МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

дисциплина «Программирование сетевых приложений»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

ПРОГРАММИРОВАНИЕ СЕТЕВОГО КЛИЕНТ-СЕРВЕРНОГО ВЗАИМОДЕЙСТВИЯ С ИСПОЛЬЗОВАНИЕМ HTTPS

Выполнил: студент гр. ИТП-41

Болотникова Е.В.

Принял: преподаватель

Гуменников Е.Д.

Гомель 2022

**Цель работы:** изучить основные принципы разработки распределенных приложений при помощи протокола HTTPS.

**Задание:**

Для задания представленном ниже для коммуникации использовать HTTPS протокол (генерировать тестовый сертификат с использованием OpenSSL, можно использовать сертификаты генерируемые инструментами Windows). (Вся работа должна быть выполнена с использованием стандартных сокетов, все HTTPS заголовки формируются вручную)

Разработать http­сервер (использовать пул потоков для параллельной обработки запросов клиентов). В качестве клиентского приложения используется веб­браузер. Все заголовки формируются вручную. Браузер должен правильно отображать получаемую информацию. Рекомендуется на GET­запрос возвращать html страницу с формой ввода, и выполнять POST­запрос, на который сервер присылает решение.

Индивидуальное задание:

Разработать конечный автомат для анализа корректности email. Найти все email в тексте, отправленном пользователем. Клиенту возвращается список всех найденных email в тексте.

**Ход выполнения**

В процессе выполнения данной лабораторной работы было разработано серверное приложение, работающее при помощи протокола https. К данному приложению могут подключаться пользователи через браузер. В ответ пользователю отправляется страничка с базовой функциональностью, при помощи которой пользователь может делать запросы на сервер и получать ответы. На рисунке 1 представлена разработанная страница сайта.

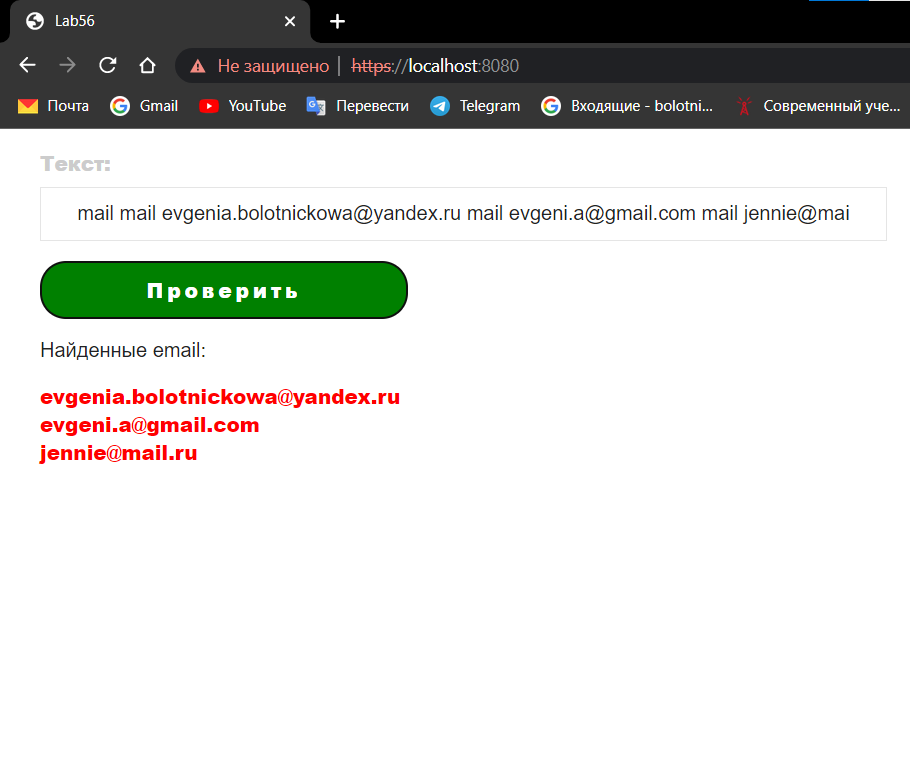


Рисунок 1 – Страница приложения

На рисунке 2 представлено создание сертификата:

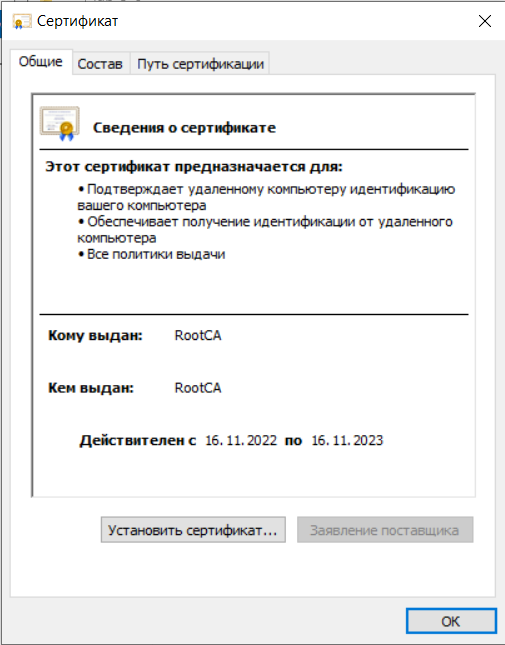


Рисунок 2 – Создание сертификата

**Вывод**: в процессе выполнения данной лабораторной работы были изучены основные принципы разработки клиент серверных приложений с использованием протокола https. А также полученные знания были применены на практике для разработки серверного приложения для обработки пользовательских запросов.

**Листинг Program.cs:**

using System;

using System.Threading;

namespace Lab6PSP

{

class Program

{

static void Main(string[] args)

{

// Определим нужное максимальное количество потоков

// Пусть будет по 4 на каждый процессор

int MaxThreadsCount = Environment.ProcessorCount \* 4;

// Установим максимальное количество рабочих потоков

ThreadPool.SetMaxThreads(MaxThreadsCount, MaxThreadsCount);

// Установим минимальное количество рабочих потоков

ThreadPool.SetMinThreads(2, 2);

new Server(8080);

}

}

}

**Листинг Server.cs:**

using System;

using System.IO;

using System.Net;

using System.Net.Security;

using System.Net.Sockets;

using System.Security.Authentication;

using System.Security.Cryptography;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading;

namespace Lab6PSP

{

class Server

{

readonly TcpListener Listener;

static X509Certificate2 serverCertificate = null;

public Server(int Port)

{

string s = IPAddress.Any.ToString();

serverCertificate = new X509Certificate2("E:/studies/4/ПСП/ЛАБ\_6/lab\_5\_6/RootCA.pfx", "password");

Listener = new TcpListener(IPAddress.Any, Port);

Listener.Start();

Console.WriteLine("Server is running. https://localhost:" + Port);

while (true)

{

ThreadPool.QueueUserWorkItem(new WaitCallback(ClientThread), Listener.AcceptTcpClient());

}

}

static void ClientThread(object StateInfo)

{

TcpClient Client = (TcpClient)StateInfo;

SslStream sslStream = new SslStream(Client.GetStream(), false);

try

{

sslStream.AuthenticateAsServer(serverCertificate, false, SslProtocols.Tls12, false);

string Request = "";

byte[] Buffer = new byte[1024];

int Count;

while ((Count = sslStream.Read(Buffer, 0, Buffer.Length)) > 0)

{

Request += Encoding.UTF8.GetString(Buffer, 0, Count);

if (Request.IndexOf("\r\n\r\n") >= 0 || Request.Length > 4096)

{

break;

}

}

Match ReqMatch = Regex.Match(Request, @"^\w+\s+([^\s]+)[^\s]\*\s+HTTP/.\*|");

if (ReqMatch == Match.Empty)

{

SendError(Client, 400);

return;

}

string RequestUri = ReqMatch.Groups[1].Value;

RequestUri = Uri.UnescapeDataString(RequestUri);

if (RequestUri.IndexOf("..") >= 0)

{

SendError(Client, 400);

return;

}

string Headers;

byte[] HeadersBuffer;

if (RequestUri.EndsWith("/") ||

RequestUri.EndsWith(".css") ||

RequestUri.EndsWith(".html") ||

RequestUri.EndsWith(".js") ||

RequestUri.EndsWith(".svg") ||

RequestUri.EndsWith(".ico") ||

RequestUri.EndsWith(".htm"))

{

if (RequestUri.EndsWith("/"))

{

RequestUri += "index.html";

}

string FilePath = "Resources/" + RequestUri;

if (!File.Exists(FilePath))

{

SendError(Client, 404);

return;

}

string Extension = RequestUri.Substring(RequestUri.LastIndexOf('.'));

string ContentType = "";

switch (Extension)

{

case ".htm":

case ".html":

ContentType = "text/html";

break;

case ".css":

ContentType = "text/stylesheet";

break;

case ".js":

ContentType = "text/javascript";

break;

case ".jpg":

ContentType = "image/jpeg";

break;

case ".svg":

ContentType = "image/svg+xml";

break;

case ".jpeg":

case ".png":

case ".gif":

ContentType = "image/" + Extension.Substring(1);

break;

default:

if (Extension.Length > 1)

{

ContentType = "application/" + Extension.Substring(1);

}

else

{

ContentType = "application/unknown";

}

break;

}

FileStream FS;

try

{

FS = new FileStream(FilePath, FileMode.Open, FileAccess.Read, FileShare.Read);

}

catch (Exception)

{

SendError(Client, 500);

return;

}

Headers = "HTTP/1.1 200 OK\nContent-Type: " + ContentType + "; charset=utf-8" + "\nContent-Length: " + FS.Length + "\n\n";

HeadersBuffer = Encoding.UTF8.GetBytes(Headers);

sslStream.Write(HeadersBuffer, 0, HeadersBuffer.Length);

while (FS.Position < FS.Length)

{

Count = FS.Read(Buffer, 0, Buffer.Length);

sslStream.Write(Buffer, 0, Count);

}

FS.Close();

Client.Close();

}

else if (RequestUri.Length > 1)

{

string text = RequestUri.Remove(0, 1);

string emails = "";

string emailsRegex = "([a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+)";

MatchCollection matches = Regex.Matches(text, emailsRegex);

foreach (Match match in matches)

{

emails += match.Value + "<br/>";

}

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

Headers = "HTTP/1.1 200 OK\nContent-Type: " + "text" + "\nContent-Length: " + message.Length + "\n\n";

HeadersBuffer = Encoding.UTF8.GetBytes(Headers);

sslStream.Write(HeadersBuffer, 0, HeadersBuffer.Length);

sslStream.Write(message, 0, message.Length);

}

}

catch (Exception e)

{

Console.WriteLine(e);

}

finally

{

sslStream.Close();

Client.Close();

}

}

private static void SendError(TcpClient Client, int Code)

{

string CodeStr = Code.ToString() + " " + ((HttpStatusCode)Code).ToString();

string Html = "<html><body><h1>" + CodeStr + "</h1></body></html>";

string Str = "HTTP/1.1 " + CodeStr + "\nContent-type: text/html\nContent-Length:" + Html.Length.ToString() + "\n\n" + Html;

byte[] Buffer = Encoding.UTF8.GetBytes(Str);

Client.GetStream().Write(Buffer, 0, Buffer.Length);

Client.Close();

}

~Server()

{

if (Listener != null)

{

Listener.Stop();

}

}

}

}