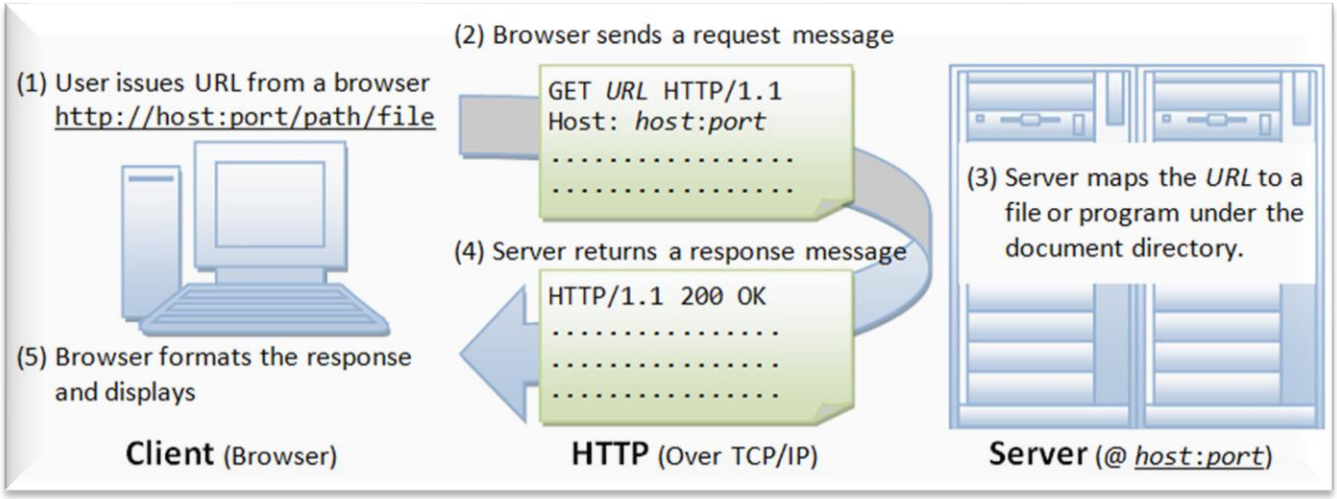
Network Programming
Section three

Basic web concept (HTTP: hypertext transfer protocol):

When you issue a URL from your browser to get a web resource using HTTP (e.g: https://www.google.com/index.html) The browser converts the URL into an HTTP request message and sends it to the HTTP server.

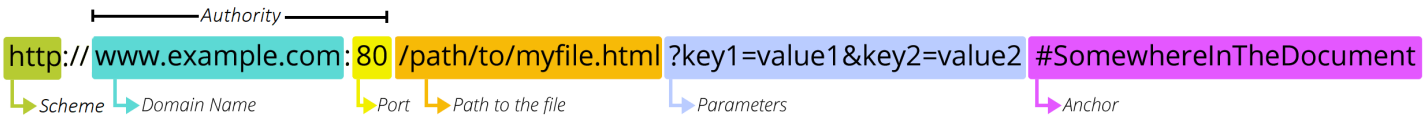
Http request and response messages for client and server

- HTTP messages are how data is exchanged between a server and a client.
- There are two types of messages:
  - *requests* sent by the client to trigger an action on the server.
  - *responses*, the answer from the server.



HTTP Request (Client Side):

- **Request URI (Uniform Resource Identifier):**
  - is a string of characters in a particular syntax that identifies a resource (A file on a server, An email address, A news message….)
  - There are two types of URIs:
    - Uniform resource locators (URLs):



    - Uniform resource names (URNs):
      namespace:resource_name

- **HTTP Method:**

  - defines the action the client wants to perform on the resource identified by the Request URI.
  - Common HTTP methods include:
    - **GET**: for reading data.
    - **POST**: for sending data to create something new.
    - **PUT**: for updating data.
    - **DELETE**: for removing data.

- **Each request whatever its method consists of two parts:**

1. **Headers:**

  - HTTP headers are key-value pairs included in the request to provide additional information to the server.
  - Common headers include:
    - **Host**: Specifies the domain name or IP address of the server.
    - **User-Agent**: Identifies the client software (e.g., browser) making the request.
    - **Accept**: Informs the server about the types of media that the client can process (e.g., JSON, XML, HTML).
    - **Content-Type**: Specifies the media type of the request body (e.g., application/json).
    - **Cookie**: Contains session-related data sent to the server.
    - **Connection**: Specifies whether the client wants to keep the connection alive for multiple requests (e.g., "Connection: keep-alive") or close it after a single request (e.g., "Connection: close").
    - **Content-Length**: Specifies the size of the request body in bytes. It helps the server know how much data to expect in the request.
    - **Proxy-Authorization**: used when the client is communicating with a proxy server that requires authentication.
    - Other headers you can search on (If-Modified-Since, If-None-Match, Range, Origin, Proxy-Authorization, TE (Transfer-Encoding), X-Requested-With, DNT (Do Not Track), Forwarded)

2. **Request Body (for methods like POST):**

  - The request body is used to send data to the server, typically with methods like POST, PUT, or PATCH.
  - The format of the request body depends on the **Content-Type** header.
  - Examples of request bodies:
    - JSON: **{ "name": "John", "age": 30 }**
    - Form Data: **name=John&age=30**
    - XML: **<user><name>John</name><age>30</age></user>**

HTTP Response (server Side):

1. Status Line:

- The status line is the first line of an HTTP response. It consists of two parts:
  - HTTP Status Code: A three-digit numeric code that indicates the outcome of the request. It tells the client whether the request was successful, encountered an error, or needs further action.

    - Reason Phrase: A short textual description associated with the status code, providing additional context. While it's helpful for humans, it's not used by machines to determine the response meaning.
- Here are some commonly used HTTP status codes:
    - 1xx (Informational): These status codes indicate that the request was received, and the server is continuing to process it.
    - 2xx (Successful): These status codes indicate that the request was successfully received, understood, and accepted by the server.
        - 200 OK: The request was successful.
        - 201 Created: The request resulted in the creation of a new resource.
        - 204 No Content: The request was successful, but there is no response body.
    - 3xx (Redirection): These status codes indicate that further action is required to complete the request, such as redirecting to a different URL.
        - 301 Moved Permanently: The requested resource has permanently moved to a different URL.
        - 302 Found: The requested resource has temporarily moved to a different URL.
    - 4xx (Client Error): These status codes indicate that the client has made an error or the request cannot be fulfilled due to client-side issues.
        - 400 Bad Request: The request is malformed or invalid.
        - 401 Unauthorized: Authentication is required for access.
        - 403 Forbidden: The client does not have permission to access the resource.
        - 404 Not Found: The requested resource could not be found.
    - 5xx (Server Error): These status codes indicate that the server encountered an error or is otherwise incapable of performing the request.
        - 500 Internal Server Error: A generic server error occurred.
        - 502 Bad Gateway: The server, while acting as a gateway or proxy, received an invalid response from an upstream server.
        - 503 Service Unavailable: The server is temporarily unavailable due to maintenance or overload.

2. Headers:

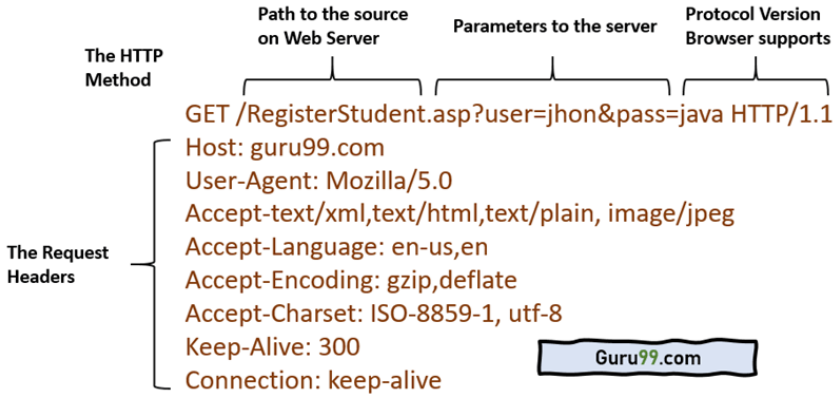
- HTTP response headers are key-value pairs that provide additional information about the response and how it should be handled. Some common response headers include:
    - **Date**: The timestamp when the response was generated.
    - **Server**: The name and version of the server software.
    - **Content-Type**: Describes the media type (e.g., HTML, JSON, XML) of the response body.
    - **Content-Length**: Indicates the size of the response body in bytes.
    - **X-Frame-Options**: used to protect a web page from being shown inside another web page (in an iframe). It helps prevent certain types of cyberattacks.
    - **Cache-Control**: Provides directives for caching the response at the client or intermediary servers.
    - **Location**: Used for redirection, indicating a new URL to follow.

3. Response Body:

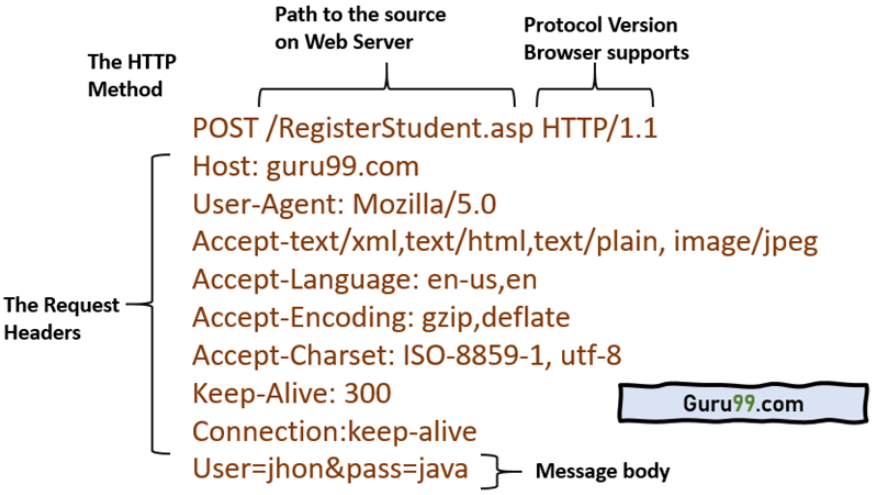
- The response body contains the actual data being sent from the server to the client. Its content type is specified in the **Content-Type** header.
- The response body can be HTML, JSON, XML, plain text, or any other format depending on the server's response.

Examples of HTTP requests using Different Methods:

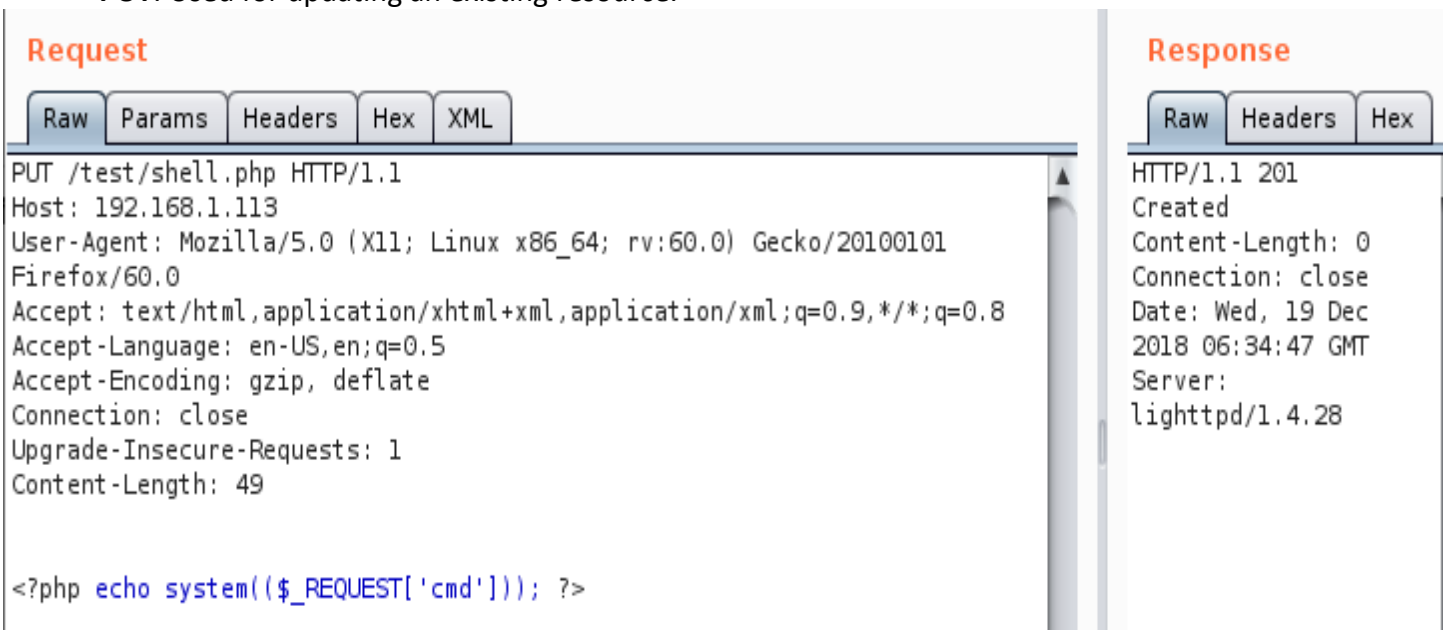
- **GET**: Used for retrieving data.



- **POST**: Used for submitting data, like creating a new resource.



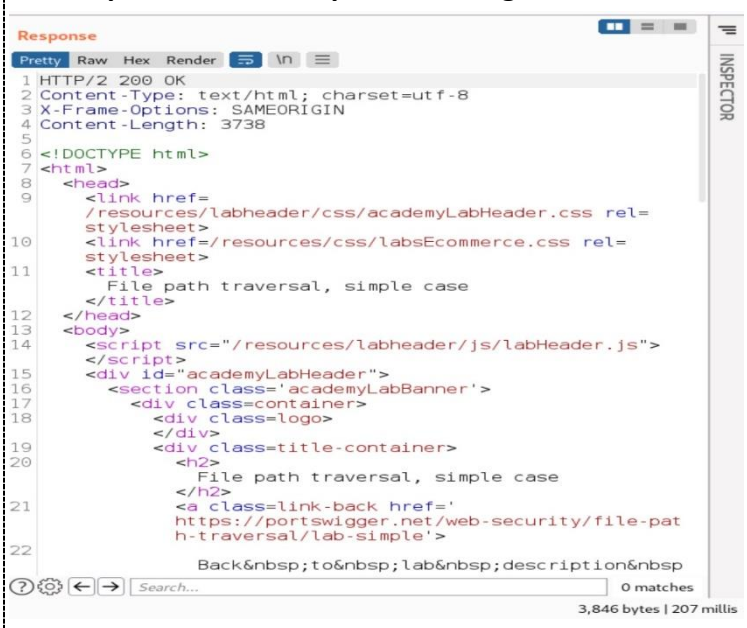
- **PUT**: Used for updating an existing resource.



- **DELETE**: Used for removing a resource.

  DELETE /api/products/456 HTTP/1.1
  Host: example.com

Examples for HTTP response message:



How to represent http protocol in java program

In Java, you can use the **java.net** package to work with HTTP and represent HTTP protocol in your program. The **HttpURLConnection** class is commonly used to send HTTP requests and receive HTTP responses. Here's a basic example of how to perform an HTTP GET request in Java:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
public class HttpExample {
   public static void main(String[] args) {
      try {
         // Create a URL object for the target resource
         URL url = new URL("https://en.wikipedia.org/wiki/Main_Page");
         // Open a connection to the URL
         HttpURLConnection connection = (HttpURLConnection) url.openConnection();
         // Set the HTTP request method (GET in this case)
         connection.setRequestMethod("GET");
         // Get the response code
         int responseCode = connection.getResponseCode();
         System.out.println("Response Code: " + responseCode);
         // Read the response data
         BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
         StringBuilder response = new StringBuilder();
         String line;
         while ((line = reader.readLine()) != null) {
            response.append(line);
         }
         reader.close();
         // Print the response data
         System.out.println("Response Data:");
         System.out.println(response.toString());
         // Close the connection
         connection.disconnect();
      } catch (IOException e) {
```

```
        e.printStackTrace();
    }}}
```

HTTP POST request in Java using the **HttpURLConnection** class. This example sends a POST request to a JSONPlaceholder API, which is a fake online REST API for testing and prototyping.

```
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
public class HttpPostExample {
    public static void main(String[] args) {
        try {
            // Create a URL object for the JSONPlaceholder API endpoint
            URL url = new URL("https://jsonplaceholder.typicode.com/posts");
            // Open a connection to the URL
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            // Set the HTTP request method to POST
            connection.setRequestMethod("POST");
            // Enable input and output streams for the connection
            connection.setDoInput(true);
            connection.setDoOutput(true);
            // Set the content type to JSON
            connection.setRequestProperty("Content-Type", "application/json");
            // Create the JSON data to send in the request body
            String jsonData = "{\n" +
                    "    \"title\": \"Sample Post\",\n" +
                    "    \"body\": \"This is a sample post request.\",\n" +
                    "    \"userId\": 1\n" +
                    "}";
            // Write the JSON data to the output stream
            try (OutputStream outputStream = connection.getOutputStream()) {
                byte[] input = jsonData.getBytes("utf-8");
                outputStream.write(input, 0, input.length);
            }
            // Get the response code
            int responseCode = connection.getResponseCode();
            System.out.println("Response Code: " + responseCode);
            // Read the response data
            try (BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()))) {
                StringBuilder response = new StringBuilder();
                String line;
                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }
                System.out.println("Response Data:");
                System.out.println(response.toString());
            }
            // Close the connection
            connection.disconnect();
        } catch (IOException e) {
            e.printStackTrace();
    }}}
```

HTTP PUT request in Java using the **HttpURLConnection** class. In this example, we'll update an existing resource on the JSONPlaceholder API:

```
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
```

```java
public class HttpPutExample {
    public static void main(String[] args) {
        try {
            // Create a URL object for the JSONPlaceholder API endpoint
            URL url = new URL("https://jsonplaceholder.typicode.com/posts/1");
            // Open a connection to the URL
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            // Set the HTTP request method to PUT
            connection.setRequestMethod("PUT");
            // Enable input and output streams for the connection
            connection.setDoInput(true);
            connection.setDoOutput(true);
            // Set the content type to JSON
            connection.setRequestProperty("Content-Type", "application/json");
            // Create the JSON data to send in the request body for the update
            String jsonData = "{\n" +
                    "    \"title\": \"Updated Post Title\",\n" +
                    "    \"body\": \"This is an updated post body.\",\n" +
                    "    \"userId\": 1,\n" +
                    "    \"id\": 1\n" +
                    "}";
            // Write the JSON data to the output stream
            try (OutputStream outputStream = connection.getOutputStream()) {
                byte[] input = jsonData.getBytes("utf-8");
                outputStream.write(input, 0, input.length);
            }
            // Get the response code
            int responseCode = connection.getResponseCode();
            System.out.println("Response Code: " + responseCode);
            // Read the response data
            try (BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()))) {
                StringBuilder response = new StringBuilder();
                String line;
                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }
                System.out.println("Response Data:");
                System.out.println(response.toString());
            }
            // Close the connection
            connection.disconnect();
        } catch (IOException e) {
            e.printStackTrace();
        }}}
```