Table of Contents

System Design & Dev.

BioStar 2 API Quick Start Guide

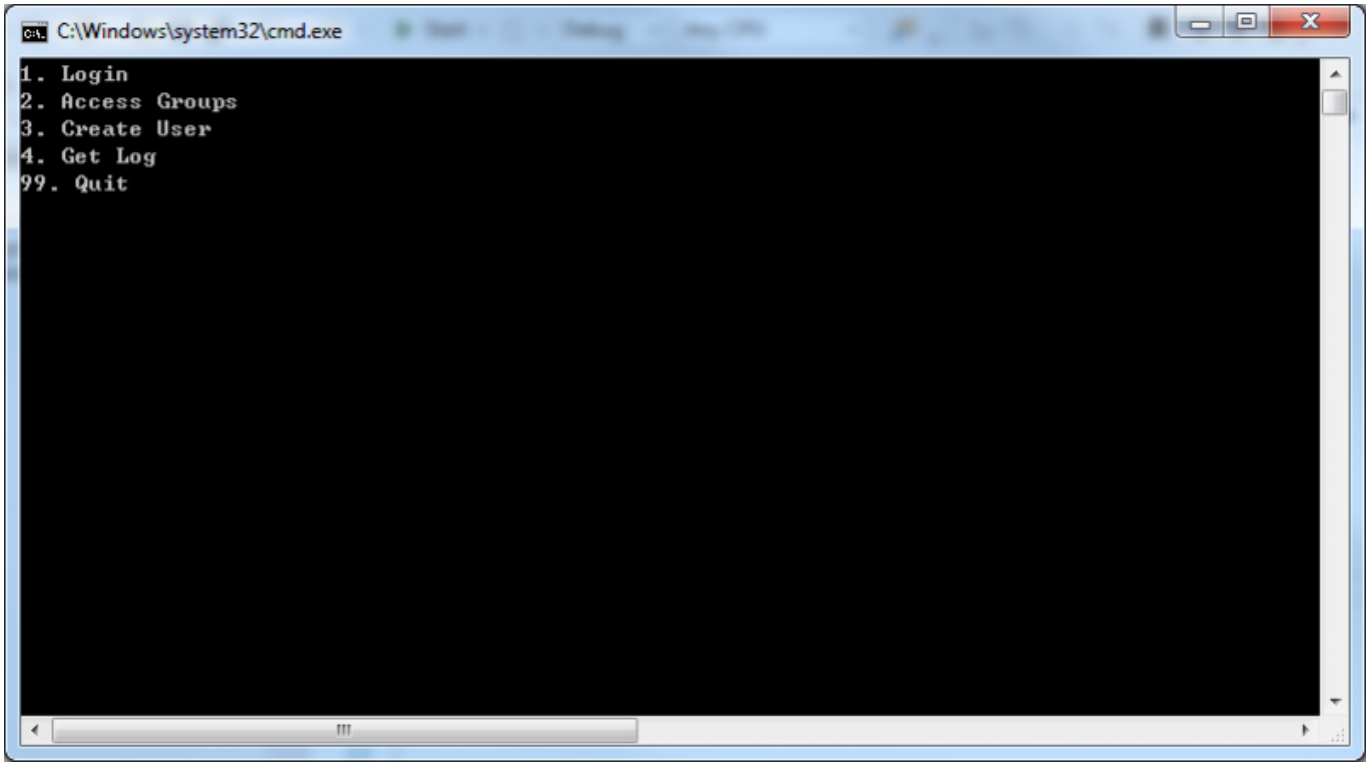
Introduction

There are two ways to utilize BioStar API. One is called Web API(via BioStar Cloud Server) and Another one is called Local API. **We recommend that you should use Local API.** After you install BioStar 2 API Server, you can see the documentation of BioStar API which provides more detailed information on how to use BioStar API.

In this article, I'm going to take a closer look at a sample application that I made for those who are familiar with C# or standalone Windows application. Since BioStar API is RESTful API, those who are not familiar with RESTful API might have difficulties implementing their own applications with BioStar API. Therefore, I'd like to guide them to kick-start their development with the sample application in this article.

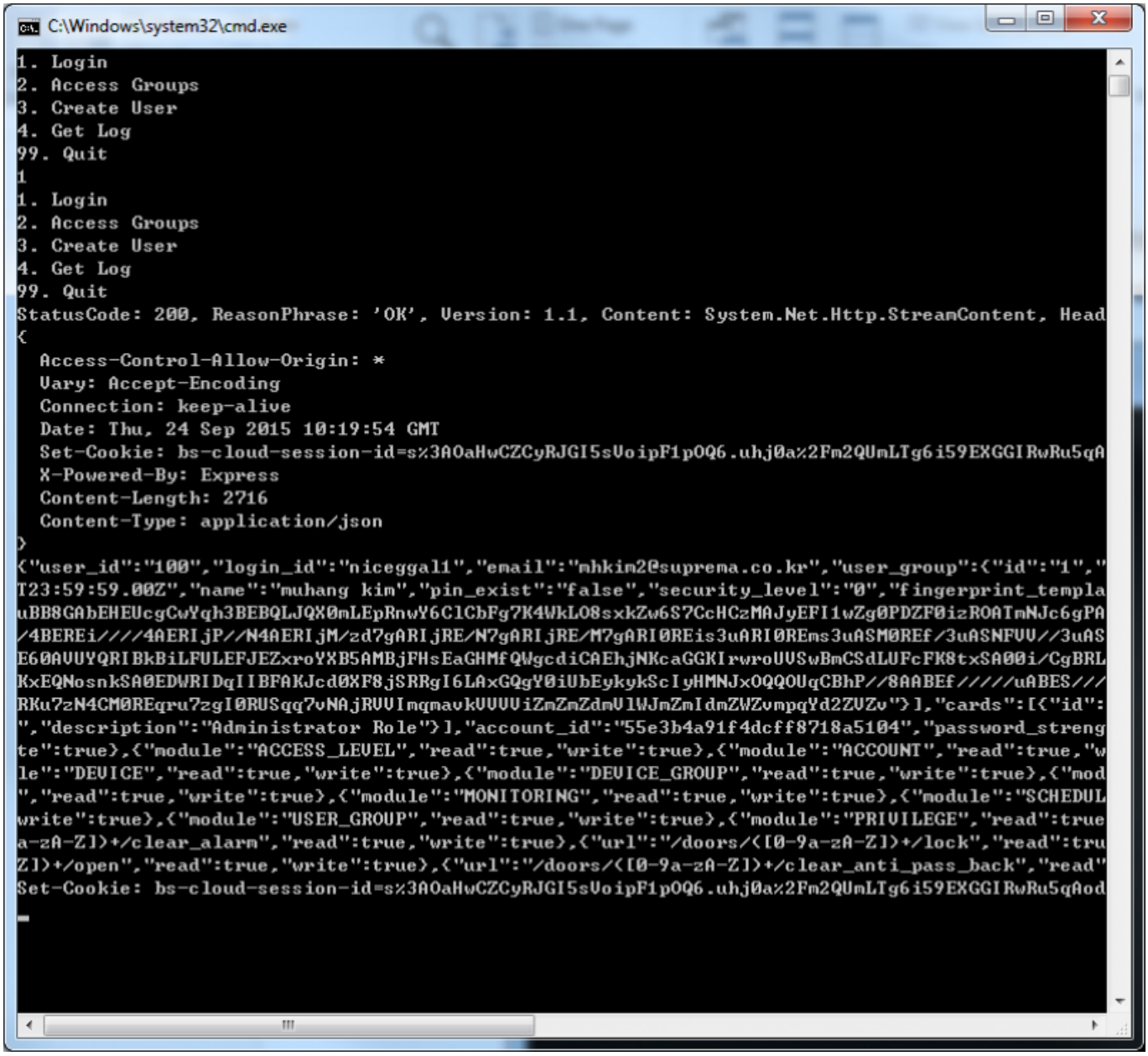
Features

This sample application is a Visual C# console application and includes four basic functions: login, user creation, access group retrieval and log retrieval.



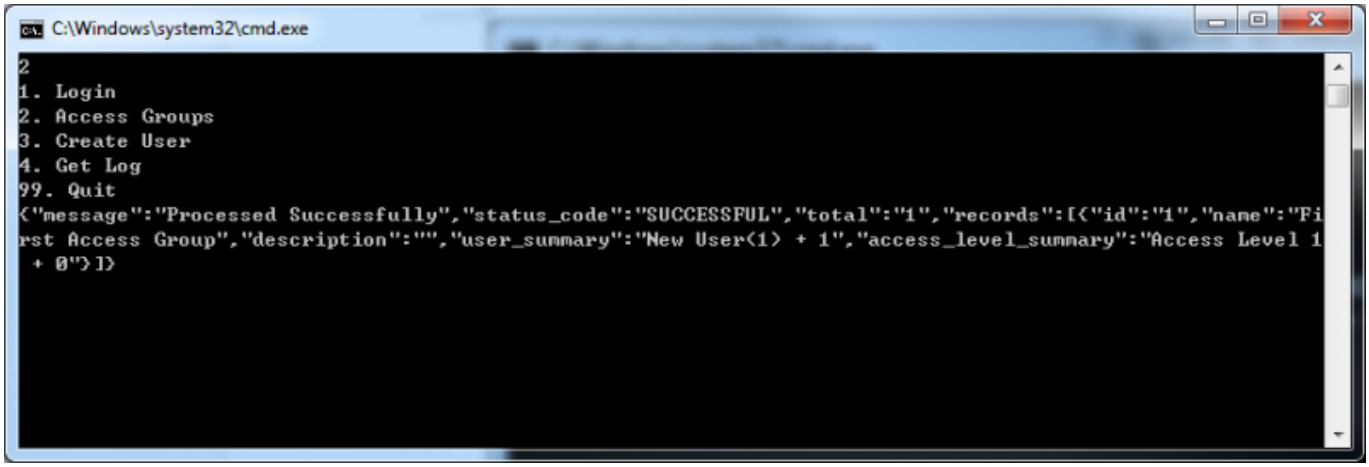
[Figure 1. When you've launched the sample application]

First, you have to log in before using any other functions. If you type '1' and press Enter, the sample application logs into your local BioStar server through BioStar API server.



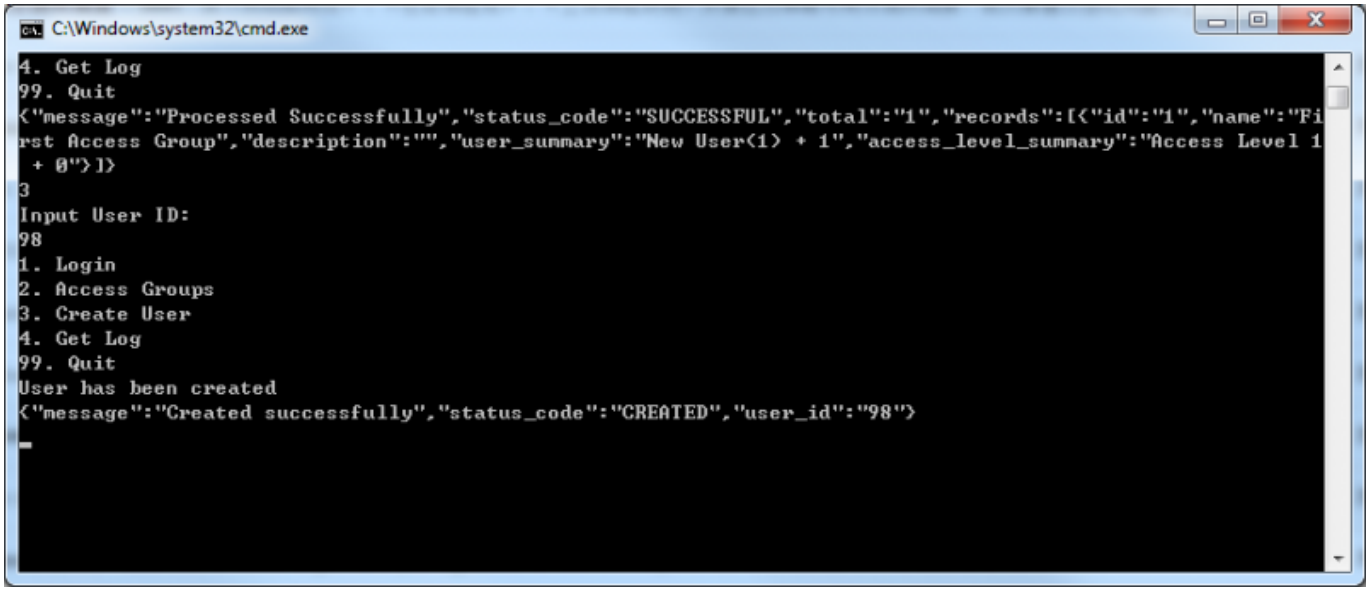
[Figure 2. After logging in]

When successfully logged in, you get the information of the user that you used for logging in. The user data is in the JSON format and includes very detailed data including fingerprint templates as you can see from Figure 2. Now that we've logged in, we can use other functions. In order to retrieve access groups, type "2" and press Enter.



[Figure 3. Retrieving access groups]

As you can see from the screenshot, there is only one access group in BioStar 2 Server and the name of the access group is “First Access Group” and if you look at the “user_summary” property, you can notice that it has two users. Now type “3” and press Enter to create a new user. You will be asked to enter the user ID. I entered “98” as user ID and got the success message from the server.



[Figure 4. Creating a new user]

Lastly, let’s retrieve log events from the server. Type “4” and press Enter and you will get log data from the server as shown in the screenshot below:

[Figure 5. Getting log list]

Analysis of the source code

1. Log in

The most important part in this sample application is log-in. Let’s take a look at the function below:

```
24      static async void LoginTask()
25      {
26          string resourceAddress = "http://127.0.0.1:8795/v2/login";
27
28          HttpClient httpClient = new HttpClient();
29
30          JavaScriptSerializer serializer = new JavaScriptSerializer();
31
32          Dictionary<string, string> dicLoginUser = new Dictionary<string, string>();
33          dicLoginUser.Add("name", "ts22");
34          dicLoginUser.Add("password", "rlaangkd!1");
35          dicLoginUser.Add("user_id", "niceggal1");
36
37          string jsonLoginUser = serializer.Serialize(dicLoginUser);
38
39          StringContent sc = new StringContent(jsonLoginUser, Encoding.UTF8, "application/json");
40          HttpResponseMessage httpResponse = await httpClient.PostAsync(resourceAddress, sc);
41
42
43          if(httpResponse.IsSuccessStatusCode == true)
44          {
45              Console.WriteLine(httpResponse.ToString());
46              string httpResponseBody = await httpResponse.Content.ReadAsStringAsync();
47              Console.WriteLine(httpResponseBody);
48
49
50              MemoryStream responseMemoryStream = new MemoryStream();
51              StreamWriter sw = new StreamWriter(responseMemoryStream);
52              sw.Write(httpResponse.ToString());
53              sw.Flush();
54
55              bool isSessionIDContained = httpResponse.Headers.Contains("Set-Cookie");
56              if (isSessionIDContained == true)
57              {
58                  IEnumerable<string> sessionEnum = httpResponse.Headers.GetValues("Set-Cookie");
59                  foreach(string element in sessionEnum)
60                  {
61                      Console.WriteLine("Set-Cookie: " + element);
62                      string[] strCookieArr = element.Split(new string[] { "bs-cloud-session-id=" }, StringSplitOptions.None);
63                      string[] strCookieArr2 = strCookieArr[1].Split(new string[] { ";" }, StringSplitOptions.None);
64                      sessionID = strCookieArr2[0];
65                  }
66              }
67              else
68              {
69                  Console.WriteLine("Session ID not found");
70              }
71          }
72          else
73          {
74              Console.WriteLine("Failed to log in");
75              Console.WriteLine(httpResponse.ToString());
76          }
77      }
```

- Line 26: This is the URL that we are using to log in to your local BioStar server. In case of Local API, “http://127.0.0.1:8795/v2/” is prefix. “login” after the prefix indicates a behavior or action we want to take.
- Line 27: In this sample code, we use class HttpClient to send a request and receive a response from the BioStar Cloud.
- Line 30: Class JavaScriptSerializer is needed to convert the data into JSON formatted data or parse the JSON formatted data into any format you want.
- Line 32-35: These lines creates a dictionary which consists of a string key and a string value. Three parameters are essential: your subdomain name, ID and password. The “name” field is for the subdomain name, so set this field to your subdomain name.
- Line 37: This line converts the dictionary to a JSON formatted string.
- Line 39: This line sets the JSON formatted string as the HTTP request content, UTF8 as encoding option, and JSON as media type.
- Line 40: We use HTTP POST method to make a HTTP request for login.
- Line 45-53: We output the content of the HTTP response for debugging purposes.
- Line 55-65: If the login information is valid, we receive the session information from the server. Every time we make an API call, we have to put that session information in the HTTP header. So, line

55 to 65 extracts the session information from the HTTP response header for later use.

2. Retrieving access groups

```
203         static async void AccessGroupsTask()
204         {
205             if (sessionID == null)
206             {
207                 Console.WriteLine("You must log in first!");
208                 return;
209             }
210 
211             CookieContainer cookieContainer = new CookieContainer();
212 
213             HttpClientHandler handler = new HttpClientHandler();
214             handler.CookieContainer = cookieContainer;
215 
216             HttpClient client = new HttpClient(handler);
217 
218 
219             cookieContainer.Add(new Uri(    "http://127.0.0.1:8795"      ), new Cookie("bs-cloud-session-id", sessionID));
220             HttpResponseMessage httpResponse = await client.GetAsync(     "http://127.0.0.1:8795/v2/access_groups"      );
221 
222             if (httpResponse.IsSuccessStatusCode == true)
223             {
224                 string httpResponseBody = await httpResponse.Content.ReadAsStringAsync();
225                 Console.WriteLine(httpResponseBody);
226             }
227             else
228             {
229                 Console.WriteLine("Retrieving Access Groups Failed");
230                 Console.WriteLine(httpResponse.ToString());
231             }
232         }
```

- Line 205-209: We first need to check if the login was successfully made and the session ID was stored.
- Line 211: We use class CookieContainer to send the session ID information to the BioStar server.
- Line 219: When putting the session ID in the cookie, we have to specify the URI.
- Line 220: Retrieving access groups should be done via HTTP GET method.

3. Retrieving events

```
125         static async void GetLogTask()
126         {
127             if(sessionID == null)
128             {
129                 Console.WriteLine("You must log in first!");
130                 return;
131             }
132 
133             CookieContainer cookieContainer = new CookieContainer();
134 
135             HttpClientHandler handler = new HttpClientHandler();
136             handler.CookieContainer = cookieContainer;
137 
138             HttpClient httpClient = new HttpClient(handler);
139 
140             HttpClient client = new HttpClient(handler);
141             cookieContainer.Add(new Uri( "http://127.0.0.1:8795"       ), new Cookie("bs-cloud-session-id", sessionID));
142 
143             string resourceAddress = "http://127.0.0.1/v2/monitoring/event_log/search";
144 
145             string startTime = "1970-01-01T00:00:00Z";
146             string endTime = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ");
147 
148             DateTime dtLatestLogTime = new DateTime(1970, 1, 1);
149 
150             JavaScriptSerializer serializer = new JavaScriptSerializer();
151 
152             for (int logCallIndex = 0; logCallIndex < 1000; logCallIndex++)
153             {
154                 endTime = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ");
155 
156                 string payload = "{ \"datetime\": [\"" + startTime + "\", \"" + endTime + "\"] }";
157 
158 
159                 StringContent sc = new StringContent(payload, Encoding.UTF8, "application/json");
160                 HttpResponseMessage httpResponse = await httpClient.PostAsync(resourceAddress, sc);
162              if (httpResponse.IsSuccessStatusCode == true)
163              {
164                  Console.WriteLine("Succeeded to retrieve log from " + startTime + " to " + endTime);
165                  string httpResponseBody = await httpResponse.Content.ReadAsStringAsync();
166                  Console.WriteLine(httpResponseBody);
167 
168                  endTime = startTime;
169 
170                  Dictionary<string, dynamic> logValues = serializer.Deserialize<Dictionary<string, dynamic>>(httpResponseBody);
171                  foreach(KeyValuePair<string, dynamic> logElement in logValues)
172                  {
173                      if (logElement.Key == "records")
174                      {
175                          foreach (Dictionary<string, dynamic> recordElement in logElement.Value)
176                          {
177                              if(recordElement.ContainsKey("datetime"))
178                              {
179                                  Console.WriteLine(recordElement["datetime"]);
180                                  DateTime dtLogTime = DateTime.Parse(recordElement["datetime"]);
181 
182                                  if(dtLogTime > dtLatestLogTime)
183                                  {
184                                      dtLatestLogTime = dtLogTime;
185                                      startTime = dtLatestLogTime.ToUniversalTime().AddSeconds(1).ToString("yyyy-MM-ddTHH:mm:ssZ");
186                                  }
187                              }
188                          }
189                      }
190                  }
191 
192                  System.Threading.Thread.Sleep(1000);
193              }
194              else
195              {
196                  Console.WriteLine("Log Retrieval Failed from " + startTime + " to " + endTime);
197                  Console.WriteLine(httpResponse.ToString());
198                  break;
199              }
```

- Line 152: We use the For loop to repeatedly retrieve the events from the server at an interval.
- Line 154-156: When retrieving the events from the server, we have to specify the start time and end time. This time, rather than using class Dictionary, we build a JSON formatted string manually for demonstration purposes.
- Line 170-188: We use class Dictionary with a string key and a dynamic value to parse the JSON

formatted data into a dictionary data structure. Since the events are in the form of an array, we have to use dynamic type of value in the dictionary.

4. Creating a user

```
79      static async void CreateUserTask()
80      {
81          if (sessionID == null)
82          {
83              Console.WriteLine("You must log in first!");
84              return;
85          }
86
87          CookieContainer cookieContainer = new CookieContainer();
88
89          HttpClientHandler handler = new HttpClientHandler();
90          handler.CookieContainer = cookieContainer;
91
92          HttpClient httpClient = new HttpClient(handler);
93
94          HttpClient client = new HttpClient(handler);
95          cookieContainer.Add(new Uri(    "http://127.0.0.1:8795"     ), new Cookie("bs-cloud-session-id", sessionID));
96
97          string resourceAddress = "http://127.0.0.1:8795/v2/users";
98
99          Console.WriteLine("Input User ID: ");
100         string userInputID = Console.ReadLine();
101
102         JavaScriptSerializer serializer = new JavaScriptSerializer();
103
104         Dictionary<string, string> dicNewUser = new Dictionary<string, string>();
105         dicNewUser.Add("user_id", userInputID);
106
107         string payload = serializer.Serialize(dicNewUser);
108
109         StringContent sc = new StringContent(payload, Encoding.UTF8, "application/json");
110         HttpResponseMessage httpResponse = await httpClient.PostAsync(resourceAddress, sc);
```

- Line 99-100: We receive a user input for the ID of a new user.
- Line 104-105: The only mandatory property that we have to provide when creating a new user is user ID.

Conclusion

So far, we have taken a brief look at how we can utilize BioStar API. Much of this article is not BioStar API specific. Rather, it's about basic usage of class HttpClient and what to do to use Web API. Therefore, even if you're not familiar with using Web API, I believe you can kick start on how to use BioStar API with this article. For more detailed information on BioStar API, visit our BioStar API website: https://api.biostar2.com

From:
http://kb.supremainc.com/knowledge/ -

Permanent link:
http://kb.supremainc.com/knowledge/doku.php?id=en:biostar_2_api_quickstart_guide&rev=1519689923

Last update: **2018/02/27 09:05**