



**Green University of Bangladesh**  
**Department of Computer Science and Engineering (CSE)**  
**Faculty of Sciences and Engineering Semester:**  
**(Summer, Year: 2025), B.Sc. in CSE (Day)**

**LAB REPORT NO: 02**  
**Course Title: Computer Networking Lab**  
**Course Code: CSE-318 Section: 231-D2**

**Lab Experiment Name: Developing and Executing an HTTP POST Request Using Java**

**Student Details**

Name		ID
1.	Promod Chandra Das	231002005

**Lab Date : 15.07.2025**

**Submission Date: 20.07.2025**

**Course Teacher's Name : Fatema Akter**

**[For Teachers use only: Don't Write Anything inside this box]**

<b><u>Lab Report Status</u></b>	
<b>Marks:</b> .....	<b>Signature:</b> .....
<b>Comments:</b> .....	<b>Date:</b> .....

## **1.TITLE OF THE LAB EXPERIMENT :**

### **Developing and Executing an HTTP POST Request Using Java**

## **2.OBJECTIVES :**

The primary objective of this experiment is to thoroughly understand the underlying mechanism of the HTTP POST method and how it functions within Java using the `HttpURLConnection` class. This lab focuses on building the foundational knowledge required to interact with web servers by sending structured data, particularly in JSON format. The experiment aims to illustrate the practical use of POST requests in a RESTful API context, where data is submitted to a web server for processing or storage. Through this implementation, students will learn how to configure an HTTP connection, set appropriate headers, and manage input/output streams efficiently.

Additionally, the experiment helps in developing the ability to analyze server responses, interpret HTTP status codes, and troubleshoot connection-related issues. It also emphasizes the importance of ensuring secure and well-structured communication between client and server. By the end of this lab, students are expected to have hands-on experience with sending POST requests programmatically, managing data exchange in client-server architecture, and applying the concept of network programming in real-world API-based applications.

## **3.PROCEDURE :**

- 1.Begin by initializing the main class and method.
- 2.Define a `URL` object with the target server endpoint.
- 3.Create a connection using `HttpURLConnection` and cast from the `URL` object.
- 4.Set the HTTP method type to "POST" explicitly.
- 5.Configure necessary request headers (e.g., Content-Type, Accept).
- 6.Enable output stream functionality for sending data.
- 7.Construct the JSON data string representing the payload.
- 8.Fetch the output stream from the connection object.
- 9.Write the JSON string to the output stream.
- 10.Flush and close the output stream to ensure data transmission.
- 11.Capture the HTTP response code returned by the server.
- 12.If the response indicates success (e.g., 200 OK), proceed to read the response input stream.

13. Use a buffer or scanner to accumulate the response message.
14. Close the input stream after reading is complete.
15. Display the response content on the console for verification.
16. Properly terminate the HTTP connection.
17. Conclude the program execution.

#### 4. IMPLEMENTATION :

```
package com.mycompany.httppost;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;

public class Httppost {

    public static void main(String[] args) throws IOException {

        URL url = new URL("https://jsonplaceholder.typicode.com/posts");

        HttpURLConnection connect = (HttpURLConnection) url.openConnection();

        connect.setRequestMethod("POST");

        connect.setRequestProperty("User-Agent", "Chrome");
        connect.setRequestProperty("Content-Type", "application/json");

        connect.setDoOutput(true);

        String jsonInput = "{\"title\":\"foo\",\"body\":\"bar\",\"userId\":1}";

        OutputStream os = connect.getOutputStream();
        os.write(jsonInput.getBytes());
        os.flush();
        os.close();

        int responseCode = connect.getResponseCode();
        System.out.println("Response Code: " + responseCode);
        System.out.println("Response Message: " + connect.getResponseMessage());
    }
}
```

```

        if (responseCode == HttpURLConnection.HTTP_CREATED) {
            BufferedReader reader = new BufferedReader(new InputStreamReader(connect.getInputStream()));
            StringBuilder response = new StringBuilder();
            String line;

            while ((line = reader.readLine()) != null) {
                response.append(line);
            }

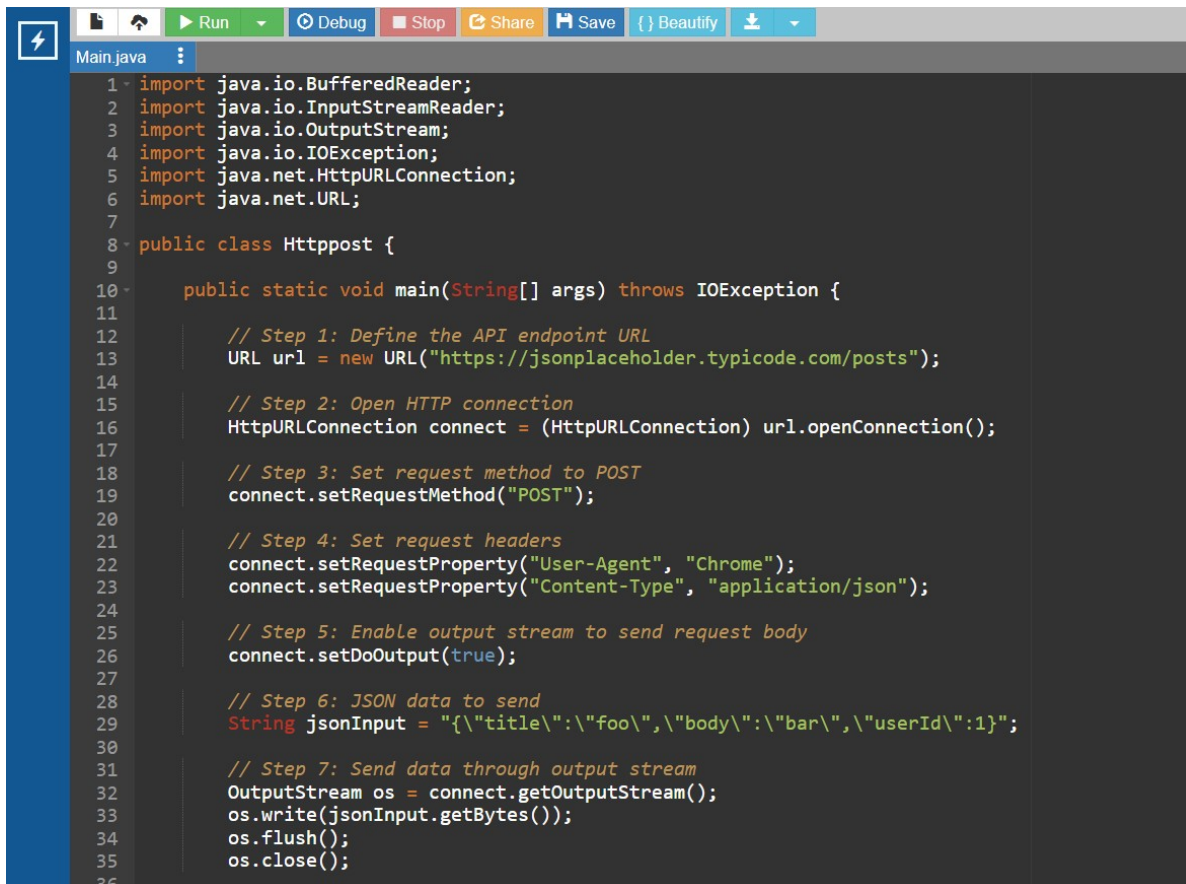
            reader.close();

            System.out.println("POST Response: " + response.toString());

        } else {
            System.out.println("POST Request Failed");
        }

        connect.disconnect();
    }
}

```



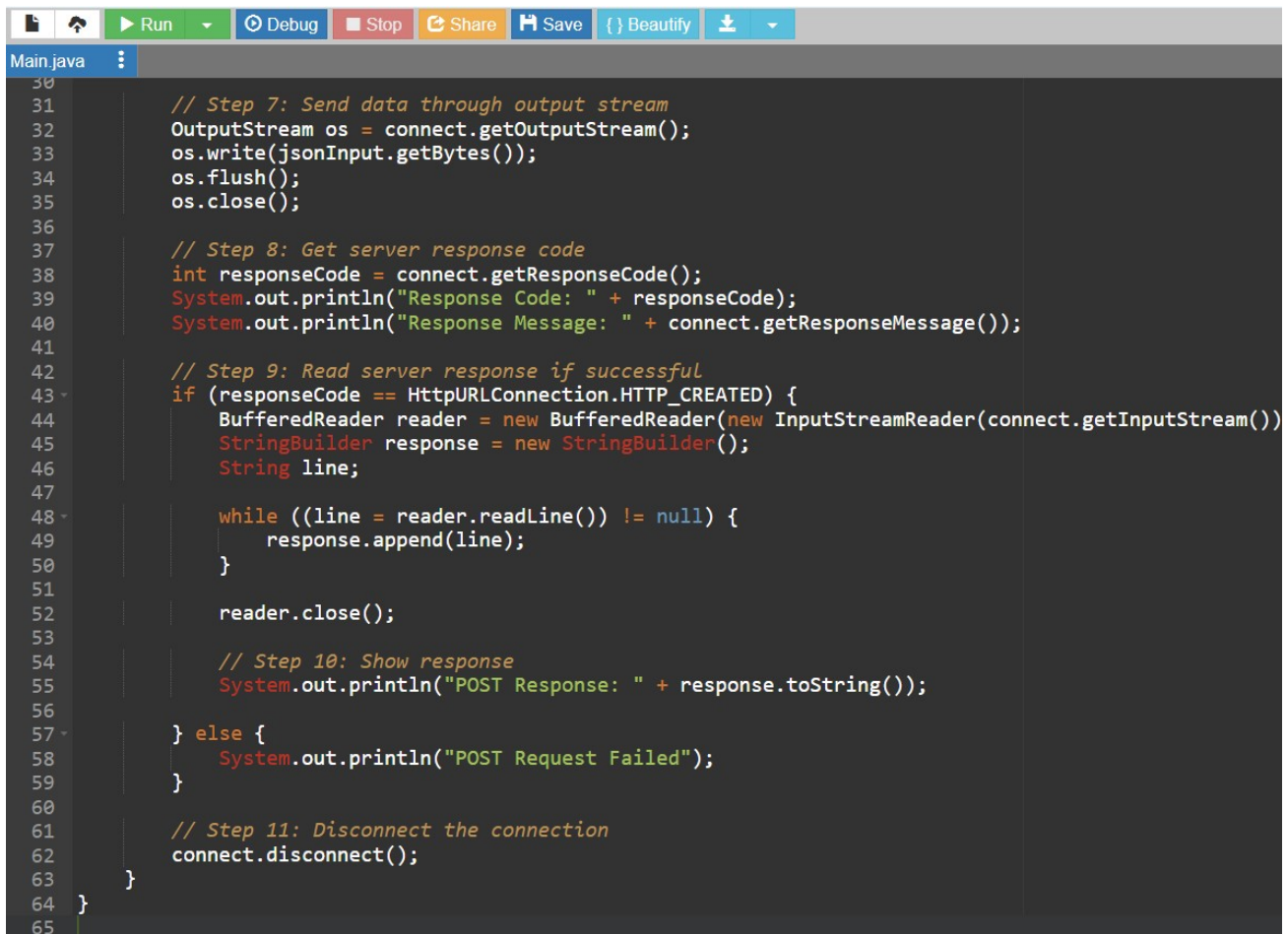
The screenshot shows an IDE window titled 'Main.java' with a toolbar at the top containing icons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The code is as follows:

```

1  import java.io.BufferedReader;
2  import java.io.InputStreamReader;
3  import java.io.OutputStream;
4  import java.io.IOException;
5  import java.net.HttpURLConnection;
6  import java.net.URL;
7
8  public class Httppost {
9
10     public static void main(String[] args) throws IOException {
11
12         // Step 1: Define the API endpoint URL
13         URL url = new URL("https://jsonplaceholder.typicode.com/posts");
14
15         // Step 2: Open HTTP connection
16         HttpURLConnection connect = (HttpURLConnection) url.openConnection();
17
18         // Step 3: Set request method to POST
19         connect.setRequestMethod("POST");
20
21         // Step 4: Set request headers
22         connect.setRequestProperty("User-Agent", "Chrome");
23         connect.setRequestProperty("Content-Type", "application/json");
24
25         // Step 5: Enable output stream to send request body
26         connect.setDoOutput(true);
27
28         // Step 6: JSON data to send
29         String jsonInput = "{\"title\": \"foo\", \"body\": \"bar\", \"userId\": 1}";
30
31         // Step 7: Send data through output stream
32         OutputStream os = connect.getOutputStream();
33         os.write(jsonInput.getBytes());
34         os.flush();
35         os.close();
36

```

**Figure : 01**



```

30
31 // Step 7: Send data through output stream
32 OutputStream os = connect.getOutputStream();
33 os.write(jsonInput.getBytes());
34 os.flush();
35 os.close();
36
37 // Step 8: Get server response code
38 int responseCode = connect.getResponseCode();
39 System.out.println("Response Code: " + responseCode);
40 System.out.println("Response Message: " + connect.getResponseMessage());
41
42 // Step 9: Read server response if successful
43 if (responseCode == HttpURLConnection.HTTP_CREATED) {
44     BufferedReader reader = new BufferedReader(new InputStreamReader(connect.getInputStream()))
45     StringBuilder response = new StringBuilder();
46     String line;
47
48     while ((line = reader.readLine()) != null) {
49         response.append(line);
50     }
51
52     reader.close();
53
54     // Step 10: Show response
55     System.out.println("POST Response: " + response.toString());
56
57 } else {
58     System.out.println("POST Request Failed");
59 }
60
61 // Step 11: Disconnect the connection
62 connect.disconnect();
63 }
64 }
65

```

**Figure : 02**

## 5. OUTPUT :

```
bash
```

```
Response Code: 201
```

```
Response Message: Created
```

```
POST Response: {"title":"foo","body":"bar","userId":1,"id":101}
```

## 6. DISCUSSION :

This lab provided valuable hands-on experience in working with the HTTP POST method using Java's `HttpURLConnection` class. Through this experiment, I explored the core process of establishing a connection with a server and transmitting JSON-formatted data in a structured and secure manner. I

learned how to configure HTTP request headers, enable data output through `setDoOutput(true)`, and correctly write data to the output stream.

One of the more challenging aspects was ensuring that the JSON payload was properly constructed and compatible with the server's expected format. Misplacing headers or forgetting to flush and close the output stream often resulted in failed responses, which taught me the importance of precision and error checking in network programming. Debugging issues using HTTP status codes gave me a clearer perspective on how to interpret server feedback effectively.

This lab also emphasized the fundamentals of RESTful communication, especially how clients and servers exchange data using HTTP protocol standards. By the end of the lab, I gained a much deeper understanding of how POST requests function internally, including both client-side setup and server response handling. Additionally, encountering and resolving subtle syntax errors boosted my confidence in dealing with real-world API scenarios. This foundation will be extremely useful as I move forward into more advanced web development and API integration tasks using Java.