**TASK 1**

HTTP REQUEST

using System;

using System.Net;

using System.Net.Sockets;

using System.Threading.Tasks;

namespace SSE

{

public class **HttpRequest**

{

private HttpRequest() {}

public static async Task<string> Get(Url url)

{

// resolve DNS

var addrs = await Dns.GetHostAddressesAsync(url.Host);

// extract a IPv4 from the list of available addresses (since tu-chemnitz.de

// is not available through IPv6 yet)

// a more general approach would be to try a request to each address

// in the list in a return the result from the first successfull request

IPAddress ipv4 = null;

foreach (var addr in addrs)

{

if (addr.AddressFamily == AddressFamily.InterNetwork)

{

ipv4 = addr;

break;

}

}

// if there is no IPv4 in the list of addresses for the given host

if (ipv4 == null)

{

throw new ArgumentException("Cannot resolve IPv4 for host: " + url.Host);

}

// construct HTTP request

var request = "GET " + url.Path + "?" + url.Query + " HTTP/1.1\n" +

"Host: " + url.Host + "\n" +

"\n";

// send TCP request

// Note: updated TcpRequest implementation to also support IPv6 requests

return await TcpRequest.Do(ipv4, url.Port, request);

}

}

}

using System;

using System.Threading.Tasks;

namespace SSE

{

public class Program

{

public static void **Main**(string[] args)

{

Task.Run(async () =>

{

var url = new Url("http://www.tu-chemnitz.de");

var resp = await HttpRequest.Get(url);

Console.WriteLine(resp);

}).Wait();

}

}

}

using System.Net.Sockets;

using System.Text;

using System.Threading.Tasks;

namespace SSE

{

public class **TcpRequest**

{

private TcpRequest() {}

public static async Task<string> Do(IPAddress ip, int port, string message)

{

var client = new TcpClient(ip.AddressFamily);

await client.ConnectAsync(ip, port);

using (var r = new StreamReader(client.GetStream(), Encoding.ASCII))

using (var w = new StreamWriter(client.GetStream(), Encoding.ASCII))

{

await w.WriteAsync(message);

w.Flush();

var buffer = new char[4096];

var byteCount = await r.ReadAsync(buffer, 0, buffer.Length);

return new string(buffer);

}

}

}

}

using System;

using System.Text.RegularExpressions;

namespace SSE

{

/// <summary>

/// A class for generating and parsing HTTP-URIs.

/// </summary>

public class **Url**

{

const string VALID\_CHARACTERS = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789$-\_.~";

public string Scheme = "";

public string Host = "";

public int Port = 80;

public string Path = "";

public string Query = "";

public string FragmentId = "";

/// <summary>

/// Constructor for parsing URLs.

/// </summary>

public Url(string urlStr)

{

Match match = Regex.Match(urlStr, @"

^(?:

(?<scheme>[^:]\*)

\:\/\/(?<host>[^:^/^?^#]\*)

(?:\:(?<port>\d\*))?

)?

(?<path>\/[^?^#]\*)?

(?:\?(?<query>[^#]\*))?

(?:\#(?<fragmentid>.\*))?$

", RegexOptions.IgnorePatternWhitespace);

if (match.Success)

{

this.Scheme = match.Groups["scheme"].Value.ToLower();

this.Host = Decode(match.Groups["host"].Value);

if (match.Groups["port"].Value != "")

this.Port = Convert.ToInt32(match.Groups["port"].Value);

this.Path = Decode(match.Groups["path"].Value);

this.Query = Decode(match.Groups["query"].Value);

this.FragmentId = Decode(match.Groups["fragmentid"].Value);

}

else

{

throw new FormatException("Could not parse URL: " + urlStr);

}

}

/// <summary>

/// Constructor for building URLs from their components.

/// </summary>

public Url(string scheme, string host, int port, string path, string query, string fragmentId)

{

this.Scheme = scheme;

this.Host = host;

this.Port = port;

this.Path = path;

this.Query = query;

this.FragmentId = fragmentId;

}

/// <summary>

/// Returns the string representation of the URL.

/// </summary>

public override string ToString()

{

string url = Scheme + "://" + Host;

if (Port != 80) url += ":" + Port;

if (Path != "") url += "/" + Encode(Path.Substring(1));

if (Query != "")

{

var queryParts = Query.Split('&');

var qEncoded = "";

foreach ( var queryPart in queryParts )

{

var nameValue = queryPart.Split( '=' );

qEncoded += "&" + Encode( nameValue[0] ) + "=" + Encode( nameValue[1] );

}

url += "?" + qEncoded.Substring( 1 );

}

if (FragmentId != "") url += "#" + Encode(FragmentId);

return url;

}

/// <summary>

/// Encodes any special characters in the URL with an escaping sequence.

/// </summary>

public static string Encode(string s)

{

string result = "";

for (int i = 0; i < s.Length; i++)

{

if (VALID\_CHARACTERS.Contains(s[i].ToString()))

{

// allowed character

result += s[i];

}

else

{

// character has to be encoded as "%" + HexDigit + HexDigit

result += "%" + Convert.ToByte(s[i]).ToString("X");

}

}

return result;

}

/// <summary>

/// Decodes any escaping sequence in the URL with the corresponding characters.

/// </summary>

public static string Decode(string s)

{

while (s.Contains("%"))

{

int pos = s.IndexOf("%");

byte b = byte.Parse(s.Substring(pos + 1, 2), System.Globalization.NumberStyles.HexNumber);

s = s.Substring(0, pos) + Convert.ToChar(b) + s.Substring(pos + 3);

}

return s;

}

}

}

**TASK 2**

using System;

using System.Collections.Generic;

using System.Linq;

namespace SSE

{

public class **HttpMessage**

{

public const string GET = "GET";

public const string POST = "POST";

public const string PUT = "PUT";

public const string DELETE = "DELETE";

public const string HEAD = "HEAD";

public const string OPTIONS = "OPTIONS";

public const string TRACE = "TRACE";

public string Method = "";

public string Host = "";

public string Resource = "";

public Dictionary<string, string> Headers = new Dictionary<string, string>();

public string Content = "";

public string StatusCode = "";

public string StatusMessage = "";

/// <summary>

/// Construct an HTTP request.

/// </summary>

public HttpMessage(string method, string host, string resource, Dictionary<string, string> headers, string content)

{

this.Method = method;

this.Host = host;

this.Resource = resource;

this.Headers = headers;

if (Headers == null)

Headers = new Dictionary<string, string>();

this.Content = content;

StatusCode = null;

StatusMessage = null;

}

/// <summary>

/// Construct an HTTP response.

/// </summary>

public HttpMessage(string statusCode, string statusMessage, Dictionary<string, string> headers, string content)

{

this.Method = null;

this.Host = null;

this.Resource = null;

this.Headers = headers;

if (Headers == null)

Headers = new Dictionary<string, string>();

this.Content = content;

StatusCode = statusCode;

StatusMessage = statusMessage;

}

/// <summary>

/// Constructs an HTTP message by parsing a (received) string.

/// </summary>

/// <param name="message"></param>

public HttpMessage(string message)

{

// loop through lines in message

var lines = message.Split('\n');

var firstLine = lines.First();

// parse first line

var parts = firstLine.Split(' ');

if (parts.Count() != 3)

{

throw new FormatException("Malformed HTTP message: " + firstLine);

}

if (parts[0] == "HTTP/1.0" || parts[0] == "HTTP/1.1")

{ // is response

StatusCode = parts[1];

StatusMessage = parts[2];

}

else

{ // is request

Method = parts[0];

Resource = parts[1];

}

// parse other lines

for (int i = 1; i < lines.Count(); i++)

{

var line = lines[i];

if (line == "")

{ // empty line -> end of header; the rest is content

Content = String.Join("\n", lines.Skip(i + 1));

break;

}

else

{ // parse header (name-value-pair)

var colonAt = line.IndexOf(':');

if (colonAt == -1)

{

throw new FormatException("Malformed header: " + line);

}

var name = line.Substring(0, colonAt).ToLower().Trim();

var value = line.Substring(colonAt + 1).Trim();

if (name == "host")

{

Host = value;

}

else

{

Headers[name] = value;

}

}

}

}

/// <summary>

/// Returns the string representation of the message.

/// </summary>

public override string ToString()

{

// set content length

Headers["content-length"] = Content.Length.ToString();

// build first line

string message;

if (Method != null)

{

message = Method + " " + Resource + " HTTP/1.1\nhost: " + Host + "\n";

}

else

{

message = "HTTP/1.1 " + StatusCode + " " + StatusMessage + "\n";

}

// add Headers

foreach (string name in Headers.Keys)

{

message += name + ": " + Headers[name] + "\n";

}

// add content

message += "\n" + Content;

return message;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Xunit;

namespace SSE

{

public class **HttpMessageTests**

{

[Fact]

public void TestResponseMessage()

{

var msg = new HttpMessage("HTTP/1.1 200 OK\nContent-Type: text/html\nContent-Length: 12\n\nhello world\n");

Assert.Equal("200", msg.StatusCode);

Assert.Equal("OK", msg.StatusMessage);

Assert.Equal("text/html", msg.Headers["content-type"]);

Assert.Equal("12", msg.Headers["content-length"]);

Assert.Equal("hello world\n", msg.Content);

}

[Fact]

public void TestRequestMessage()

{

var msg = new HttpMessage("POST /test HTTP/1.1\nHost: example.org\nContent-Length: 5\n\nhallo");

Assert.Equal("POST", msg.Method);

Assert.Equal("/test", msg.Resource);

Assert.Equal("example.org", msg.Host);

Assert.Equal("5", msg.Headers["content-length"]);

Assert.Equal("hallo", msg.Content);

}

}

}