ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное бюджетное образовательное учреждение высшего образования "Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича"

Факультет «Инфокоммуникационных сетей и систем»

Кафедра «Программной инженерии и вычислительной техники»

Отчет по курсовой работе

«Разработка почтового клиента»

По дисциплине «Математические методы и вычислительные

алгоритмы современных систем связи»

Выполнил:

Студент группы ИКПИ-51

Колесов Д. С.

Принял:

Коробов С. А.

Санкт-Петербург

2019

**Оглавление**

[Постановка задачи 3](#_Toc4251436)

[Описание инструментов разработки программы 3](#_Toc4251437)

[Разработка архитектуры программы 3](#_Toc4251438)

[Диаграмма классов 4](#_Toc4251439)

[Работа приложения 5](#_Toc4251440)

[Заключение 10](#_Toc4251441)

[Список используемой литературы 11](#_Toc4251442)

[Листинг кода 12](#_Toc4251443)

[SocketManagment 12](#_Toc4251444)

[EnteringPanelController 14](#_Toc4251445)

[MainPanelController 15](#_Toc4251446)

[SendPanelController 17](#_Toc4251447)

[DialogWindowController 19](#_Toc4251448)

[POPSocket 19](#_Toc4251449)

[SMTPSocket 31](#_Toc4251450)

[ShortContent 34](#_Toc4251451)

[MailShortInfo 35](#_Toc4251452)

[AttachmentGet 36](#_Toc4251453)

[AttachmentSend 36](#_Toc4251454)

[AppConstants 37](#_Toc4251455)

[PostClientEvents 37](#_Toc4251456)

Постановка задачи

Разработать клиентское ПО для получения/отправки почтовых сообщений. Приложение должно реализовывать следующие задачи:

* Запрос статистики электронных сообщений у сервера;
* Получение сообщений с вложенными файлами по протоколу IMAP или POP3;
* Отправка сообщений с вложенными файлами по протоколу SMTP;
* Удаление писем с почтового сервера.

Приложение должно использовать только безопасные (зашифрованные соединения), иметь графический интерфейс, достаточный для выполнения всех задач.

Описание инструментов разработки программы

В качестве инструмента для разработки почтового клиента была выбрана межплатформенная среда разработки Unity, которая позволяет создавать приложения, работающие под более чем 20 различными операционными системами. Unity использует языки программирования C# и JavaScript.

Взаимодействие с почтовым сервером осуществляется посредством протоколов SMTP и POP3:

* SMTP – базовый протокол доставки электронной почты в сетях TCP/IP;
* POP3 – это сетевой протокол, предназначенный для управления почтовыми ящиками и сообщениями на почтовом сервере.

Оба протокола для связи с сервером используют криптографический протокол SSL, который позволяет установить зашифрованное соединение.

Для создания подключения в языке C# используются классы TcpClient и SslStream.

Разработка интерфейса приложения осуществляется с помощью визуального редактора Unity.

Разработка архитектуры программы

Для взаимодействия с почтовым сервером программа должна:

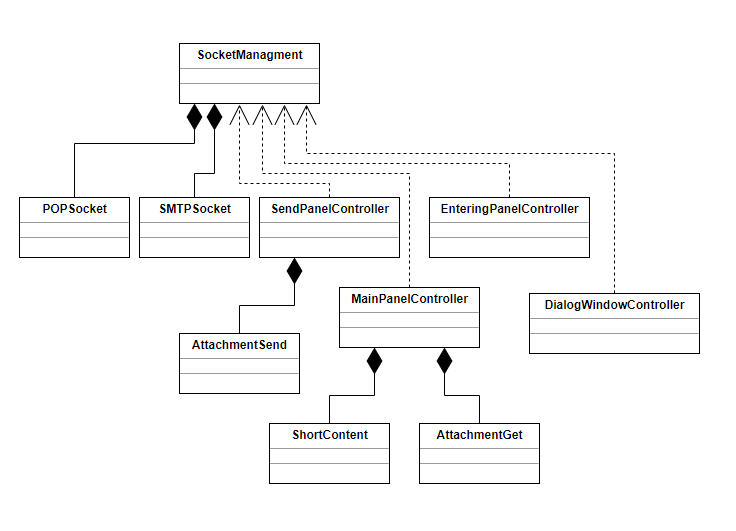
* Удовлетворять стандартам SMTP и POP3;
* Иметь многопоточную структуру, чтобы сбор информации об обновлениях почтового ящика и вход в приложение не замораживали интерфейс пользователя. Это достигается с помощью встроенного в Unity механизма сопрограмм (Coroutines);
* Иметь различные окна для разных задач: вход в приложение, отправка письма, главное окно для просмотра сообщений;

В соответствии с вышеприведенными задачами были разработаны следующие элементы:

* SocketManagment – главный класс для управления клиентами SMTP и POP3, а также окнами приложения;
* EnteringPanelController, MainPanelController, SendPanelController, DialogWindowController – класс управления для каждого окна приложения;
* POPSocket, SMTPSocket – класс SMTP и POP3 клиентов;
* ShortContent – класс для управления элементом представления короткой информации о письме;
* MailShortInfo – структура с полями короткой информации о письме;
* AttachmentGet, AttachmentSend – класс для управления элементами представления прикреплений;
* AppConstants, PostClientEvents – классы констант и событий приложения;
* Messenger – класс мессенджера Unity, разработанный сообществом;

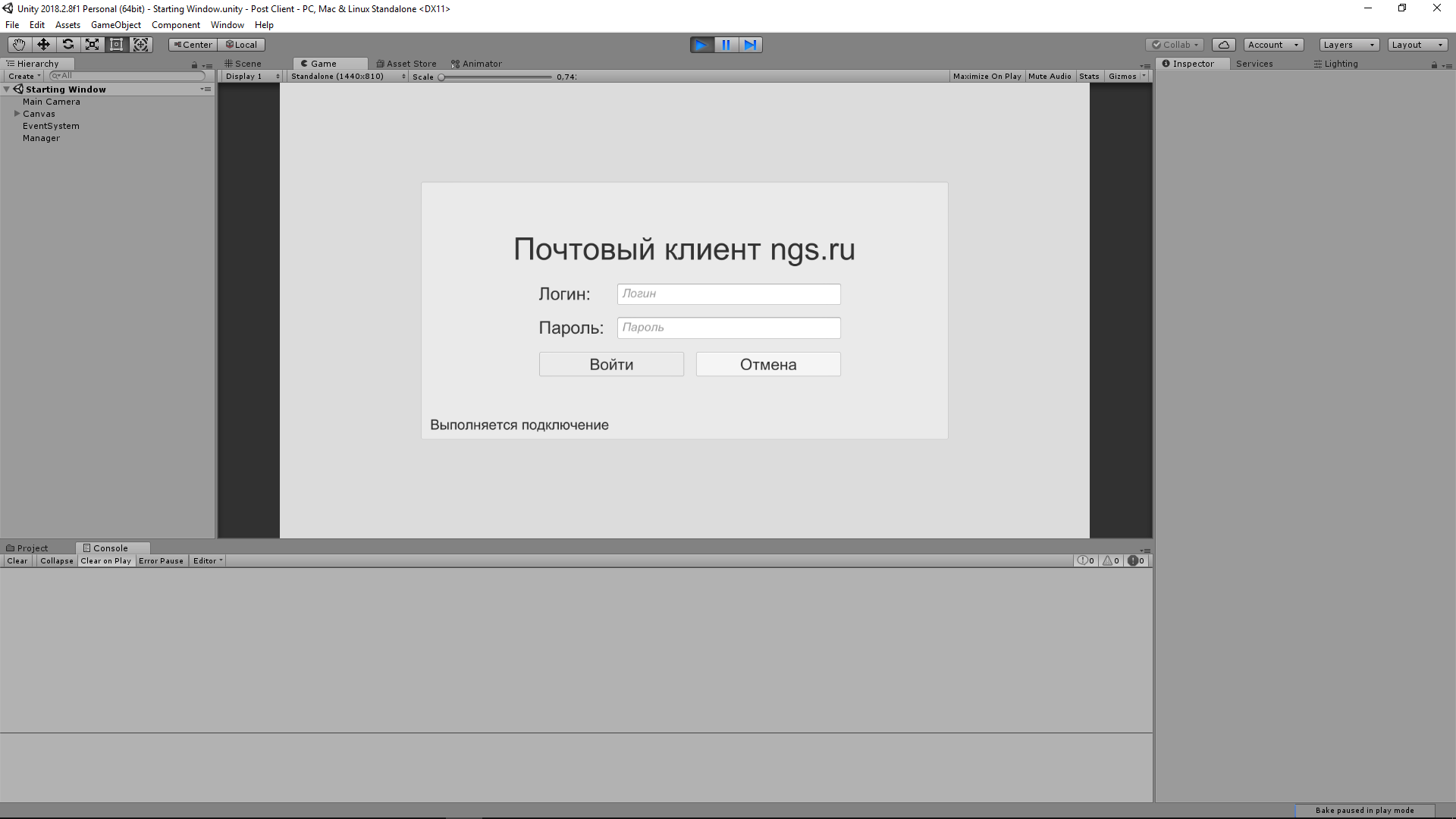
Диаграмма классов

Ниже представлена диаграмма классов приложения.



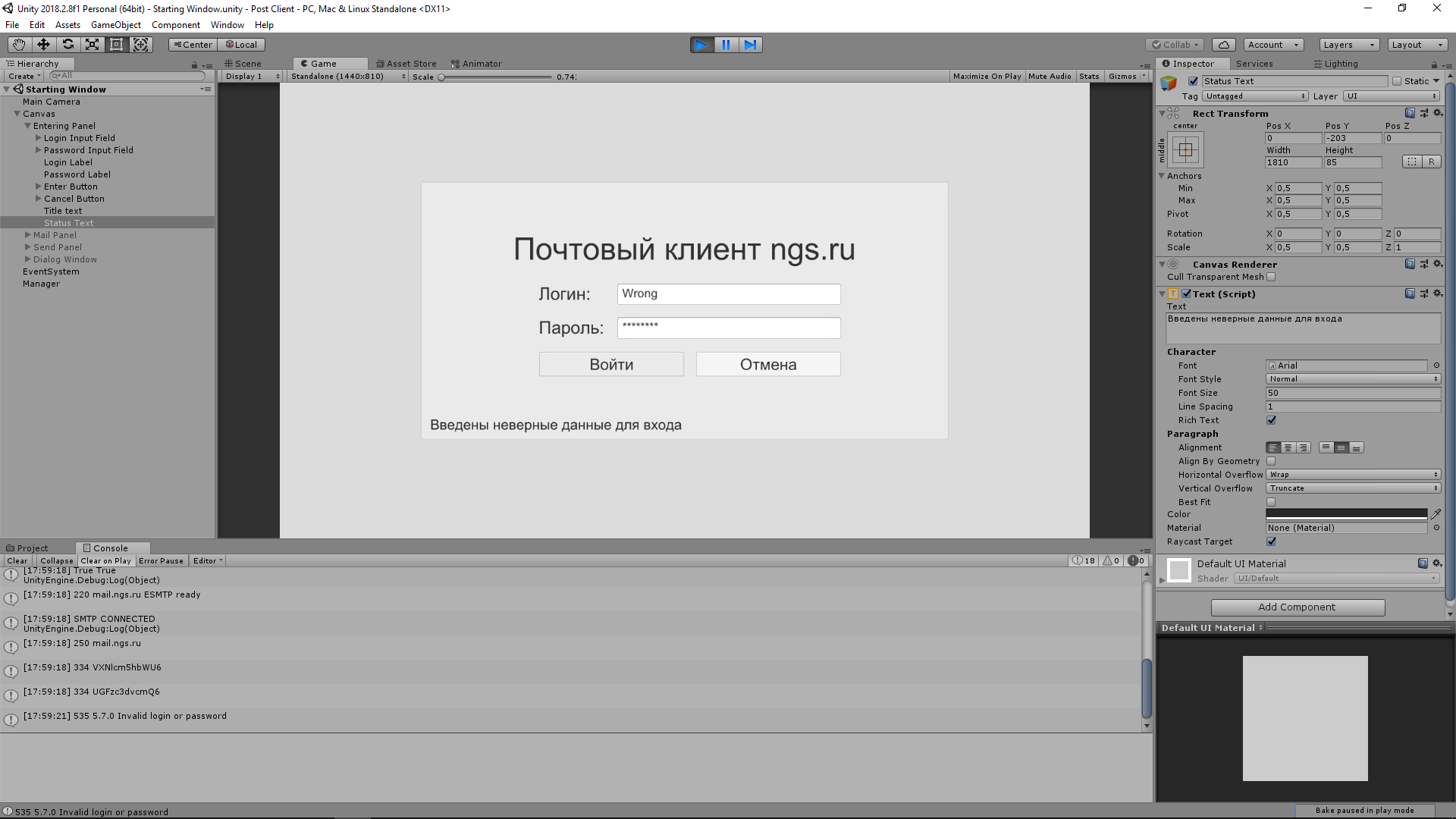
Работа приложения

При входе в приложение, открывается начальное окно авторизации и выполняется подключение к серверам. Для входа в приложение следует ввести логин, пароль и нажать кнопку «Войти». Для выхода из приложения следует нажать кнопку «Отмена».

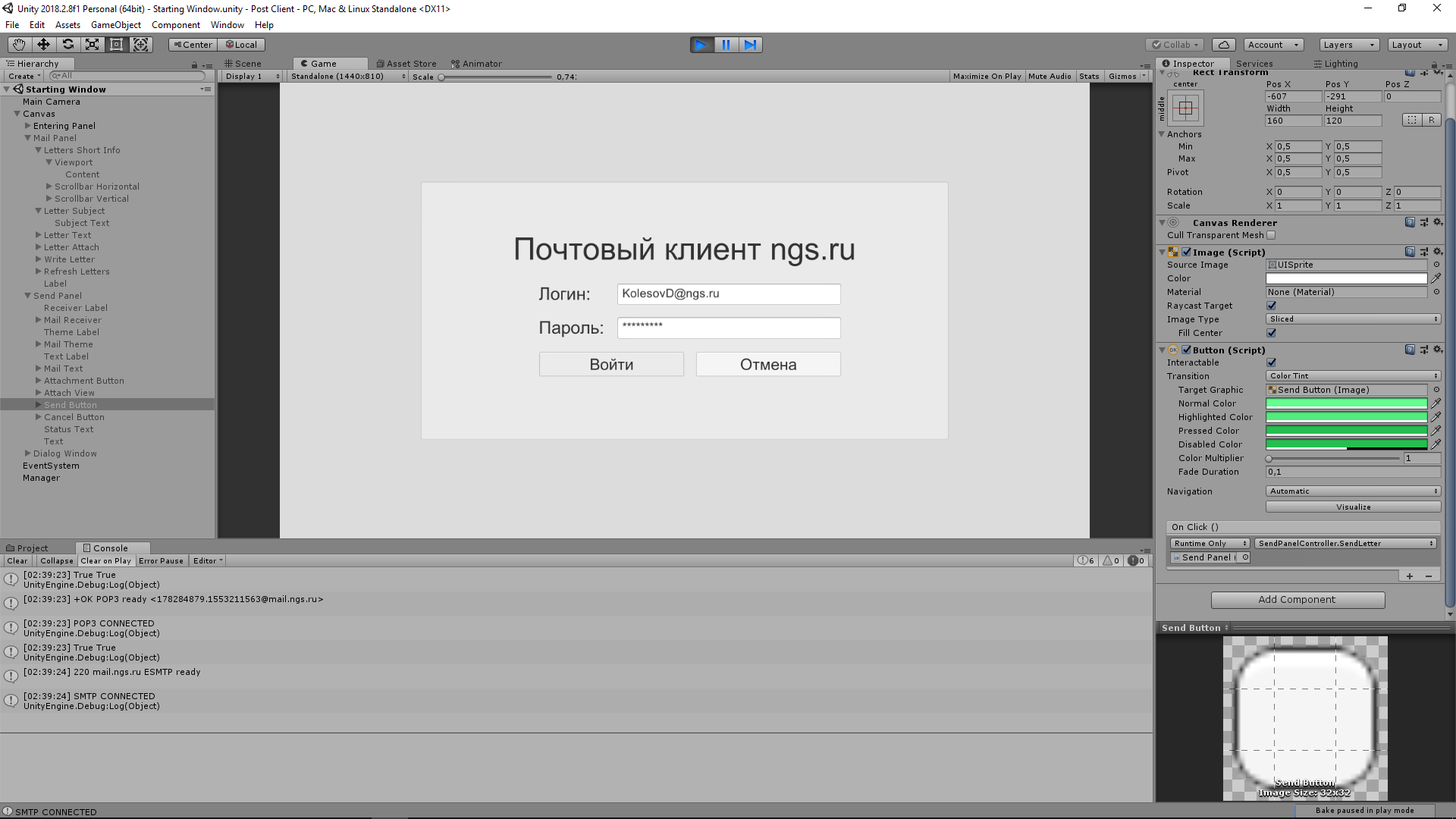


После того, как подключение выполнена, необходимо ввести логин и пароль для входа в приложение.

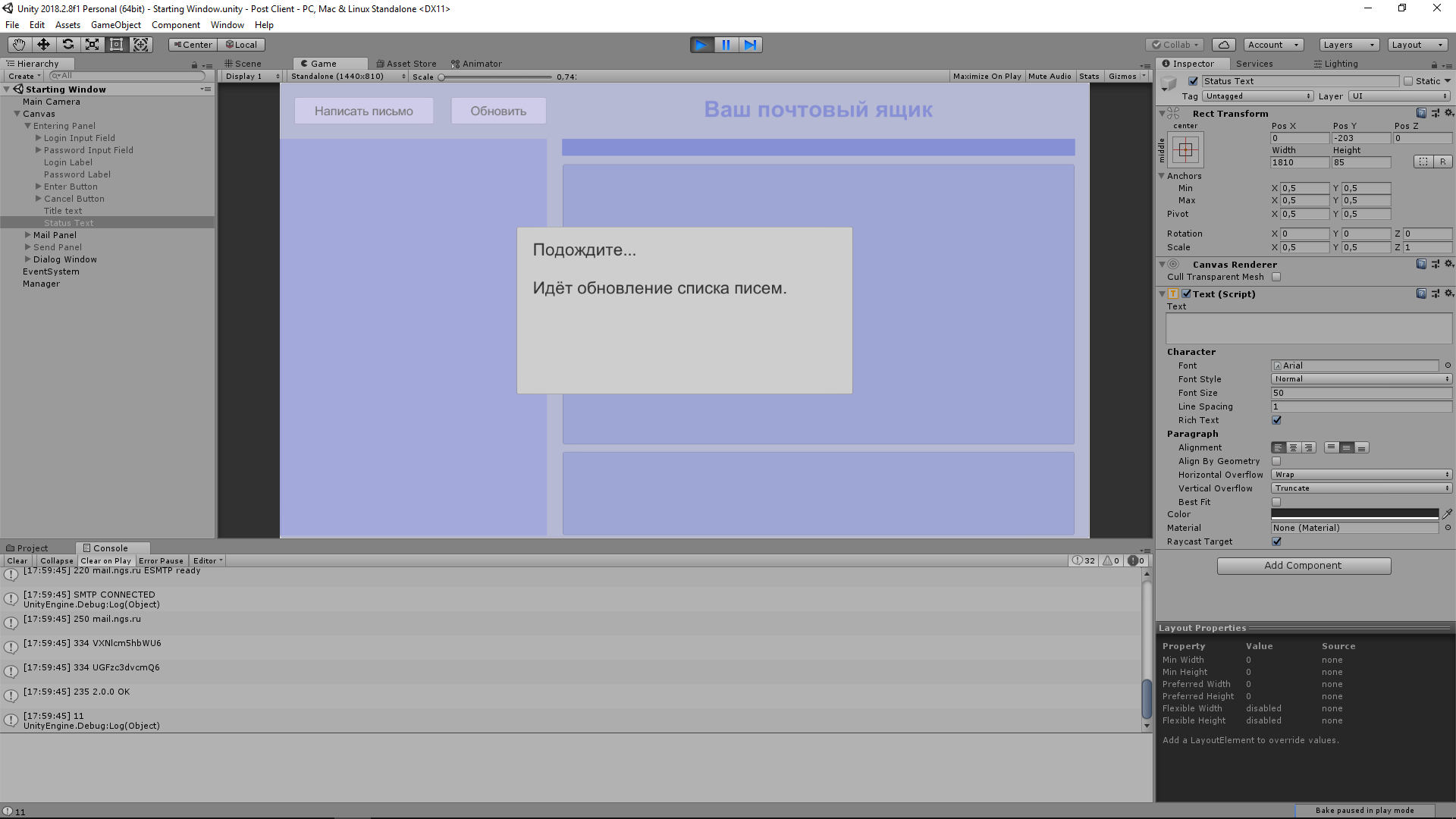
При вводе неправильного логина и пароля, приложение сообщит об этом пользователю.



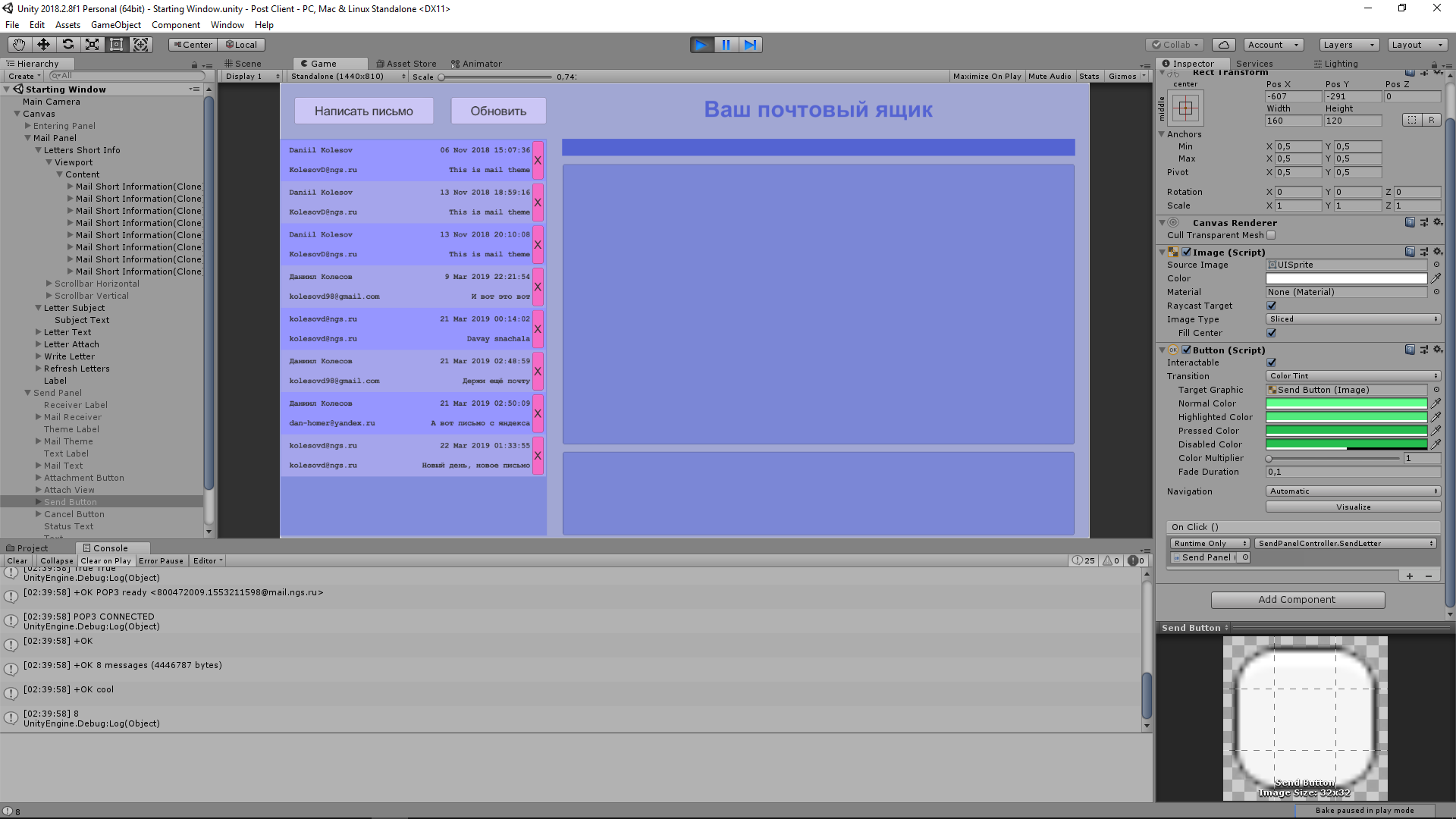
После ввода правильных данных следует нажать на кнопку «Войти» и приложение выполнит вход.



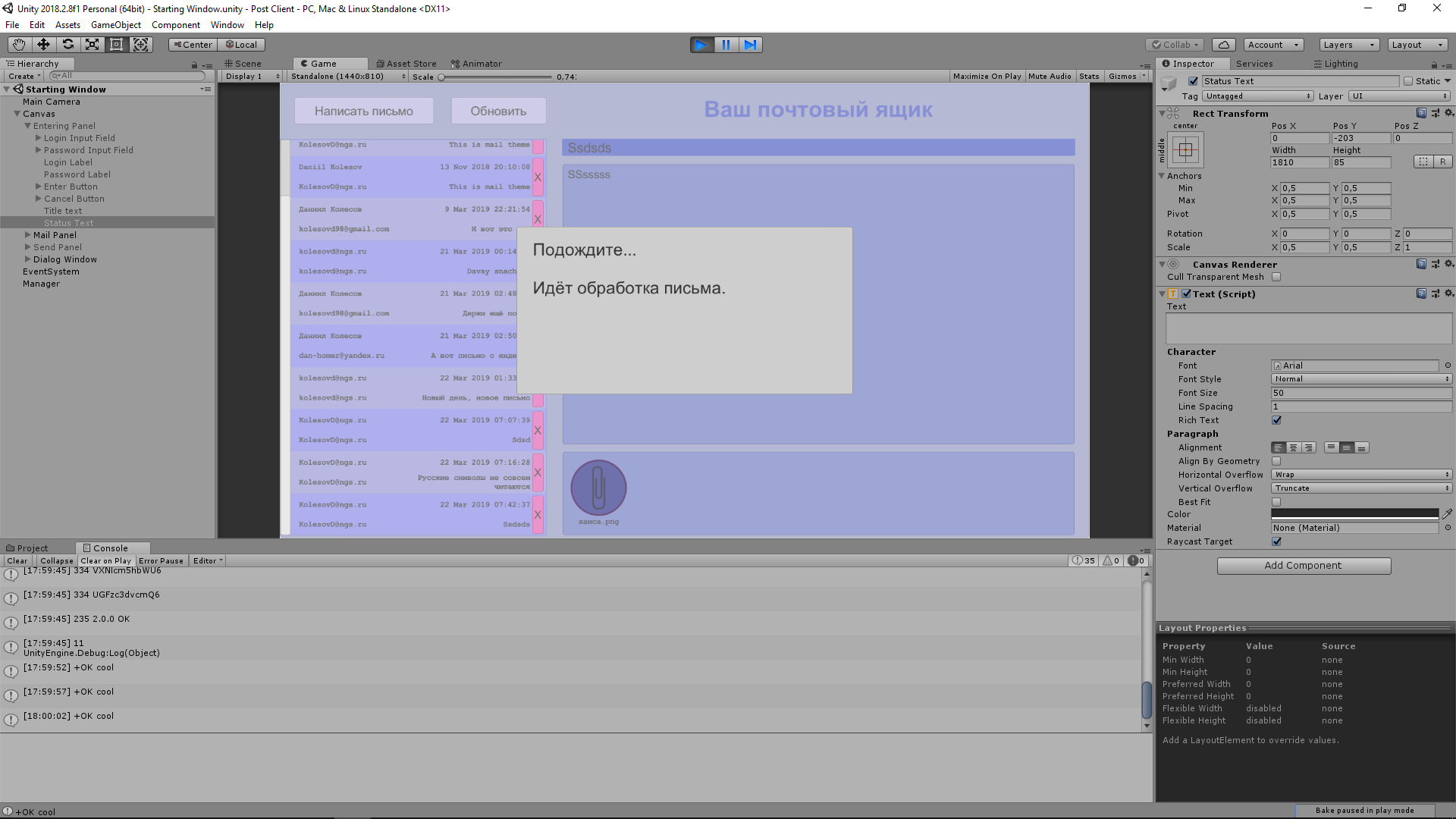
После входа, произойдёт автоматическое обновление списка писем.



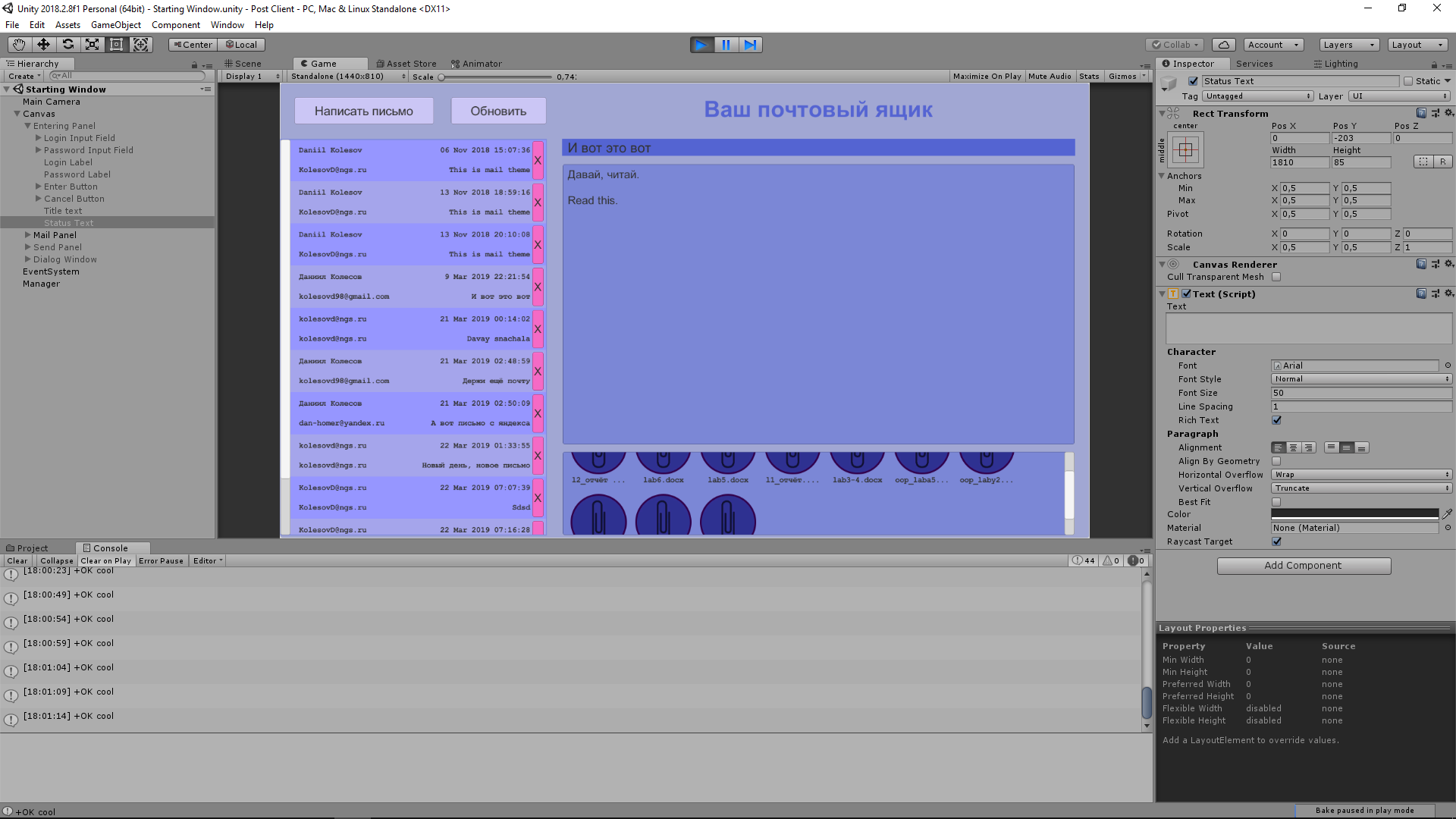
Для перехода в окно отправки письма следует нажать кнопку «Написать письмо». Для того чтобы обновить список писем следует нажать кнопку «Обновить». Для удаления письма следует нажать на кнопку с крестиком рядом с информацией о письме. Для прочтения письма следует нажать на краткую информацию.



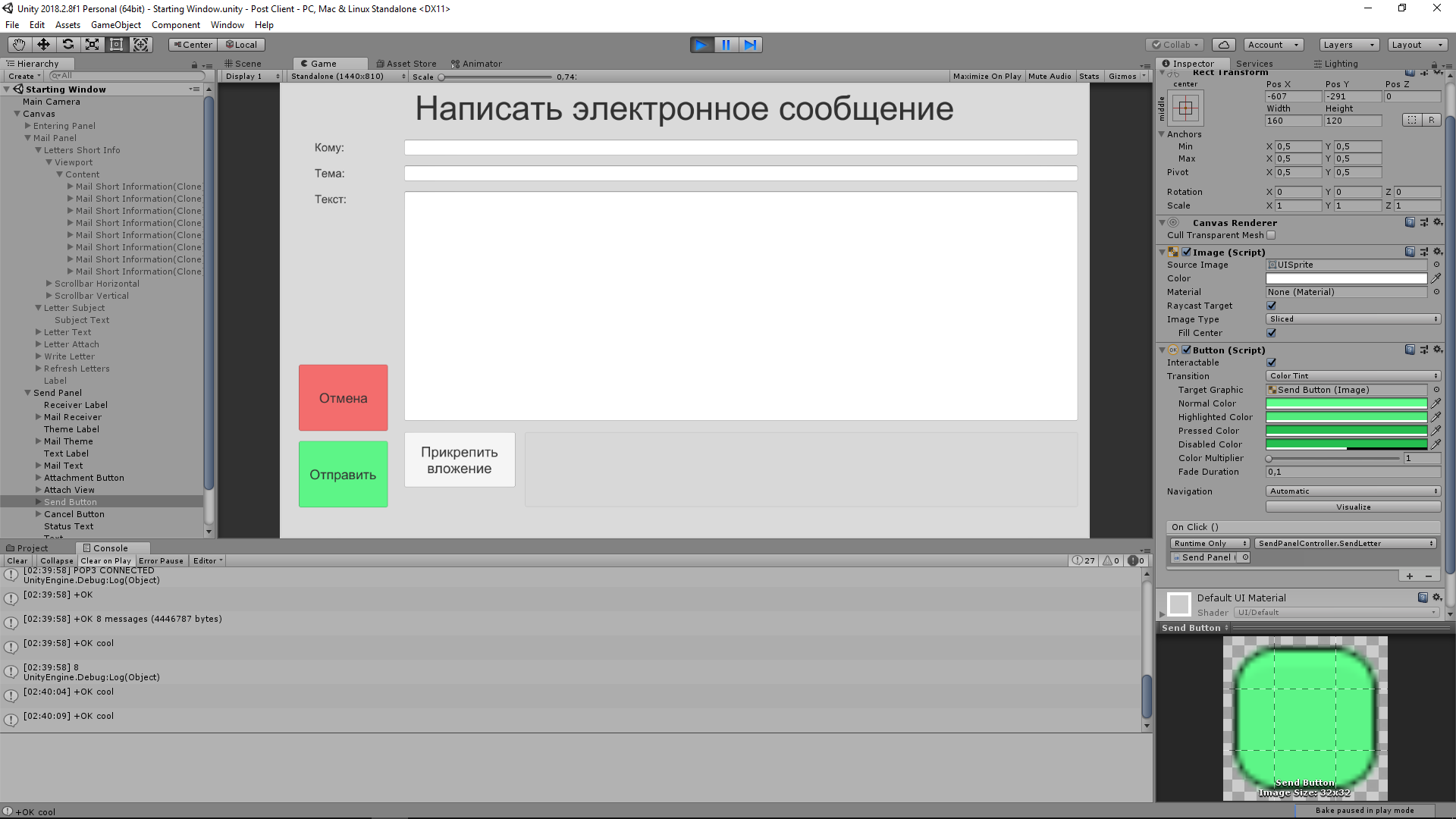
После нажатия на область с краткой информацией о письме, происходит его загрузка.



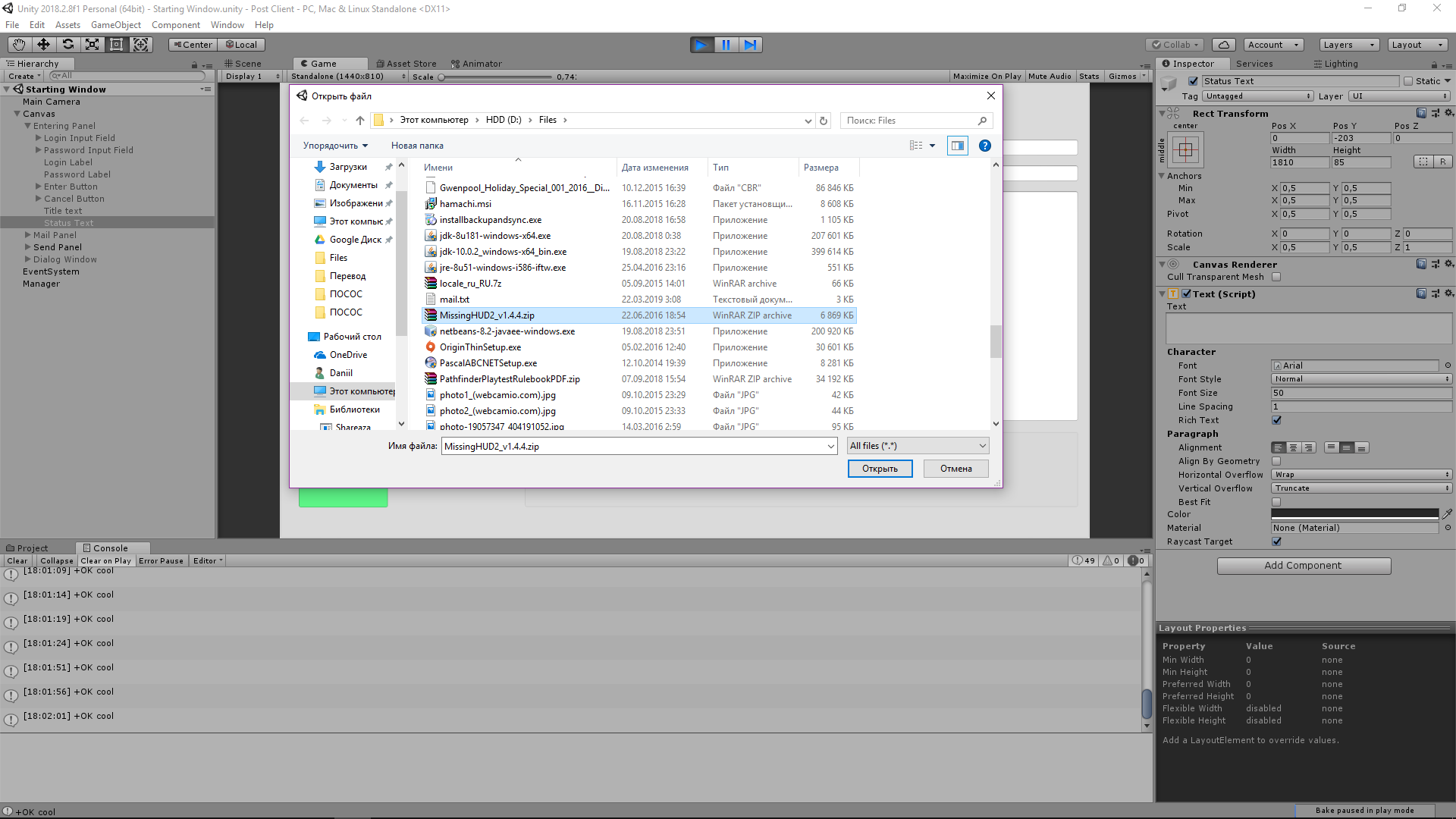
В верхней полосе отображается заголовок, а дальше – текст. Снизу идёт панель с вложениями.



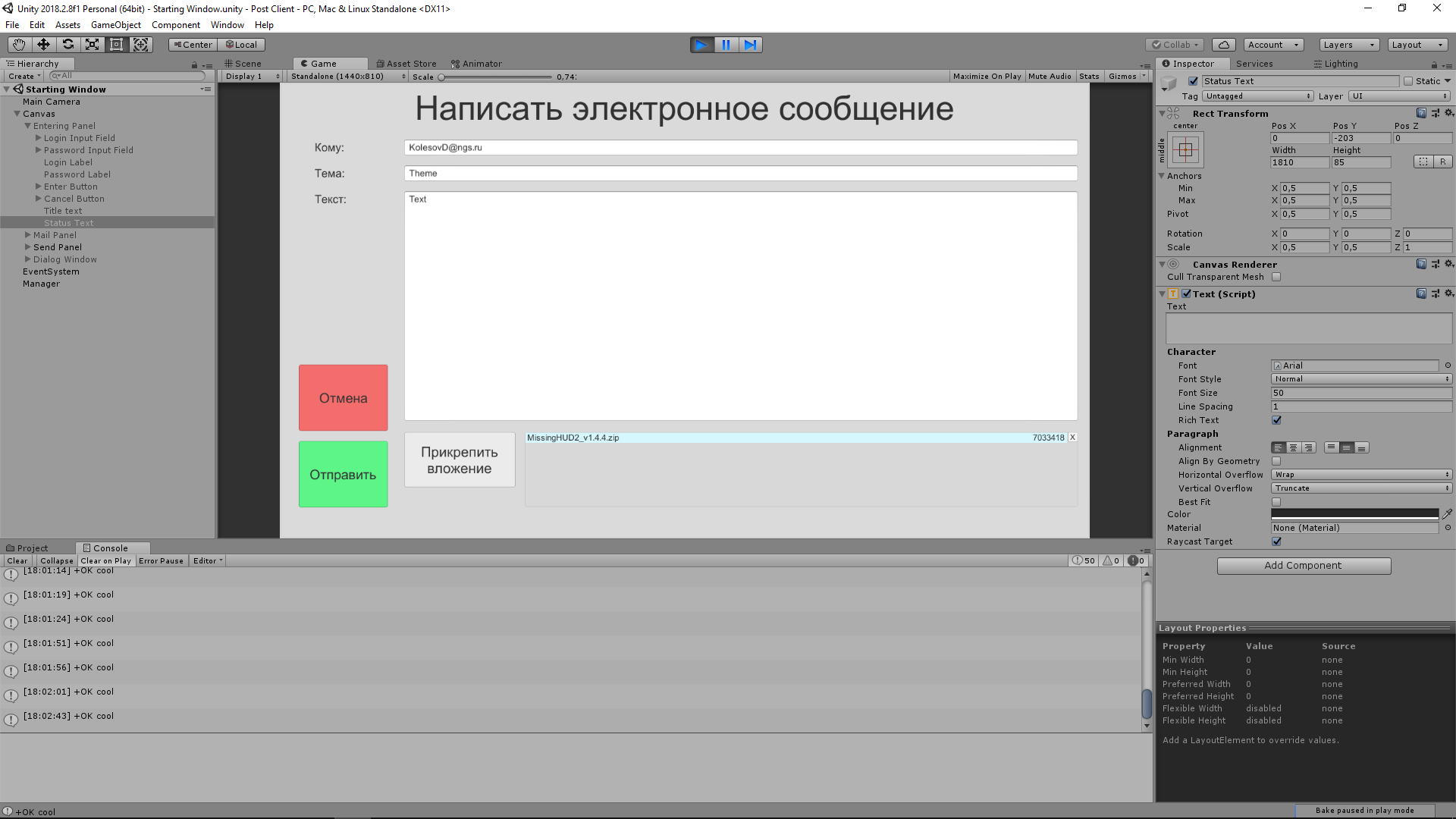
В окне отправки сообщения есть поля получателя, темы и текста сообщения. Для прикрепления вложений следует нажать кнопку «Прикрепить вложение». Для отправки письма следует нажать кнопку «Отправить». Для выхода на главное окно следует нажать кнопку «Отмена».



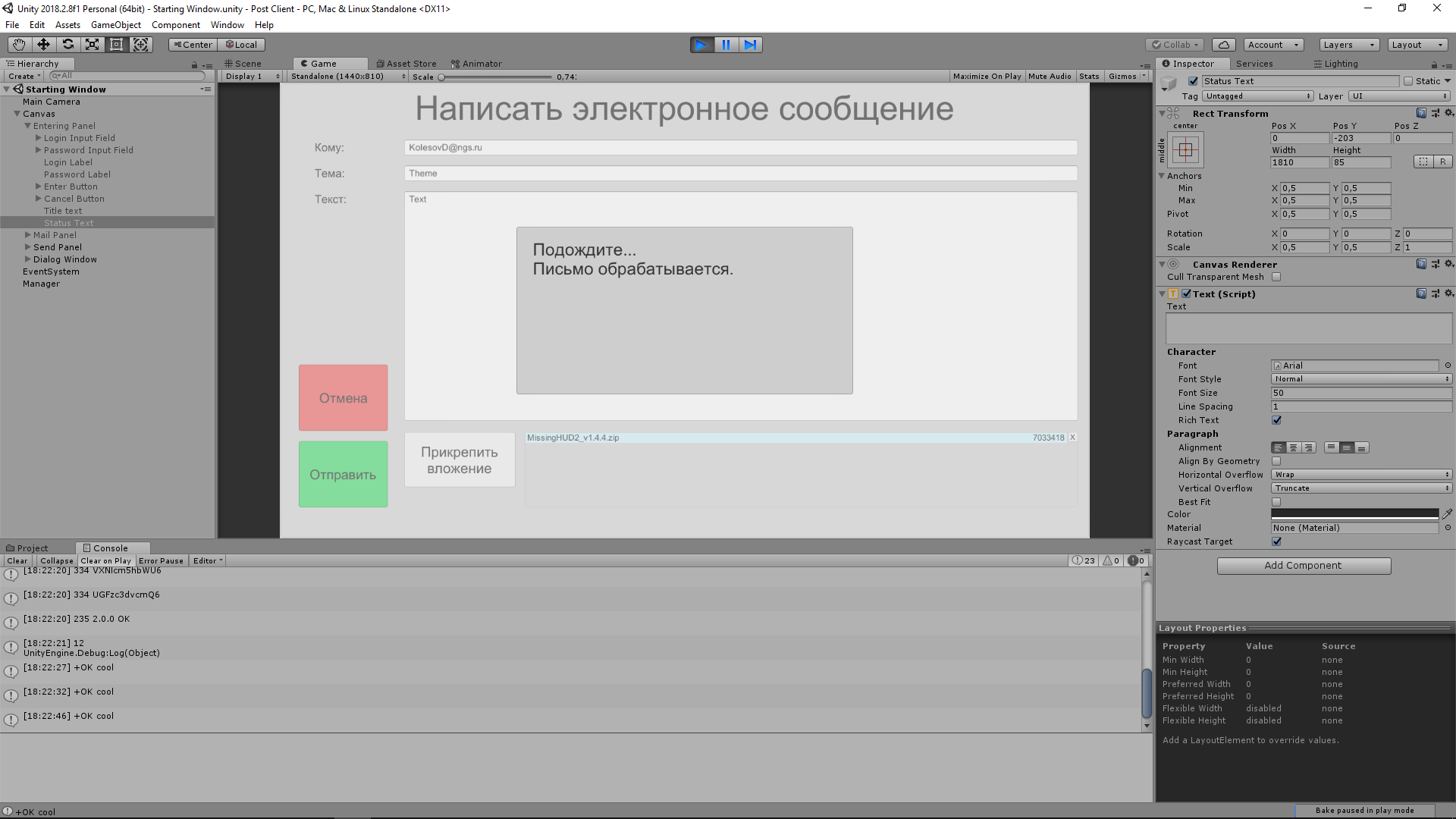
При нажатии кнопки «Прикрепить вложение» открывается диалоговое окно с возможностью выбора файла для прикрепления.



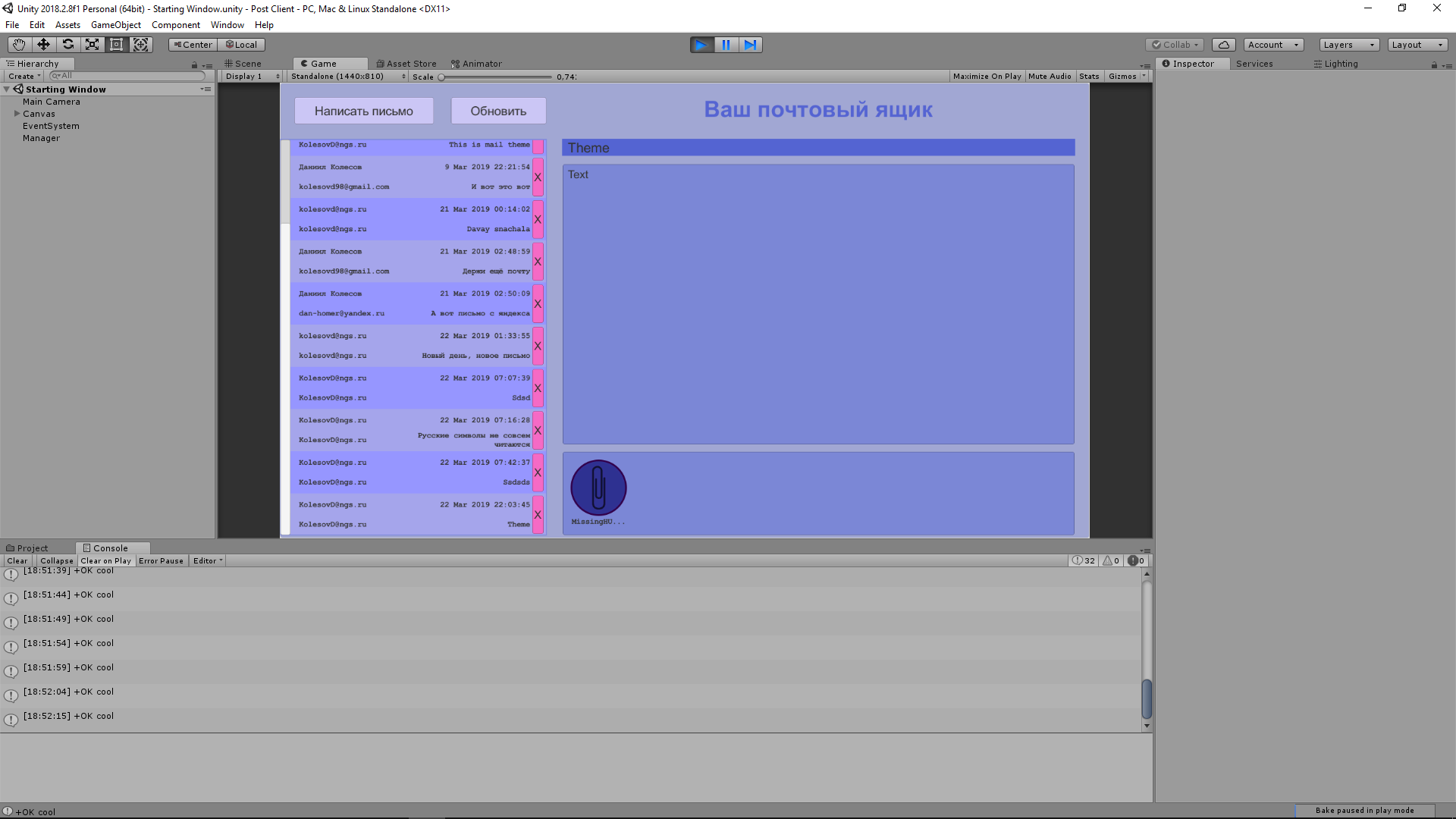
После выбора файла, он прикрепляется к письму.



При нажатии кнопки «Отправить» происходит формирование письма и его отправка.



Отправленное письмо в почтовом ящике.



Заключение

В ходе решения задачи был разработан почтовый клиент, который позволяет отслеживать статистику почтовых ящиков и отправлять как обычные сообщения с простым текстом на почтовый сервер, так и сложные сообщения с вложениями.

Список используемой литературы

1. Джозеф Хокинг, Unity в действии,2016. – 336 с.
2. Документация Unity, https://docs.unity3d.com/ru/current/Manual/index.html
3. Спецификация IMAP, https://tools.ietf.org/html/rfc3501
4. Спецификация SMTP, https://tools.ietf.org/html/rfc821

Листинг кода

SocketManagment

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

[RequireComponent(typeof(POPSocket))]

[RequireComponent(typeof(SMTPSocket))]

public class SocketManagment : MonoBehaviour {

[SerializeField] private GameObject enteringPanel;

[SerializeField] private GameObject readMailPanel;

[SerializeField] private GameObject sendMailPanel;

[SerializeField] private GameObject dialogWindow;

private POPSocket \_pop;

private SMTPSocket \_smtp;

private void Awake()

{

\_pop = GetComponent<POPSocket>();

\_smtp = GetComponent<SMTPSocket>();

Messenger<string, string>.AddListener(PostClientEvents.START\_POP3\_SESSION, InializeSockets);

Messenger<int>.AddListener(PostClientEvents.FORM\_LETTER, ShowMailInformation);

Messenger<int>.AddListener(PostClientEvents.DELETE\_LETTER, DeleteMail);

Messenger.AddListener(PostClientEvents.OPEN\_SENDMAIL\_PANEL, SwitchFromViewToSend);

Messenger.AddListener(PostClientEvents.CLOSE\_SENDMAIL\_PANEL, SwitchFromSendToView);

Messenger<string>.AddListener(PostClientEvents.SHOW\_DIALOG\_WINDOW, OpenDialogWindow);

Messenger.AddListener(PostClientEvents.CLOSE\_DIALOG\_WINDOW, CloseDialogWindow);

Messenger.AddListener(PostClientEvents.CONNECTION\_PROBLEMS, ConnectionProblems);

Messenger<bool>.AddListener(PostClientEvents.CONNECTION\_PROBLEMS\_WITH\_TURNING\_MESSAGE, ConnectionProblems);

Messenger<string, string>.AddListener(PostClientEvents.SEND\_LETTER, SendLetter);

}

void Start()

{

readMailPanel.SetActive(false);

sendMailPanel.SetActive(false);

dialogWindow.SetActive(false);

}

private void OnDestroy()

{

\_pop.CloseSocket();

\_smtp.CloseSocket();

Messenger<string, string>.RemoveListener(PostClientEvents.START\_POP3\_SESSION, InializeSockets);

Messenger<int>.RemoveListener(PostClientEvents.FORM\_LETTER, ShowMailInformation);

Messenger<int>.RemoveListener(PostClientEvents.DELETE\_LETTER, DeleteMail);

Messenger.RemoveListener(PostClientEvents.OPEN\_SENDMAIL\_PANEL, SwitchFromViewToSend);

Messenger.RemoveListener(PostClientEvents.CLOSE\_SENDMAIL\_PANEL, SwitchFromSendToView);

Messenger<string>.RemoveListener(PostClientEvents.SHOW\_DIALOG\_WINDOW, OpenDialogWindow);

Messenger.RemoveListener(PostClientEvents.CLOSE\_DIALOG\_WINDOW, CloseDialogWindow);

Messenger.RemoveListener(PostClientEvents.CONNECTION\_PROBLEMS, ConnectionProblems);

Messenger<bool>.RemoveListener(PostClientEvents.CONNECTION\_PROBLEMS\_WITH\_TURNING\_MESSAGE, ConnectionProblems);

Messenger<string, string>.RemoveListener(PostClientEvents.SEND\_LETTER, SendLetter);

}

public void InializeSockets(string login, string password)

{

if (!\_pop.IsConnected)

\_pop.Connect();

if (!\_smtp.IsConnected)

\_smtp.Connect();

if (\_pop.IsConnected && \_smtp.IsConnected)

{

bool popWorking = \_pop.StartPOPSession(login, password);

bool smtpWorking = \_smtp.CheckConnection(login, password);

if (popWorking && smtpWorking)

{

enteringPanel.SetActive(false);

readMailPanel.SetActive(true);

UpdateMailList();

}

else if (popWorking || smtpWorking)

{

\_pop.CloseSocket();

\_smtp.CloseSocket();

}

else

{

if (\_pop.IsConnected) Messenger<string>.Broadcast(PostClientEvents.UPDATE\_STATUS, "Введены неверные данные для входа");

}

}

}

public void ShowMailInformation(int number)

{

\_pop.ReadLetter(number);

}

public void UpdateMailList()

{

\_pop.FormMailList();

}

public void DeleteMail (int number)

{

\_pop.DeleteMail(number);

}

public void SwitchFromViewToSend()

{

readMailPanel.SetActive(false);

sendMailPanel.SetActive(true);

}

public void SwitchFromSendToView()

{

readMailPanel.SetActive(true);

sendMailPanel.SetActive(false);

}

public void SendLetter(string receiver, string letter)

{

\_smtp.WriteALetter(receiver, letter);

}

public void ConnectionProblems()

{

\_smtp.CloseSocket();

\_pop.CloseSocket();

enteringPanel.SetActive(true);

readMailPanel.SetActive(false);

sendMailPanel.SetActive(false);

dialogWindow.SetActive(false);

enteringPanel.GetComponent<EnteringPanelController>().UpdateStatusString("Проблемы с соединением");

}

public void ConnectionProblems(bool needMessage)

{

\_smtp.CloseSocket();

\_pop.CloseSocket();

enteringPanel.SetActive(true);

readMailPanel.SetActive(false);

sendMailPanel.SetActive(false);

dialogWindow.SetActive(false);

if (needMessage) enteringPanel.GetComponent<EnteringPanelController>().UpdateStatusString("Проблемы с соединением");

}

public void OpenDialogWindow(string message)

{

dialogWindow.GetComponent<DialogWindowController>().SetDialogWindowText(message);

dialogWindow.SetActive(true);

}

public void CloseDialogWindow()

{

dialogWindow.SetActive(false);

}

}

EnteringPanelController

using System.Collections;

using System.Collections.Generic;

using System.IO;

using UnityEngine;

using UnityEngine.UI;

public class EnteringPanelController : MonoBehaviour {

[SerializeField] private InputField loginInput;

[SerializeField] private InputField passwordInput;

[SerializeField] private Text statusLabel;

private void Awake()

{

Messenger<string>.AddListener(PostClientEvents.UPDATE\_STATUS, UpdateStatusString);

Messenger.AddListener(PostClientEvents.CLEAR\_STATUS, ClearStatusString);

}

private void OnDestroy()

{

Messenger<string>.RemoveListener(PostClientEvents.UPDATE\_STATUS, UpdateStatusString);

Messenger.RemoveListener(PostClientEvents.CLEAR\_STATUS, ClearStatusString);

}

public void SendLoginAndPassword()

{

string login = loginInput.text;

string password = passwordInput.text;

ClearStatusString();

Messenger<string, string>.Broadcast(PostClientEvents.START\_POP3\_SESSION, login, password);

Messenger<string>.Broadcast(PostClientEvents.UPDATE\_SENDER\_MAIL, login);

}

public void ClearStatusString()

{

statusLabel.text = "";

}

public void UpdateStatusString(string message)

{

statusLabel.text = message;

}

public void CloseApp()

{

Application.Quit();

}

}

MainPanelController

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class MainPanelController : MonoBehaviour {

[SerializeField] private GameObject content;

[SerializeField] private GameObject mailShorts;

[SerializeField] private Text mailSubjectText;

[SerializeField] private RectTransform contentField;

[SerializeField] private Text mailText;

[SerializeField] private GameObject attachment;

[SerializeField] private GameObject attachPanel;

private List<GameObject> \_shortMailInfo;

private List<GameObject> \_attachments;

private const int \_attachHeight = 100;

private const int \_attachWidth = 100;

private const int \_attachBound = 15;

private const int \_attachNameHeight = 20;

public int numberOfMails { get

{

return \_shortMailInfo.Count;

} }

private void Start () {

\_shortMailInfo = new List<GameObject>();

\_attachments = new List<GameObject>();

}

private void Awake()

{

Messenger<List<MailShortInfo>>.AddListener(PostClientEvents.FORM\_LIST\_OF\_MAILS, FormListOfMails);

Messenger<List<string[]>>.AddListener(PostClientEvents.SHOW\_LETTER, ShowLetter);

}

private void OnDestroy()

{

Messenger<List<MailShortInfo>>.RemoveListener(PostClientEvents.FORM\_LIST\_OF\_MAILS, FormListOfMails);

Messenger<List<string[]>>.RemoveListener(PostClientEvents.SHOW\_LETTER, ShowLetter);

}

public void FormListOfMails(List<MailShortInfo> mailsShortInfo)

{

foreach (GameObject mailShort in \_shortMailInfo)

Destroy(mailShort);

\_shortMailInfo.Clear();

int numberOfLists = mailsShortInfo.Count;

content.GetComponent<RectTransform>().sizeDelta = new Vector2 (mailShorts.GetComponent<RectTransform>().sizeDelta.x, 0);

for (int i = 0; i < numberOfLists; i++)

{

if (!mailsShortInfo[i].sendDate.Equals(""))

{

Vector2 contentSize = content.GetComponent<RectTransform>().sizeDelta;

contentSize.y += mailShorts.GetComponent<RectTransform>().sizeDelta.y;

content.GetComponent<RectTransform>().sizeDelta = contentSize;

GameObject nextMailShort = Instantiate(mailShorts) as GameObject;

nextMailShort.transform.SetParent(content.transform);

Vector3 contentPos = nextMailShort.transform.localPosition;

contentPos.x = mailShorts.transform.position.x;

contentPos.y = -mailShorts.GetComponent<RectTransform>().sizeDelta.y \* i;

nextMailShort.transform.localPosition = contentPos;

nextMailShort.GetComponent<RectTransform>().sizeDelta = mailShorts.GetComponent<RectTransform>().sizeDelta;

nextMailShort.GetComponent<ShortContent>().SetText(mailsShortInfo[i].senderName, mailsShortInfo[i].senderMail, mailsShortInfo[i].sendDate, mailsShortInfo[i].senderedSubject);

if (i % 2 == 0)

nextMailShort.GetComponent<ShortContent>().SetColor(new Color32(150, 150, 255, 255));

else nextMailShort.GetComponent<ShortContent>().SetColor(new Color32(165, 165, 235, 255));

nextMailShort.GetComponent<ShortContent>().letterNumber = i + 1;

\_shortMailInfo.Add(nextMailShort);

}

else Debug.Log(mailsShortInfo[i].senderMail + " equals null");

}

Messenger.Broadcast(PostClientEvents.CLOSE\_DIALOG\_WINDOW);

}

public void ShowLetter(List<string[]> letter)

{

if (letter == null)

{

mailSubjectText.text = "Ошибка чтения письма";

}

else

{

int letterSize = letter.Count;

int attachInARow = (int) (attachPanel.GetComponent<RectTransform>().rect.width - \_attachBound)/(\_attachWidth + \_attachBound);

foreach (GameObject attach in \_attachments)

Destroy(attach);

\_attachments.Clear();

for (int i = 0; i < letterSize; i++)

{

if (letter[i][0] == AppConstants.subjectOfMessage)

mailSubjectText.text = letter[i][1];

else if (letter[i][0] == AppConstants.textPartOfMessage)

{

TextGenerator textGen = new TextGenerator();

TextGenerationSettings rectSetting = mailText.GetGenerationSettings(mailText.rectTransform.rect.size);

contentField.sizeDelta = new Vector2(0, textGen.GetPreferredHeight(letter[i][1], rectSetting) + 16);

mailText.text = letter[i][1];

}

else

{

attachPanel.GetComponent<RectTransform>().sizeDelta = new Vector2(0, (\_attachments.Count / 7 + 1) \* (\_attachHeight + \_attachBound + \_attachNameHeight) + \_attachBound);

Vector3 attachPos = new Vector3(\_attachments.Count % attachInARow \* (\_attachWidth + \_attachBound) + \_attachBound + \_attachWidth / 2, -(\_attachments.Count / 7 \* (\_attachHeight + \_attachBound + \_attachNameHeight) + \_attachBound + \_attachHeight / 2), 0);

GameObject nextAttachment = Instantiate(attachment) as GameObject;

nextAttachment.transform.SetParent(attachPanel.transform);

nextAttachment.transform.localPosition = attachPos;

nextAttachment.GetComponent<AttachmentGet>().SetContent(letter[i]);

\_attachments.Add(nextAttachment);

}

}

Messenger.Broadcast(PostClientEvents.CLOSE\_DIALOG\_WINDOW);

}

}

}

SendPanelController

using System.Collections;

using System.Collections.Generic;

using System.IO;

using UnityEngine;

using UnityEngine.UI;

using UnityEditor;

using System.Text.RegularExpressions;

public class SendPanelController : MonoBehaviour {

[SerializeField] private Text receiver;

[SerializeField] private Text mailTheme;

[SerializeField] private Text mailText;

[SerializeField] private Text statusText;

[SerializeField] private RectTransform attachView;

[SerializeField] private GameObject attachPanel;

[SerializeField] private GameObject attachExample;

private List<GameObject> \_attachToSendList;

private string senderMail;

void Awake()

{

Messenger<GameObject>.AddListener(PostClientEvents.DELETE\_ATTACHMENT\_TO\_SEND, DeleteAttach);

Messenger<string>.AddListener(PostClientEvents.UPDATE\_SENDER\_MAIL, SetSenderMail);

Messenger.AddListener(PostClientEvents.SMTP\_CLEAR\_INPUTS, ClearInputs);

}

void Start()

{

\_attachToSendList = new List<GameObject>();

senderMail = "default@none.no";

}

void OnDestroy()

{

Messenger<GameObject>.RemoveListener(PostClientEvents.DELETE\_ATTACHMENT\_TO\_SEND, DeleteAttach);

Messenger<string>.RemoveListener(PostClientEvents.UPDATE\_SENDER\_MAIL, SetSenderMail);

Messenger.RemoveListener(PostClientEvents.SMTP\_CLEAR\_INPUTS, ClearInputs);

}

public void SendLetter()

{

StartCoroutine(SendLetterCoroutine());

}

public IEnumerator SendLetterCoroutine()

{

if (receiver.text.Length == 0)

statusText.text = "Не выбран получатель.";

else if (!new Regex("^\\w\*@\\w\*\\.\\w\*").Match(receiver.text).Success)

statusText.text = "Введите корректную почту.";

else if (mailTheme.text.Length == 0)

statusText.text = "Не заполнена тема.";

else if (mailText.text.Length == 0)

statusText.text = "Не заполнен текст письма.";

else

{

statusText.text = "";

Messenger<string>.Broadcast(PostClientEvents.SHOW\_DIALOG\_WINDOW, "Подождите...\n\nПисьмо обрабатывается.");

byte[] theme = System.Text.Encoding.UTF8.GetBytes(mailTheme.text);

byte[] text = System.Text.Encoding.UTF8.GetBytes(mailText.text);

string sendingContent = string.Concat("Subject: =?UTF-8?B?" + System.Convert.ToBase64String(theme) + "?=\n", "From: <" + senderMail + ">\n",

"To: <" + receiver.text + ">\n", "X-Mailer: DanPostClient\n", "Content-Type: multipart/mixed; boundary=\"globalbound\"\n",

"\n--globalbound\n", "Content-Type: text/plain; charset=\"UTF-8\"\n", "Content-Transfer-Encoding: base64\n",

"\n" + System.Convert.ToBase64String(text) + "\n");

if (\_attachToSendList.Count != 0)

{

foreach (GameObject attach in \_attachToSendList)

{

yield return 0;

sendingContent = string.Concat(sendingContent, "--globalbound\n",

"Content-Type: application/octet-stream; name=\"=?UTF-8?B?" + attach.GetComponent<AttachmentSend>().GetBase64Name() + "?=\"\n",

"Content-Disposition: attachment; filename=\"=?UTF-8?B?" + attach.GetComponent<AttachmentSend>().GetBase64Name() + "?=\"\n",

"Content-Transfer-Encoding: base64\n", "\n" + attach.GetComponent<AttachmentSend>().GetBase64Content() + "\n");

}

}

sendingContent = string.Concat(sendingContent, "--globalbound--\n", ".\n");

Messenger<string, string>.Broadcast(PostClientEvents.SEND\_LETTER, receiver.text, sendingContent);

}

}

public void AddAttach()

{

statusText.text = "";

string path = EditorUtility.OpenFilePanel("Открыть файл", "", "");

if (path.Length != 0)

{

byte[] fileContent = File.ReadAllBytes(path);

GameObject nextAttachment = Instantiate(attachExample) as GameObject;

nextAttachment.transform.SetParent(attachPanel.transform);

Vector2 panelSize = attachPanel.GetComponent<RectTransform>().sizeDelta;

nextAttachment.GetComponent<RectTransform>().sizeDelta = new Vector2(-20, attachExample.GetComponent<RectTransform>().sizeDelta.y);

if (panelSize.y < attachView.sizeDelta.y) nextAttachment.transform.localPosition = new Vector3(482.5f, -10 - (20 \* \_attachToSendList.Count), 0);

else nextAttachment.transform.localPosition = new Vector3(474, -10 - (20 \* \_attachToSendList.Count), 0);

panelSize.y = nextAttachment.GetComponent<RectTransform>().sizeDelta.y \* (\_attachToSendList.Count + 1);

attachPanel.GetComponent<RectTransform>().sizeDelta = panelSize;

if (\_attachToSendList.Count % 2 == 0)

nextAttachment.GetComponent<Image>().color = new Color32(215, 247, 255, 255);

else nextAttachment.GetComponent<Image>().color = new Color32(153, 198, 226, 255);

nextAttachment.GetComponent<AttachmentSend>().SetContent(path.Substring(path.LastIndexOf('/') + 1, path.Length - (path.LastIndexOf('/') + 1)), fileContent.Length, fileContent);

\_attachToSendList.Add(nextAttachment);

}

}

public void RefreshAttaches()

{

statusText.text = "";

float panelSizeY = attachPanel.GetComponent<RectTransform>().sizeDelta.y;

for (int i = 0; i < \_attachToSendList.Count; i++)

{

if (panelSizeY < attachView.sizeDelta.y) \_attachToSendList[i].transform.localPosition = new Vector3(482.5f, -10 - (20 \* i), 0);

else \_attachToSendList[i].transform.localPosition = new Vector3(474, -10 - (20 \* i), 0);

if (i % 2 == 0)

\_attachToSendList[i].GetComponent<Image>().color = new Color32(215, 247, 255, 255);

else \_attachToSendList[i].GetComponent<Image>().color = new Color32(153, 198, 226, 255);

}

attachPanel.GetComponent<RectTransform>().sizeDelta = new Vector2(attachPanel.GetComponent<RectTransform>().sizeDelta.x, attachExample.GetComponent<RectTransform>().sizeDelta.y \* \_attachToSendList.Count);

}

public void DeleteAttach(GameObject attach)

{

statusText.text = "";

\_attachToSendList.Remove(attach);

Destroy(attach);

RefreshAttaches();

}

public void SetSenderMail(string lastSenderMail)

{

senderMail = lastSenderMail;

}

public void ClosePanel()

{

ClearInputs();

Messenger.Broadcast(PostClientEvents.CLOSE\_SENDMAIL\_PANEL);

}

public void ClearInputs()

{

receiver.GetComponentInParent<InputField>().text = "";

mailTheme.GetComponentInParent<InputField>().text = "";

mailText.GetComponentInParent<InputField>().text = "";

statusText.text = "";

foreach (GameObject attach in \_attachToSendList)

Destroy(attach);

\_attachToSendList.Clear();

}

}

DialogWindowController

using UnityEngine;

using UnityEngine.UI;

public class DialogWindowController : MonoBehaviour {

[SerializeField] private Text dialogWindowText;

public void SetDialogWindowText(string text)

{

dialogWindowText.text = text;

}

}

POPSocket

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System;

using System.Net.Sockets;

using System.Net.Security;

using System.IO;

using System.Text.RegularExpressions;

public class POPSocket : MonoBehaviour {

private TcpClient \_popClient;

private SslStream \_popStream;

private byte[] \_buff = new byte[5000];

private string \_stringBuff;

private int \_numberOfBytesInLastReading;

private int \_readMailIterator;

private string \_stringsReadedInLetter;

private string \_lastLogin;

private string \_lastPassword;

private IEnumerator updateConnection;

private const int \_numberOfStringsToReadPerOneFrame = 50;

public int numberOfLettersInMailbox { get; private set; }

public bool IsConnected { get { return \_popClient==null ? false : \_popClient.Connected; } }

public bool IsLogged { get; private set; }

public bool IsWorking { get; private set; }

private void Awake()

{

\_numberOfBytesInLastReading = 0;

updateConnection = UpdateConnection();

}

private void OnDestroy()

{

CloseSocket();

}

void Start()

{

numberOfLettersInMailbox = 0;

IsLogged = false;

IsWorking = false;

StartCoroutine(StartConnection());

}

public IEnumerator StartConnection()

{

Messenger<string>.Broadcast(PostClientEvents.UPDATE\_STATUS, "Выполняется подключение");

yield return new WaitForSeconds(.5f);

Connect();

}

public void Connect()

{

try

{

if (\_popClient != null)

{

\_popStream.Close();

\_popClient.Close();

}

\_popClient = new TcpClient(AppConstants.pop3ServerAdress, 995);

\_popStream = new SslStream(\_popClient.GetStream());

\_popStream.AuthenticateAsClient(AppConstants.pop3ServerAdress);

Debug.Log(\_popStream.IsAuthenticated + " " + \_popStream.IsSigned);

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

Debug.Log("POP3 CONNECTED");

Messenger.Broadcast(PostClientEvents.CLEAR\_STATUS);

}

catch

{

Messenger<bool>.Broadcast(PostClientEvents.CONNECTION\_PROBLEMS\_WITH\_TURNING\_MESSAGE, false);

Messenger<string>.Broadcast(PostClientEvents.UPDATE\_STATUS, "Попытка подключения POP3 сокета закончилась неудачей");

}

}

public bool StartPOPSession(string login, string password)

{

\_lastLogin = login;

\_lastPassword = password;

if (IsConnected)

{

string answer;

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("USER " + login + "\n"));

ReceiveFromSocket();

answer = GetStringFromSocket();

Debug.Log(answer);

if (!answer[0].Equals('+'))

return IsLogged;

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("PASS " + password + "\n"));

ReceiveFromSocket();

answer = GetStringFromSocket();

Debug.Log(answer);

if (!answer[0].Equals('+'))

return IsLogged;

IsLogged = true;

StartCoroutine(updateConnection);

return IsLogged;

}

else

{

return false;

}

}

public void GetNumberOfLetters()

{

if (IsLogged)

{

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("STAT\n"));

ReceiveFromSocket();

\_stringBuff = GetStringFromSocket();

int numberOfLetters = 0;

int.TryParse(\_stringBuff.Split(' ')[1], out numberOfLetters);

numberOfLettersInMailbox = numberOfLetters;

}

}

public void FormMailList()

{

StartCoroutine(FormMailListCoroutine());

}

public IEnumerator FormMailListCoroutine()

{

if (IsLogged)

{

IsWorking = true;

Messenger<string>.Broadcast(PostClientEvents.SHOW\_DIALOG\_WINDOW, "Подождите...\n\nИдёт обновление списка писем.");

GetNumberOfLetters();

Debug.Log(numberOfLettersInMailbox);

yield return 0;

List<MailShortInfo> mailsShortInfo = new List<MailShortInfo>(numberOfLettersInMailbox);

for (int i = 0; i < numberOfLettersInMailbox; i++)

{

MailShortInfo nextInfo = new MailShortInfo() { senderName = "", senderMail = "", senderedSubject = "", sendDate = "" };

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("TOP " + (i + 1) + " 0\n"));

yield return StartCoroutine(ReadSocketList());

\_stringsReadedInLetter = string.Join("", \_stringsReadedInLetter.Split(new char[] { '\0', '\r' }));

string[] stringArray = \_stringsReadedInLetter.Split('\n');

for (int j = 0; j < stringArray.Length; j++)

{

if (stringArray[j].IndexOf("From:") == 0)

{

string senderMail;

string senderName;

if (stringArray[j].Contains("<"))

{

senderMail = stringArray[j].Substring(stringArray[j].IndexOf('<') + 1, stringArray[j].IndexOf('>') - stringArray[j].IndexOf('<') - 1);

senderName = stringArray[j].IndexOf('<') == 6 ? senderMail : NormalizeString(stringArray[j].Substring(6, stringArray[j].IndexOf('<') - 7));

}

else

{

senderName = NormalizeString(stringArray[j].Substring(6, stringArray[j].Length - 6));

senderMail = senderName;

}

nextInfo.senderName = senderName;

nextInfo.senderMail = senderMail;

}

if (stringArray[j].IndexOf("Subject: ") == 0)

{

string subject = stringArray[j].Substring(9);

for (int k = j; k < stringArray.Length - 1; k++)

if (!stringArray[k + 1].Contains(":"))

subject += stringArray[k + 1];

else break;

subject = Regex.Replace(subject, @"=\?\S\*\?=", (Match match) =>

{

return NormalizeString(match.Value);

});

nextInfo.senderedSubject = subject;

}

if (stringArray[j].IndexOf("Date: ") == 0)

{

string sendDate;

if (stringArray[j].Contains("+"))

sendDate = stringArray[j].Substring(stringArray[j].IndexOf(',') + 2, stringArray[j].IndexOf('+') - (stringArray[j].IndexOf(',') + 3));

else sendDate = stringArray[j].Substring(stringArray[j].IndexOf(',') + 2, stringArray[j].IndexOf('-') - (stringArray[j].IndexOf(',') + 3));

nextInfo.sendDate = sendDate;

}

}

mailsShortInfo.Add(nextInfo);

}

IsWorking = false;

Messenger<List<MailShortInfo>>.Broadcast(PostClientEvents.FORM\_LIST\_OF\_MAILS, mailsShortInfo);

}

}

public void ReadLetter(int number)

{

StartCoroutine(ReadLetterCoroutine(number));

}

private IEnumerator ReadLetterCoroutine(int number)

{

if (IsLogged)

{

IsWorking = true;

GetNumberOfLetters();

IsWorking = false;

if (number <= numberOfLettersInMailbox && number >= 1)

{

IsWorking = true;

Messenger<string>.Broadcast(PostClientEvents.SHOW\_DIALOG\_WINDOW, "Подождите...\n\nИдёт чтение письма.");

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("RETR " + number + "\n"));

yield return StartCoroutine(ReadSocketList());

Messenger<string>.Broadcast(PostClientEvents.SHOW\_DIALOG\_WINDOW, "Подождите...\n\nИдёт обработка письма.");

IsWorking = false;

string[] mailSplitString = \_stringsReadedInLetter.Split('\n');

List<string[]> parsedMail = new List<string[]>();

yield return StartCoroutine(ReadMail(mailSplitString, parsedMail));

}

}

}

private IEnumerator ReadSocketList()

{

\_stringsReadedInLetter = "";

int numberOfReading = 50000;

do

{

ReceiveFromSocket();

\_stringsReadedInLetter += GetStringFromSocket();

numberOfReading--;

if (numberOfReading % 1000 == 0)

Debug.Log(numberOfReading);

if (numberOfReading % \_numberOfStringsToReadPerOneFrame == 0)

yield return 0;

} while (!\_stringsReadedInLetter.Contains("\n.\r\n") && (numberOfReading>0));

}

public void RefreshMailList()

{

StopCoroutine(updateConnection);

IsLogged = false;

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("QUIT\n"));

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

Connect();

string answer;

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("USER " + \_lastLogin + "\n"));

ReceiveFromSocket();

answer = GetStringFromSocket();

Debug.Log(answer);

if (!answer[0].Equals('+'))

{

Messenger.Broadcast(PostClientEvents.CONNECTION\_PROBLEMS);

Messenger<string>.Broadcast(PostClientEvents.UPDATE\_STATUS, "Повторите данные для входа");

return;

}

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("PASS " + \_lastPassword + "\n"));

ReceiveFromSocket();

answer = GetStringFromSocket();

Debug.Log(answer);

if (!answer[0].Equals('+'))

{

Messenger.Broadcast(PostClientEvents.CONNECTION\_PROBLEMS);

Messenger<string>.Broadcast(PostClientEvents.UPDATE\_STATUS, "Повторите данные для входа");

return;

}

IsLogged = true;

StartCoroutine(updateConnection);

StartCoroutine(FormMailListCoroutine());

}

public void DeleteMail(int number)

{

if (IsLogged)

{

IsWorking = true;

GetNumberOfLetters();

IsWorking = false;

if (number <= numberOfLettersInMailbox && number >= 1)

{

IsWorking = true;

Messenger<string>.Broadcast(PostClientEvents.SHOW\_DIALOG\_WINDOW, "Подождите...\n\nИдёт удаление письма.");

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("DELE " + number + "\n"));

ReceiveFromSocket();

IsWorking = false;

RefreshMailList();

}

}

}

public void CloseSocket()

{

if (IsConnected)

{

if (IsLogged)

{

IsWorking = true;

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("QUIT\n"));

ReceiveFromSocket();

IsWorking = false;

IsLogged = false;

}

\_popStream.Close();

\_popClient.Close();

}

}

private void ReceiveFromSocket()

{

Array.Clear(\_buff, 0, \_buff.Length);

\_numberOfBytesInLastReading = \_popStream.Read(\_buff, 0, \_buff.Length);

}

private string GetStringFromSocket()

{

return System.Text.Encoding.UTF8.GetString(\_buff, 0, \_numberOfBytesInLastReading);

}

private string NormalizeString (string oldString)

{

Regex contentRegex = new Regex(@"=\?(?<convertTo>\S\*)\?(?<convertFrom>\S\*)\?(?<content>\S\*)\?=");

Match contentMatch = contentRegex.Match(oldString);

if (contentMatch.Success)

{

string convertFrom = contentMatch.Groups["convertFrom"].Value;

string convertTo = contentMatch.Groups["convertTo"].Value;

string content = contentMatch.Groups["content"].Value;

if (convertFrom.Equals("Q") || convertFrom.Equals("q") || convertFrom.Equals("quoted-printable"))

{

if (content[0].Equals('\_'))

content = content.Substring(1);

content = string.Join(" ", content.Split('\_'));

}

return content.Contains("\n") ? DecodeString(content.Split('\n'), convertFrom, convertTo) : DecodeString(content, convertFrom, convertTo);

}

return oldString;

}

private string DecodeString(string codedString, string convertFrom, string convertTo)

{

byte[] encodedBytes;

string senderedSubject;

switch (convertFrom)

{

case "Q":

case "q":

case "quoted-printable":

encodedBytes = DecodeQuotedPrintable(codedString);

break;

case "B":

case "b":

case "base64":

encodedBytes = Convert.FromBase64String(codedString);

break;

default:

encodedBytes = System.Text.Encoding.UTF8.GetBytes(codedString);

break;

}

switch (convertTo)

{

case "UTF-8":

case "utf-8":

senderedSubject = System.Text.Encoding.UTF8.GetString(encodedBytes);

break;

case "KOI8-R":

case "koi8-r":

senderedSubject = System.Text.Encoding.GetEncoding("koi8-r").GetString(encodedBytes);

break;

default:

senderedSubject = System.Text.Encoding.UTF8.GetString(encodedBytes);

break;

}

return senderedSubject;

}

private string DecodeString(string[] codedString, string convertFrom, string convertTo)

{

string senderedSubject = "";

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

senderedSubject += DecodeString(codedString[i], convertFrom, convertTo) + '\n';

return senderedSubject;

}

private byte[] DecodeQuotedPrintable(string stringToDecode)

{

List<byte> byteList = new List<byte>();

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

{

char symbol = stringToDecode[i];

switch (symbol)

{

case '=':

if (i != stringToDecode.Length - 1)

{

byte nextByte = Convert.ToByte(stringToDecode.Substring(i + 1, 2), 16);

byteList.Add(nextByte);

i += 2;

}

break;

default:

byteList.Add((byte)symbol);

break;

}

}

return byteList.ToArray();

}

private string FindDate(string mailString)

{

Regex dateRegex = new Regex("Date: .\*");

Match dateMatch = dateRegex.Match(mailString);

if (dateMatch.Success)

{

string date = dateMatch.Value;

string sendDate;

if (date.Contains("+"))

sendDate = date.Substring(date.IndexOf(',') + 2, date.IndexOf('+') - (date.IndexOf(',') + 3));

else sendDate = date.Substring(date.IndexOf(',') + 2, date.IndexOf('-') - (date.IndexOf(',') + 3));

return sendDate;

}

else return "";

}

private IEnumerator ReadMail(string[] mailContent, List<string[]> parseContent)

{

parseContent = new List<string[]>();

int splitLength = mailContent.Length;

for (\_readMailIterator = 0; \_readMailIterator < splitLength; \_readMailIterator++)

{

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

if (\_readMailIterator % \_numberOfStringsToReadPerOneFrame == 0)

yield return 0;

if (mailContent[\_readMailIterator].IndexOf("Subject: ") == 0)

{

string subject = mailContent[\_readMailIterator].Substring(9);

for (int i = \_readMailIterator; i <splitLength - 1; i++)

{

if (!mailContent[i + 1].Contains(":"))

{

mailContent[i + 1] = string.Join("", mailContent[i + 1].Split(new char[] { '\0', '\r' }));

subject += mailContent[i + 1];

}

else break;

}

subject = Regex.Replace(subject, @"=\?\S\*\?=", (Match match) =>

{

return NormalizeString(match.Value);

});

parseContent.Add(new string[2] { AppConstants.subjectOfMessage, subject });

}

else if (mailContent[\_readMailIterator].Contains("Content-Type: multipart"))

{

string multipartBounder = mailContent[\_readMailIterator].Contains("\"") ? new Regex("boundary=\"(?<bounder>.\*)\"").Match(mailContent[\_readMailIterator]).Groups["bounder"].Value : new Regex ("boundary=(?<bounder>.\*)").Match(mailContent[\_readMailIterator]).Groups["bounder"].Value;

while (multipartBounder.Equals(""))

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

multipartBounder = mailContent[\_readMailIterator].Contains("\"") ? new Regex("boundary=\"(?<bounder>.\*)\"").Match(mailContent[\_readMailIterator]).Groups["bounder"].Value : new Regex ("boundary=(?<bounder>.\*)").Match(mailContent[\_readMailIterator]).Groups["bounder"].Value;

}

do

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

} while (!mailContent[\_readMailIterator].Equals("--" + multipartBounder));

yield return StartCoroutine(ReadMailMultipart(mailContent, splitLength, multipartBounder, parseContent));

}

else if (mailContent[\_readMailIterator].Contains("Content-Type: text/plain"))

{

string codingTo = new Regex("charset=\"?(?<codingTo>\\S\*)\"?"). Match(mailContent[\_readMailIterator]).Groups["codingTo"].Value;

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

string codingFrom = new Regex(@"Content-Transfer-Encoding: (?<codingFrom>\S\*)").Match(mailContent[\_readMailIterator]).Groups["codingFrom"].Value;

while (codingFrom.Equals(""))

{

if (mailContent[\_readMailIterator].Equals(""))

while (codingFrom.Equals(""))

{

\_readMailIterator--;

codingFrom = new Regex(@"Content-Transfer-Encoding: (?<codingFrom>\S\*)").Match(mailContent[\_readMailIterator]).Groups["codingFrom"].Value;

}

else

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

codingFrom = new Regex(@"Content-Transfer-Encoding: (?<codingFrom>\S\*)").Match(mailContent[\_readMailIterator]).Groups["codingFrom"].Value;

}

}

do

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

}

while (!mailContent[\_readMailIterator].Equals(string.Empty));

\_readMailIterator++;

string[] textContent = new string[2] { AppConstants.textPartOfMessage, "" };

switch (codingFrom)

{

case "Q":

case "q":

case "quoted-printable":

default:

while (\_readMailIterator < splitLength)

{

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

if (mailContent[\_readMailIterator].Equals("."))

break;

else textContent[1] = (mailContent[\_readMailIterator].Length != 0 && mailContent[\_readMailIterator][mailContent[\_readMailIterator].Length - 1].Equals('=')) ? string.Concat(textContent[1], DecodeString(mailContent[\_readMailIterator], codingFrom, codingTo)) : string.Concat(textContent[1], DecodeString(mailContent[\_readMailIterator], codingFrom, codingTo), "\n");

\_readMailIterator++;

if (\_readMailIterator % \_numberOfStringsToReadPerOneFrame == 0)

yield return 0;

}

break;

case "B":

case "b":

case "base64":

string base64content = "";

while (\_readMailIterator < splitLength)

{

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

if (mailContent[\_readMailIterator].Equals("."))

break;

else base64content += mailContent[\_readMailIterator];

\_readMailIterator++;

if (\_readMailIterator % \_numberOfStringsToReadPerOneFrame == 0)

yield return 0;

}

textContent[1] = DecodeString(base64content, codingFrom, codingTo);

break;

}

textContent[1] = textContent[1].Trim('\n');

parseContent.Add(textContent);

}

else if (mailContent[\_readMailIterator].Contains("Content-Disposition: attachment"))

{

string attachName = new Regex("filename=\"(?<filename>.\*)\"").Match(mailContent[\_readMailIterator]).Groups["filename"].Value;

while (attachName.Equals(""))

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

attachName = new Regex("filename=\"(?<filename>.\*)\"").Match(mailContent[\_readMailIterator]).Groups["filename"].Value;

}

attachName = NormalizeString(attachName);

string[] attachContent = new string[2] { attachName , "" };

do

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

}

while (!mailContent[\_readMailIterator].Equals(string.Empty));

\_readMailIterator++;

while (\_readMailIterator < splitLength)

{

attachContent[1] += string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' , ' '}));

\_readMailIterator++;

if (\_readMailIterator % \_numberOfStringsToReadPerOneFrame == 0)

yield return 0;

}

parseContent.Add(attachContent);

}

}

Messenger<List<string[]>>.Broadcast(PostClientEvents.SHOW\_LETTER, parseContent);

}

private IEnumerator ReadMailMultipart(string[] mailContent, string bound, List<string[]> parseContent)

{

int contentLength = mailContent.Length;

yield return StartCoroutine(ReadMailMultipart(mailContent, contentLength, bound, parseContent));

}

private IEnumerator ReadMailMultipart(string[] mailContent, int contentLength, string bound, List<string[]> parseContent)

{

for (\_readMailIterator++; \_readMailIterator < contentLength; \_readMailIterator++)

{

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

if (\_readMailIterator % \_numberOfStringsToReadPerOneFrame == 0)

yield return 0;

if (mailContent[\_readMailIterator].Contains("Content-Type: multipart"))

{

string multipartBounder = mailContent[\_readMailIterator].Contains("\"") ? new Regex("boundary=\"(?<bounder>.\*)\"").Match(mailContent[\_readMailIterator]).Groups["bounder"].Value : new Regex("boundary=(?<bounder>.\*)"). Match(mailContent[\_readMailIterator]).Groups["bounder"].Value;

while (multipartBounder.Equals(""))

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

multipartBounder = mailContent[\_readMailIterator].Contains("\"") ? new Regex("boundary=\"(?<bounder>.\*)\"").Match(mailContent[\_readMailIterator]).Groups["bounder"].Value : new Regex("boundary=(?<bounder>.\*)"). Match(mailContent[\_readMailIterator]).Groups["bounder"].Value;

}

do

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

} while (!mailContent[\_readMailIterator].Equals("--" + multipartBounder));

yield return StartCoroutine(ReadMailMultipart(mailContent, contentLength, multipartBounder, parseContent));

}

else if (mailContent[\_readMailIterator].Contains("Content-Type: text/plain"))

{

string codingTo = new Regex("charset=\"?(?<codingTo>\\S\*)\"?").Match(mailContent[\_readMailIterator]).Groups["codingTc"].Value;

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

string codingFrom = new Regex(@"Content-Transfer-Encoding: (?<codingFrom>\S\*)").Match(mailContent[\_readMailIterator]).Groups["codingFrom"].Value;

while (codingFrom.Equals(""))

{

if (mailContent[\_readMailIterator].Equals(""))

while (codingFrom.Equals(""))

{

\_readMailIterator--;

codingFrom = new Regex(@"Content-Transfer-Encoding: (?<codingFrom>\S\*)").Match(mailContent[\_readMailIterator]).Groups["codingFrom"].Value;

}

else

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

codingFrom = new Regex(@"Content-Transfer-Encoding: (?<codingFrom>\S\*)").Match(mailContent[\_readMailIterator]).Groups["codingFrom"].Value;

}

}

do

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

}

while (!mailContent[\_readMailIterator].Equals(string.Empty));

\_readMailIterator++;

string[] textContent = new string[2] { AppConstants.textPartOfMessage, "" };

switch (codingFrom)

{

case "Q": case "q":

case "quoted-printable":

default:

while (\_readMailIterator < contentLength)

{

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

if (mailContent[\_readMailIterator].Contains("--" + bound))

break;

else textContent[1] = (mailContent[\_readMailIterator].Length != 0 && mailContent[\_readMailIterator][mailContent[\_readMailIterator].Length - 1].Equals('=')) ? string.Concat(textContent[1], DecodeString(mailContent[\_readMailIterator], codingFrom, codingTo)) : string.Concat(textContent[1], DecodeString(mailContent[\_readMailIterator], codingFrom, codingTo), "\n");

\_readMailIterator++;

if (\_readMailIterator % \_numberOfStringsToReadPerOneFrame == 0)

yield return 0;

}

break;

case "B": case "b":

case "base64":

string base64content = "";

while (\_readMailIterator < contentLength)

{

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

if (mailContent[\_readMailIterator].Contains("--" + bound))

break;

else base64content += mailContent[\_readMailIterator];

\_readMailIterator++;

if (\_readMailIterator % \_numberOfStringsToReadPerOneFrame == 0)

yield return 0;

}

textContent[1] = DecodeString(base64content, codingFrom, codingTo);

break;

}

textContent[1] = textContent[1].Trim('\n');

parseContent.Add(textContent);

}

else if (mailContent[\_readMailIterator].Contains("Content-Disposition: attachment"))

{

string attachName = new Regex("filename=\"(?<filename>.\*)\"").Match(mailContent[\_readMailIterator]).Groups["filename"].Value;

while (attachName.Equals(""))

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

attachName = new Regex("filename=\"(?<filename>.\*)\"").Match(mailContent[\_readMailIterator]).Groups["filename"].Value;

}

attachName = NormalizeString(attachName);

string[] attachContent = new string[2] { attachName, "" };

do

{

\_readMailIterator++;

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

}

while (!mailContent[\_readMailIterator].Equals(string.Empty));

\_readMailIterator++;

while (\_readMailIterator < contentLength)

{

mailContent[\_readMailIterator] = string.Join("", mailContent[\_readMailIterator].Split(new char[] { '\0', '\r' }));

if (mailContent[\_readMailIterator].Contains("--" + bound))

break;

attachContent[1] += string.Join("", mailContent[\_readMailIterator].Split(' '));

\_readMailIterator++;

if (\_readMailIterator % \_numberOfStringsToReadPerOneFrame == 0)

yield return 0;

}

parseContent.Add(attachContent);

}

if (mailContent[\_readMailIterator].Equals("--" + bound + "--"))

{

break;

}

}

}

public IEnumerator UpdateConnection()

{

while (IsLogged)

{

if (!IsWorking)

{

if (IsConnected)

{

\_popStream.Write(System.Text.Encoding.UTF8.GetBytes("NOOP\n"));

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

}

else Messenger.Broadcast(PostClientEvents.CONNECTION\_PROBLEMS);

}

yield return new WaitForSeconds(5);

}

}

}

SMTPSocket

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System;

using System.Net.Sockets;

using System.Net.Security;

using System.IO;

using System.Text.RegularExpressions;

public class SMTPSocket : MonoBehaviour {

private TcpClient \_smtpClient;

private SslStream \_smtpStream;

private byte[] \_buff = new byte[1000];

private string \_stringBuff;

private int \_numberOfBytesInLastReading;

private string \_lastLogin;

private string \_lastPassword;

public bool IsConnected { get { return \_smtpClient == null ? false : \_smtpClient.Connected; } }

public bool IsLogged { get; private set; }

public bool IsWorking { get; private set; }

private void Awake()

{

\_numberOfBytesInLastReading = 0;

}

private void OnDestroy()

{

CloseSocket();

}

void Start()

{

IsLogged = false;

IsWorking = false;

StartCoroutine(StartConnection());

}

public IEnumerator StartConnection()

{

Messenger<string>.Broadcast(PostClientEvents.UPDATE\_STATUS, "Выполняется подключение");

yield return new WaitForSeconds(.5f);

Connect();

CloseSocket();

}

public void Connect()

{

try

{

if (\_smtpClient != null)

{

\_smtpStream.Close();

\_smtpClient.Close();

}

\_smtpClient = new TcpClient(AppConstants.smtpServerAdress, 465);

\_smtpStream = new SslStream(\_smtpClient.GetStream());

\_smtpStream.AuthenticateAsClient(AppConstants.smtpServerAdress);

Debug.Log(\_smtpStream.IsAuthenticated + " " + \_smtpStream.IsSigned);

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

Debug.Log("SMTP CONNECTED");

Messenger.Broadcast(PostClientEvents.CLEAR\_STATUS);

}

catch

{

Messenger<bool>.Broadcast(PostClientEvents.CONNECTION\_PROBLEMS\_WITH\_TURNING\_MESSAGE, false);

Messenger<string>.Broadcast(PostClientEvents.UPDATE\_STATUS, "Попытка подключения SMTP сокета закончилась неудачей");

}

}

public bool StartSMTPSession(string login, string password)

{

\_lastLogin = login;

\_lastPassword = password;

if (IsConnected)

{

string answer;

WriteToSocket("HELO world\n");

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

WriteToSocket("AUTH LOGIN\n");

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

byte[] loginBytes = System.Text.Encoding.UTF8.GetBytes(login);

login = Convert.ToBase64String(loginBytes);

WriteToSocket(login + "\n");

ReceiveFromSocket();

answer = GetStringFromSocket();

Debug.Log(answer);

if (!answer[0].Equals('3'))

return IsLogged;

byte[] passBytes = System.Text.Encoding.UTF8.GetBytes(password);

password = Convert.ToBase64String(passBytes);

WriteToSocket(password + "\n");

ReceiveFromSocket();

answer = GetStringFromSocket();

Debug.Log(answer);

if (!answer[0].Equals('2'))

return IsLogged;

IsLogged = true;

return IsLogged;

}

else

{

return false;

}

}

public bool CheckConnection(string login, string password)

{

Connect();

bool connection = StartSMTPSession(login, password);

CloseSocket();

return connection;

}

public void WriteALetter(string sendTo, string letter)

{

Messenger<string>.Broadcast(PostClientEvents.SHOW\_DIALOG\_WINDOW, "Отправка письма.");

StartCoroutine(WriteALetterCoroutine(sendTo, letter));

}

public IEnumerator WriteALetterCoroutine(string sendTo, string letter)

{

Connect();

if (StartSMTPSession(\_lastLogin, \_lastPassword) == false)

{

CloseSocket();

Messenger<bool>.Broadcast(PostClientEvents.CONNECTION\_PROBLEMS\_WITH\_TURNING\_MESSAGE, false);

Messenger<string>.Broadcast(PostClientEvents.UPDATE\_STATUS, "Сбой соединения SMTP сокета");

}

else

{

WriteToSocket("MAIL FROM: " + \_lastLogin + "\n");

Debug.Log("MAIL FROM: " + \_lastLogin + "\n");

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

WriteToSocket("RCPT TO: " + sendTo + "\n");

Debug.Log("RCPT TO: " + sendTo + "\n");

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

WriteToSocket("DATA\n");

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

int symbolsInOnePacket = 1000;

int numberOfStrings = letter.Length / symbolsInOnePacket;

Debug.Log(numberOfStrings + " " + letter.Length);

for (int i = 0; i < numberOfStrings; i++)

{

WriteToSocket(letter.Substring(i \* symbolsInOnePacket, symbolsInOnePacket));

if (i % 5 == 0) yield return 0;

}

int lastSymbols = letter.Length % symbolsInOnePacket;

if (lastSymbols != 0)

WriteToSocket(letter.Substring(numberOfStrings \* symbolsInOnePacket, lastSymbols));

Debug.Log("Waiting for receive");

yield return 0;

ReceiveFromSocket();

Debug.Log(GetStringFromSocket());

CloseSocket();

Messenger.Broadcast(PostClientEvents.SMTP\_CLEAR\_INPUTS);

Messenger.Broadcast(PostClientEvents.CLOSE\_DIALOG\_WINDOW);

}

}

private void WriteToSocket(string stringToSend)

{

WriteToSocket(System.Text.Encoding.UTF8.GetBytes(stringToSend));

}

private void WriteToSocket(byte[] bytesToSend)

{

\_smtpStream.Write(bytesToSend, 0, bytesToSend.Length);

}

private void ReceiveFromSocket()

{

Array.Clear(\_buff, 0, \_buff.Length);

\_numberOfBytesInLastReading = \_smtpStream.Read(\_buff, 0, \_buff.Length);

}

private string GetStringFromSocket()

{

return System.Text.Encoding.UTF8.GetString(\_buff, 0, \_numberOfBytesInLastReading);

}

public void CloseSocket()

{

if (IsConnected)

{

if (IsLogged)

{

IsWorking = true;

WriteToSocket("RSET\n");

ReceiveFromSocket();

WriteToSocket("QUIT\n");

ReceiveFromSocket();

IsWorking = false;

IsLogged = false;

}

\_smtpStream.Close();

\_smtpClient.Close();

}

}

}

ShortContent

using UnityEngine;

using UnityEngine.UI;

public class ShortContent : MonoBehaviour {

[SerializeField] private Image \_background;

[SerializeField] private Color32 \_color;

[SerializeField] private Color32 \_activeColor;

[SerializeField] private Color32 \_clickColor;

[SerializeField] private Text nameText;

[SerializeField] private Text mailText;

[SerializeField] private Text themeText;

[SerializeField] private Text dateText;

public int letterNumber = 0;

private void Start () {

BoxCollider2D collider = GetComponent<BoxCollider2D>();

collider.size = new Vector2(475, 75);

}

public void PointerDown()

{

\_background.color = \_clickColor;

}

public void PointerEnter()

{

\_background.color = \_activeColor;

}

public void PointerExit()

{

\_background.color = \_color;

}

public void SetColor(Color32 newColor)

{

\_background.color = newColor;

\_color = newColor;

\_activeColor.r = newColor.r < 245 ? (byte)(newColor.r + 10) : (byte)255;

\_activeColor.g = newColor.g < 245 ? (byte)(newColor.g + 10) : (byte)255;

\_activeColor.b = newColor.b < 245 ? (byte)(newColor.b + 10) : (byte)255;

\_activeColor.a = 255;

\_clickColor.r = newColor.r < 235 ? (byte)(newColor.r + 20) : (byte)255;

\_clickColor.g = newColor.g < 235 ? (byte)(newColor.g + 20) : (byte)255;

\_clickColor.b = newColor.b < 235 ? (byte)(newColor.b + 20) : (byte)255;

\_clickColor.a = 255;

}

public void SetText(string senderName, string senderMail, string sendDate, string senderedSubject)

{

nameText.text = senderName;

mailText.text = senderMail;

themeText.text = sendDate;

dateText.text = senderedSubject;

}

public void FormLetter()

{

Messenger<int>.Broadcast(PostClientEvents.FORM\_LETTER, letterNumber);

}

public void DeleteMail()

{

Messenger<int>.Broadcast(PostClientEvents.DELETE\_LETTER, letterNumber);

}

}

MailShortInfo

public struct MailShortInfo {

public string senderName;

public string senderMail;

public string senderedSubject;

public string sendDate;

}

AttachmentGet

using System.IO;

using UnityEngine;

using UnityEngine.UI;

using UnityEditor;

public class AttachmentGet : MonoBehaviour {

[SerializeField] private Sprite nonActiveSprite;

[SerializeField] private Sprite activeSprite;

[SerializeField] private Sprite clickedSprite;

[SerializeField] private Text attachmentName;

private Image \_attachmentImage;

private string \_fullName;

private string \_base64Content;

void Start()

{

\_attachmentImage = GetComponent<Image>();

}

public void PointerDown()

{

\_attachmentImage.sprite = clickedSprite;

}

public void PointerEnter()

{

\_attachmentImage.sprite = activeSprite;

}

public void PointerExit()

{

\_attachmentImage.sprite = nonActiveSprite;

}

public void SetContent(string[] content)

{

if (content.Length == 2)

{

if (content[0].Length > 12)

attachmentName.text = string.Concat(content[0].Substring(0, 9), "...");

else attachmentName.text = content[0];

\_fullName = content[0];

\_base64Content = content[1];

}

else attachmentName.text = "I AM ERROR";

}

public void DownloadAttachment()

{

string path = EditorUtility.SaveFilePanel("Сохранить файл", "", \_fullName, \_fullName.Substring(\_fullName.LastIndexOf(".") + 1, \_fullName.Length - \_fullName.LastIndexOf(".") - 1));

if (path.Length != 0)

{

File.WriteAllBytes(path, System.Convert.FromBase64String(\_base64Content));

}

}

}

AttachmentSend

using UnityEngine;

using UnityEngine.UI;

public class AttachmentSend : MonoBehaviour {

public Text attachName;

public Text attachSize;

[SerializeField] private GameObject attachmentObject;

private byte[] attachmentContent;

public void SetContent (string name, int size, byte[] content)

{

attachName.text = name;

attachSize.text = size.ToString();

attachmentContent = content;

}

public void DeleteAttach ()

{

Messenger<GameObject>.Broadcast(PostClientEvents.DELETE\_ATTACHMENT\_TO\_SEND, attachmentObject);

}

public string GetBase64Content()

{

return System.Convert.ToBase64String(attachmentContent);

}

public string GetBase64Name()

{

byte[] name = System.Text.Encoding.UTF8.GetBytes(attachName.text);

return System.Convert.ToBase64String(name);

}

}

AppConstants

public static class AppConstants

{

public const string senderName = "SenderName";

public const string senderMail = "SenderMail";

public const string senderedSubject = "SenderedSubject";

public const string sendDate = "SendDate";

public const int nameIndexInShortInfo = 0;

public const int themeIndexInShortInfo = 1;

public const int mailIndexInShortInfo = 2;

public const int dateIndexInShortInfo = 3;

public const string textPartOfMessage = "textPartOfMessage";

public const string subjectOfMessage = "subjectOfMessage";

public const string pop3ServerAdress = "mail.ngs.ru";

public const string smtpServerAdress = "mail.ngs.ru";

}

PostClientEvents

public static class PostClientEvents {

public const string START\_POP3\_SESSION = "START\_POP3\_SESSION";

public const string UPDATE\_STATUS = "UPDATE\_STATUS";

public const string CLEAR\_STATUS = "CLEAR\_STATUS";

public const string CONNECTION\_PROBLEMS = "CONNECTION\_PROBLEMS";

public const string CONNECTION\_PROBLEMS\_WITH\_TURNING\_MESSAGE = "CONNECTION\_PROBLEMS\_WITH\_TURNING\_MESSAGE";

public const string FORM\_LETTER = "FORM\_LETTER";

public const string FORM\_LIST\_OF\_MAILS = "FORM\_LIST\_OF\_MAILS";

public const string SHOW\_LETTER = "SHOW\_LETTER";

public const string DELETE\_LETTER = "DELETE\_LETTER";

public const string SEND\_LETTER = "SEND\_LETTER";

public const string SMTP\_CLEAR\_INPUTS = "SMTP\_CLEAR\_INPUTS";

public const string UPDATE\_SENDER\_MAIL = "UPDATE\_SENDER\_MAIL";

public const string DELETE\_ATTACHMENT\_TO\_SEND = "DELETE\_ATTACHMENT\_TO\_SEND";

public const string OPEN\_SENDMAIL\_PANEL = "OPEN\_SENDMAIL\_PANEL";

public const string CLOSE\_SENDMAIL\_PANEL = "CLOSE\_SENDMAIL\_PANEL";

public const string SHOW\_DIALOG\_WINDOW = "SHOW\_DIALOG\_WINDOW";

public const string CLOSE\_DIALOG\_WINDOW = "CLOSE\_DIALOG\_WINDOW";

public const string SET\_DIALOG\_WINDOW\_TEXT = "SET\_DIALOG\_WINDOW\_TEXT";

}