МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №7

по дисциплине: **«**Программирование сетевых приложений**»**

на тему: «Программирование приложений с использованием

*WebSocket*»

Выполнил: студент гр. ИТП-41

Дорошко Д. А.

Принял: преподаватель

Гуменников Е. Д.

Гомель 2022

**Цель:** изучить работу с *web*-сокетами.

**Задание:**

Курсы валют, сервер генерирует изменения курсов валют ­ клиенты входят на сайт и получают обновленные курсы валют в реальном времени.

**Ход работы:**

Работа клиента представлена на рисунке 1.

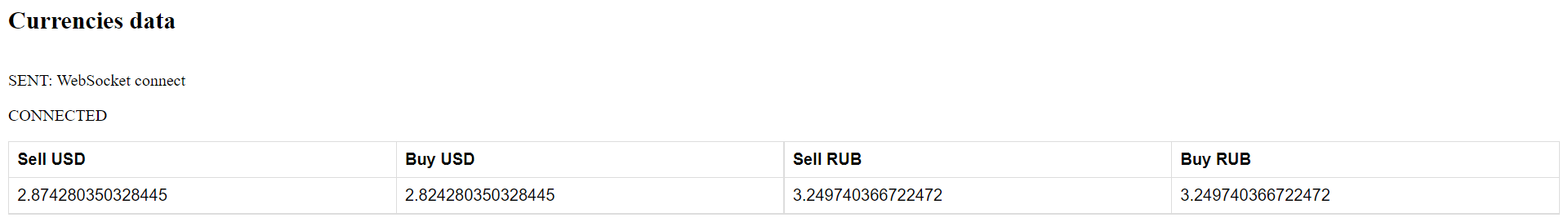


Рисунок 1 – Работа клиента

Работа сервера представлена на рисунке 2.

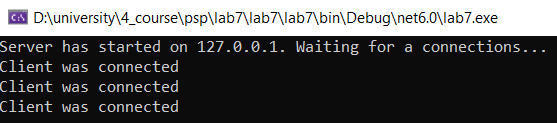


Рисунок 2 – Работа сервера

Листинг программы представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы получены навыки работы с *web*-сокетами с использованием .*NET* и *JavaScript*.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

<!DOCTYPE html>

<html lang="en">

<style>

textarea {

vertical-align: bottom;

}

#output {

overflow: auto;

}

#output > p {

overflow-wrap: break-word;

}

#output span {

color: blue;

}

#output span.error {

color: red;

}

table {

font-family: arial, sans-serif;

border-collapse: collapse;

width: 100%;

}

td, th {

border: 1px solid #dddddd;

text-align: left;

padding: 8px;

}

tr:nth-child(even) {

background-color: #dddddd;

}

</style>

<body>

<h2>Currencies data</h2>

<div id="output"></div>

<table>

<thead>

<tr>

<th>Sell USD</th>

<th>Buy USD</th>

<th>Sell RUB</th>

<th>Buy RUB</th>

</tr>

</thead>

<tbody>

</tbody>

</table>

</body>

<script>

const output = document.querySelector("#output");

const wsUri = "ws://127.0.0.1:5015/";

const websocket = new WebSocket(wsUri);

websocket.onopen = (e) => {

writeToScreen("CONNECTED");

doSend("WebSocket connect");

};

websocket.onclose = (e) => {

writeToScreen("DISCONNECTED");

};

websocket.onmessage = (e) => {

let tBody = document.querySelector('tbody');

let currency = JSON.parse(e.data);

let newRow = `

<tr>

<td>${currency.SellUSD}</td>

<td>${currency.BuyUSD}</td>

<td>${currency.SellRUB}</td>

<td>${currency.BuyRUB}</td>

</tr>

`;

tBody.innerHTML = newRow;

};

websocket.onerror = (e) => {

writeToScreen(`<span class="error">ERROR:</span> ${e.data}`);

};

function doSend(message) {

writeToScreen(`SENT: ${message}`);

websocket.send(message);

}

function writeToScreen(message) {

output.insertAdjacentHTML("afterbegin", `<p>${message}</p>`);

}

</script>

</html>

using System.Net.Sockets;

using System.Net;

using System.Text;

using System.Text.RegularExpressions;

using Newtonsoft.Json;

using System.Collections.Concurrent;

namespace lab7

{

public class CurrencyServer

{

ConcurrentBag<Currency> \_currencies = new ConcurrentBag<Currency>();

private IList<TcpClient> \_clients = new List<TcpClient>();

private TcpListener \_listener;

private bool \_isStarted;

private readonly IPAddress \_address;

private readonly int \_port;

public CurrencyServer(IPAddress address, int port)

{

\_address = address;

\_port = port;

\_listener = new TcpListener(\_address, \_port);

}

public void Start()

{

\_isStarted = true;

\_clients = new List<TcpClient>();

\_listener.Start();

Console.WriteLine($"Server has started on {\_address}. Waiting for a connections...");

ThreadPool.QueueUserWorkItem(

state =>

{

UpdateCurrencies();

});

while (\_isStarted)

{

var client = \_listener.AcceptTcpClient();

\_clients.Add(client);

ThreadPool.QueueUserWorkItem(

state =>

{

ProccessClient(state as TcpClient);

}, client);

}

}

private void UpdateCurrencies()

{

while (true)

{

var currentUSD = (decimal)new Random().NextDouble();

var currentRUB = (decimal)new Random().NextDouble();

var currentCurrency = new Currency

{

BuyUSD = currentUSD + 2,

SellUSD = currentUSD + 2.05M,

BuyRUB = currentRUB + 3,

SellRUB = currentRUB + 3,

};

\_currencies.Add(currentCurrency);

Thread.Sleep(1000);

}

}

private void ProccessClient(TcpClient client)

{

Console.WriteLine("Client was connected");

var stream = client.GetStream();

while (true)

{

while (!stream.DataAvailable) ;

while (client.Available < 3) ;

var bytes = new byte[client.Available];

stream.Read(bytes, 0, client.Available);

var clientMessage = Encoding.UTF8.GetString(bytes);

if (Regex.IsMatch(clientMessage, "^GET", RegexOptions.IgnoreCase))

{

HandShake(stream, clientMessage);

continue;

}

while (true)

{

var currency = \_currencies.FirstOrDefault();

if (currency != null)

{

var clientStream = client.GetStream();

var serverAnswer = EncodeMessageToSend(JsonConvert.SerializeObject(currency));

clientStream.Write(serverAnswer, 0, serverAnswer.Length);

}

Thread.Sleep(1100);

}

}

}

private void HandShake(NetworkStream stream, string clientMessage)

{

var swk = Regex.Match(clientMessage, "Sec-WebSocket-Key: (.\*)").Groups[1].Value.Trim();

var swka = swk + "258EAFA5-E914-47DA-95CA-C5AB0DC85B11";

var swkaSha1 = System.Security.Cryptography.SHA1.Create().ComputeHash(Encoding.UTF8.GetBytes(swka));

var swkaSha1Base64 = Convert.ToBase64String(swkaSha1);

var response = Encoding.UTF8.GetBytes(

"HTTP/1.1 101 Switching Protocols\r\n" +

"Connection: Upgrade\r\n" +

"Upgrade: websocket\r\n" +

"Sec-WebSocket-Accept: " + swkaSha1Base64 + "\r\n\r\n");

stream.Write(response, 0, response.Length);

}

private string DecodeMessage(byte[] bytes)

{

var secondByte = bytes[1];

var dataLength = secondByte & 127;

var indexFirstMask = 2;

if (dataLength == 126)

indexFirstMask = 4;

else if (dataLength == 127)

indexFirstMask = 10;

var keys = bytes.Skip(indexFirstMask).Take(4);

var indexFirstDataByte = indexFirstMask + 4;

var decoded = new byte[bytes.Length - indexFirstDataByte];

for (int i = indexFirstDataByte, j = 0; i < bytes.Length; i++, j++)

{

decoded[j] = (byte)(bytes[i] ^ keys.ElementAt(j % 4));

}

return Encoding.UTF8.GetString(decoded, 0, decoded.Length);

}

private static byte[] EncodeMessageToSend(string message)

{

byte[] response;

byte[] bytesRaw = Encoding.UTF8.GetBytes(message);

byte[] frame = new byte[10];

var length = (long)bytesRaw.Length;

frame[0] = 129;

int indexStartRawData;

if (length <= 125)

{

frame[1] = (byte)length;

indexStartRawData = 2;

}

else if (length >= 126 && length <= 65535)

{

frame[1] = 126;

frame[2] = (byte)((length >> 8) & 255);

frame[3] = (byte)(length & 255);

indexStartRawData = 4;

}

else

{

frame[1] = 127;

frame[2] = (byte)((length >> 56) & 255);

frame[3] = (byte)((length >> 48) & 255);

frame[4] = (byte)((length >> 40) & 255);

frame[5] = (byte)((length >> 32) & 255);

frame[6] = (byte)((length >> 24) & 255);

frame[7] = (byte)((length >> 16) & 255);

frame[8] = (byte)((length >> 8) & 255);

frame[9] = (byte)(length & 255);

indexStartRawData = 10;

}

response = new byte[indexStartRawData + length];

int i, reponseIdx = 0;

for (i = 0; i < indexStartRawData; i++)

{

response[reponseIdx] = frame[i];

reponseIdx++;

}

for (i = 0; i < length; i++)

{

response[reponseIdx] = bytesRaw[i];

reponseIdx++;

}

return response;

}

}

}