if found that no PUT method in that list then the user agent stops sending a request to the server and raises an error. In other cases, the user agent must send the request with the PUT method to the server to continuous preflight test. consequently, the request header with the PUT method is shown in the next figure.

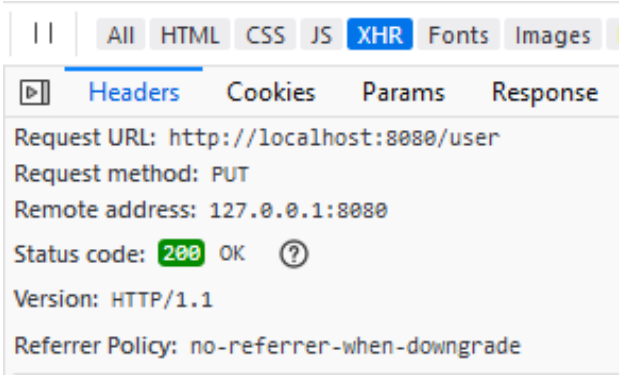

Figure 9. XHR header of PUT Request.

Figure 9 shows the request by the PUT method sent by XHR. Next figure shows some part of response header that corresponded with the request with PUT method

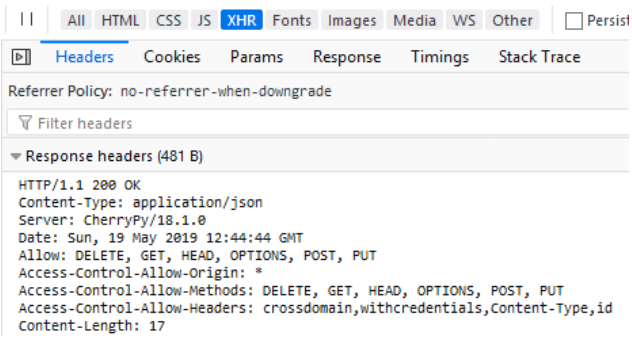

Figure 10. PUT response header.

The response header of request with PUT method looks similar like a response header of request with OPTIONS method that provides any information about all allowed rule of the server.

IV. Summary and Discussion

The result of this experiment reveals the connection between client and server during CORS working. The simple request and preflight have been applied in each case depend on the HTTP method. The key of work is response header which should be allowed the target client from other domain accessing resources under server. The security of connecting is necessary also, the server should have a strong policy such that from where? and which resources? need to be secured. The client can apply ajax() function of jQuery for simple request, But it can't apply for preflight request. the XMLHttpObject was prefer for preflight request.

Although all four main HTTP method was applied in this experiment, I found that firstly, code implementation in client side is limit with ajax() function of jQuery, code implementation by using XMLHttpdRequest object is a best choice and secondly, the form of WebAPI in this experiment is not a full pattern of RESTful. The implementation of RESTful in this research is limit like a tutorial of CherryPy [4]. Then, I recommend that in future research should apply URL friendly in next experiment.

References

[1] Kingleo713, MDN Web docs "Web APIs", Feb 28, 2019 Avialable: https://developer.mozilla.org/en-US/docs/Web/API, [Online] [Accessed: Mar. 3, 2019]

[2] Wikipedia, 'Web API', Feb. 28, 2019, Avialable: https://en.wikipedia.org/wiki/Web_API [Online][Accessed: Mar. 3, 2019]

[3] Python Software Foundation, Python 3.7.3 documentation, Aviable: https://docs.python.org/3/ [Online] [Accessed : Mar 3, 2019.

[4] CherryPy Team. CherryPy "CherryPy – A Minimalist Python Web Framework", 2019. Available: https://docs.cherrypy.org/en/latest/ [Online] [Accessed: Mar.3, 2019]

[5] D. Booth H. Haas, F. McCane, E. NewCommer, M. Champion, C. Ferris, D. Orchard "Web Services Architecture" World Wide Web Consortium, 11 Feb. 2004, Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest [Online] [Accessed: Mar. 3, 2019]

[6] R. Fielding, and J. Reschke, RFC7231 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Internet Engineering Task Force, June 2014.

[7] SQLite, 'SQLite Document'. Available: https://www.sqlite.org/docs.html. [Online][Accessed: Mar 23, 2019]

[8] sideshowbarker, 'MDN Web Docs, Cross-Origin Resource Sharing (CORS) – HTTP | MDN', Apr 12, 2019, 5:31:26 PM. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Online][Accessed: Mar 3, 2019]

[9] A. V. Kesteren, 'World Wide Web Consortium Cross-Origin Resource Sharing', Jan 16, 2014. Available: https://www.w3.org/TR/cors/. [Online] [Accessed: Mar 3, 2019]

[10] Mdnwebdocs-bot. 'Ajax – Developer guides | MDN', Mar 23, 2019. Available: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX [Online] [Accessed: Apr 10, 2019]

[11] Mdnwebdocs-bot, 'Getting Started –Developer guide', Mar 18, 2019. Available: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started [Online] [Accessed: Apr 10, 2019]

[12] JS fundation. 'JQuery API Document', 2019. Available: https://api.jquery.com/ [Online] [Accessed: May 20, 2019]