СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc152416345)

[1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ 5](#_Toc152416346)

[2 ПРОЕКТИРОВАНИЕ ПРОГРАММЫ 6](#_Toc152416347)

[3 РЕАЛИЗАЦИЯ ПРОГРАММЫ 8](#_Toc152416348)

[3.1 Детальная реализация функциональных частей ПО 8](#_Toc152416349)

[3.2 Сопроводительная документация 23](#_Toc152416350)

[3.3 Анализ ПО 23](#_Toc152416351)

[4 МЕТОДИКА И РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 24](#_Toc152416348)

[ЗАКЛЮЧЕНИЕ 28](#_Toc152416353)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc152416354)

[ПРИЛОЖЕНИЕ А (обязательное) Техническое задание 30](#_Toc152416355)

[ПРИЛОЖЕНИЕ Б (обязательное) Диаграмма вариантов использования 34](#_Toc152416356)

[ПРИЛОЖЕНИЕ В (обязательное) Таблица тестирования программы 35](#_Toc152416357)

# Введение

Данный курсовой проект предусматривает написание клиент-серверного приложения «Мессенджер».

Данная тема представляет собой актуальную проблему в современном обществе, поскольку часто возникает необходимость передать информацию кому-то, кто находится в отдалении. Для обмена сообщениями в реальном времени в таких ситуациях многие прибегают к использованию мессенджеров. В данной курсовой работе будет разработано приложение с клиент-серверной архитектурой, предназначенное для передачи сообщений между пользователями.

Для решения поставленной задачи, разработка программы для Windows 10 будет происходить в среде Microsoft Visual Studio 2022 с использованием .NET 8 и C#. Для хранения данных будет использоваться Firebase Realtime Datebase. Для пересылки сообщения в реальном времени будет использоваться ASP.NET Signalr. Разработка диаграммы вариантов использования будет происходить в Enterprise Architect.

Firebase - это облачная база данных, которая позволяет пользователям хранить и получать сохраненную информацию, а также имеет удобные средства и методы взаимодействия с ней. [Firebase](http://firebase.google.com/) хранит текстовые данные в [JSON](https://ru.wikipedia.org/wiki/JSON) формате и предоставляет удобные методы для чтения, обновления и извлечения данных. Также, Firebase может помочь с регистрацией и авторизацией пользователей, хранением сессий (авторизованные пользователи), медиафайлов к которым с легкостью предоставляет доступ благодаря Cloud Storage[1].

Firebase realtime datebase - Храните и синхронизируйте данные с нашей облачной базой данных NoSQL. Данные синхронизируются между всеми клиентами в режиме реального времени и остаются доступными, когда ваше приложение отключается от сети[2].

ASP.NET SignalR — это библиотека для разработчиков ASP.NET, которая упрощает процесс добавления веб-функций в режиме реального времени в приложения. Веб-функции в режиме реального времени — это возможность мгновенно отправлять содержимое кода сервера на подключенные клиенты по мере его доступности, а не ждать, пока клиент запросит новые данные[3].

Enterprise Architect (EA) – CASE-инструмент для проектирования и конструирования программного обеспечения. EA поддерживает спецификацию UML2.0+, описывающую визуальный язык, которым могут быть определены модели проекта[4].

# 1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ

«Клиент - сервер» вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически [клиент](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) и [сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) - это [программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Обычно программы расположены на разных [вычислительных машинах](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) и взаимодействуют между собой через [вычислительную сеть](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C) посредством [сетевых протоколов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB), но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, [передача файлов](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B0_%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2&action=edit&redlink=1) посредством [HTTP](https://ru.wikipedia.org/wiki/HTTP), [FTP](https://ru.wikipedia.org/wiki/FTP), [BitTorrent](https://ru.wikipedia.org/wiki/BitTorrent_(%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB)" \o "BitTorrent (протокол)), [потоковое мультимедиа](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BC%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0) или работа с [базами данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) или в виде сервисных функций (например, работа с [электронной почтой](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D1%87%D1%82%D0%B0), общение посредством [систем мгновенного обмена сообщениями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BC%D0%B3%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%BC%D0%B5%D0%BD%D0%B0_%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F%D0%BC%D0%B8) или просмотр [web-страниц](https://ru.wikipedia.org/wiki/Web-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0" \o "Web-страница) во [всемирной паутине](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0)). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют [сервером](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%B0%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)), а машины, выполняющие клиентские программы, соответственно, клиентами[5].

На данный момент существует огромное количество мессенджеров для обмена сообщениями и их количество растет в связи с безопасностью передачи сообщений. Примерами мессенджеров, которые используются в настоящее время можно считать WhatsApp, Viber, Telegram. Данные мессенджеры разработаны для общего пользования и предоставляют возможность обмену сообщениями большому количеству пользователей, в то время как «Мессенджер» предназначается для частного использования в рамках курсового проекта.

# 2 ПРОЕКТИРОВАНИЕ ПРОГРАММЫ

Проектирование программного обеспечения – это комплекс мер, направленный на определение внутренних свойств приложения, а также детализацию видимых компонентов. Создается специальный проект, который позволяет разработчикам четко понимать план действий, определить стек технологий и разработать методы и стратегии создания продукта[6].

Значимую роль в проектировании играет выбор подходящей технологии. Это включает выбор языка программирования (фреймворка, базы данных и другие инструменты), которые будут применяться в процессе работы. Этот выбор влияет на производительность и надежность будущего приложения, а также его масштабируемости.

Проектирование включает определение архитектурных особенностей приложения и стратегий его решения. Это подразумевает выбор между разными стилями архитектуры, шаблонами и способами сортировки данных. Этот выбор также влияет на развитие приложения и его масштабирование в дальнейшем.

Кроме того, в процессе проектирования создается техническое руководство, которое является важным руководством разработчиков по разработке и поддержке программного обеспечения.

Проектирование программного обеспечения делается для следующих целей:

* Анализ требований: понимание и анализ потребностей пользователей или заказчика для определения функций и особенностей программного продукта;
* Проектирование архитектуры: определение общей структуры программы, выбор платформы, определение компонентов системы, их взаимодействие и организацию для достижения поставленных целей;
* Детальное проектирование: разработка более подробных спецификаций компонентов системы, определение алгоритмов, классов, интерфейсов и других технических деталей, необходимых для реализации программного продукта;
* Тестирование и оценка проектирования: проведение проверок и тестирования предварительного дизайна программы с целью обнаружения ошибок и улучшения структуры перед фактической разработкой;

В конечном итоге, проектирование программного обеспечения служит для обеспечения успешного выполнения проекта, минимизации рисков и ошибок, упрощения процесса разработки и обеспечения высокого качества конечного продукта. Проектирование программного продукта нужно для того, чтобы создать четкое техническое задание с распределением обязанностей, увидеть возможности и функционал программы до начала разработки и понять алгоритм действий. В случае отсутствия проекта высока вероятность не получить готовое решение.

После инициализации программы, пользователи встречаются формой входа, где они могут аутентификацию войти в аккаунтили перейти на другую форму для создания аккаунта. После авторизации, появиться форма переписки между пользователями.

В процессе использования приложения, пользователь имеет возможность отправлять и получать сообщения в режиме реального времени как отдельно выбранному пользователю, так и в рамках выбранной группы. Этот функционал обеспечивает эффективное общение и взаимодействие пользователей в приложении. Таким образом, пользователи могут оперативно обмениваться информацией и поддерживать контакт друг с другом.

По завершении работы приложения все сообщения будут сохранены. При закрытии чата приложение полностью завершает свое выполнение. А при закрытии формы регистрации возвращаемся в форму аутентификации и входа в аккаунт. После регистрации производится переход на форму аутентификации и входа в аккаунт

Для выполнения поставленной задачи должны быть спроектированы следующие основные классы:

* ChatHub – это класс, который служит в качестве конвейера высокого уровня для обработки взаимодействия между клиентом и сервером;
* User – отвечает для хранение информации о пользователе;
* ChatMessages – хранит сообщения между пользователями;
* ChatGroup – хранит информацию о участниках группы и сообщений в них;
* Chat – отвечает за отображении формы переписки;
* Registration – отвечает за отображение формы регистрации аккаунта;
* SignIn – отвечает за отображение формы аутентификации и входа в аккаунт;

# 3 РЕАЛИЗАЦИЯ ПРОГРАММЫ

# 3.1 Детальная реализация функциональных частей ПО

Класс ChatHub - это основной класс для отправки сообщений клиентам, подключенным к серверу. Он содержит методы для отправки уведомления всем о состоянии пользователя, для отправки сообщений в общую группу и выбранному пользователю. Так же имеет метод для получения идентификатора соединения. Класс ChatHub представлен в листинге 3.1.

Листинг 3.1 – Класс ChatHub

public class ChatHub : Hub

{

public async Task Send(int userId, string message)

{

await Clients.All.SendAsync("Receive", userId, message);

}

public async Task SendToGroup(int userId, string message, int groupId = 0)

{

await Clients.All.SendAsync("ReceiveToGroup", userId, message, groupId);

}

public async Task SendToUser(int userId, string receiverConnectionId, string message)

{

await Clients.Client(receiverConnectionId).SendAsync("ReceiveToUser", userId, message, receiverConnectionId);

}

public string GetConnectionId() => Context.ConnectionId;

}

Класс User – хранит атрибуты для определения пользователя. У каждого пользователя свой личный id. Класс User представлен в листинге 3.2.

Листинг 3.2 – Класс User

public class User

{

public int Id { get; set; }

public string Name { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public bool IsOnline { get; set; }

public DateTime DateTime { get; set; }

public override bool Equals(object? obj)

Продолжение листинга 3.2

{

if(obj is User user)

{

if (user.Id == Id)

return true;

return false;

}

return false;

}

}

Класс ChatMessages – хранит Id обоих пользователей и сообщения между ними. Данный класс представлен в листинге 3.3.

Листинг 3.3 – Класс ChatMessages

public class ChatMessages

{

public ChatMessages()

{

}

public ChatMessages(int IdUser1, int IdUser2)

{

user1ID = IdUser1;

user2ID = IdUser2;

if (messages == null)

messages = new List<string>();

}

public int user1ID;

public int user2ID;

public List<string> messages;

}

Класс ChatGroup – хранит Id всех участников группы и сообщения между ними. Данный класс представлен в листинге 3.4.

Листинг 3.4 – Класс ChatGroup

public class ChatGroup

{

public ChatGroup()

{

IdUsers = new List<int>();

Messages = new List<string>();

}

public int Id { get; set; }

public string Name { get; set; }

public List<int> IdUsers { get; set; }

public List<string> Messages { get; set; }

}

Класс SignIn - это форма для входа в аккаунт. Методы, определенные в данном классе:

* Form1\_Load(object sender, EventArgs e), представленный в листинге 3.5, подключается к Firebase;
* btnSignIn\_Click(object sender, EventArgs e), представленный в листинге 3.6, проверяет авторизован ли пользователь в приложении, и входит в аккаунт;
* linkSignUp\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e), представленный в листинге 3.7, переходит на форму регистрации;
* SignIn\_FormClosed(object sender, FormClosedEventArgs e), представленный в листинге 3.8, отвечает за выключения формы;

Листинг 3.5 – Метод Form1\_Load(object sender, EventArgs e)

private void Form1\_Load(object sender, EventArgs e)

{

try

{

\_client = new FireSharp.FirebaseClient(\_config);

if (\_client != null)

{

this.CenterToScreen();

}

}

catch (Exception)

{

MessageBox.Show("Connection Fail.");

}

}

Листинг 3.6 – Метод btnSignIn\_Click(object sender, EventArgs e)

private void btnSignIn\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(txbEmail.Text) && string.IsNullOrEmpty(txbPassword.Text))

{

MessageBox.Show("enter all date");

return;

}

FirebaseResponse response = \_client.Get("Users/");

if (response.Body == "null")

goto end;

List<User> users = response.ResultAs<List<User>>();

foreach (var user in users)

{

if (txbEmail.Text == user.Email)

{

if (txbPassword.Text == user.Password)

{

new Thread(() => Application.Run(new Chat(user))).Start();

Продолжение листинга 3.6

this.Close();

return;

}

MessageBox.Show("wrong password");

return;

}

}

end:

MessageBox.Show("you don't have an account");

}

Листинг 3.7 – Метод linkSignUp\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)

private void linkSignUp\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)

{

new Thread(() => Application.Run(new Registration())).Start();

this.Close();

}

Листинг 3.8 – Метод SignIn\_FormClosed(object sender, FormClosedEventArgs e)

private void SignIn\_FormClosed(object sender,FormClosedEventArgs e)

{

Application.Exit();

}

Класс Registration - это форма для регистрации аккаунта. Методы, определенные в данном классе:

* Registration\_Load(object sender, EventArgs e), представленный в листинге 3.9, подключается к Firebase;
* btnSignUp\_Click (object sender, EventArgs e), представленный в листинге 3.10, регистриует пользователя, если он не был зарегистрирован;
* SignIn\_FormClosed(object sender, FormClosedEventArgs e), представленный в листинге 3.11, отвечает за выключения формы;

Листинг 3.9 – Метод Registration\_Load(object sender, EventArgs e)

private void Registration\_Load(object sender, EventArgs e)

{

try

{

\_client = new FireSharp.FirebaseClient(\_config);

if (\_client != null)

{

this.CenterToScreen();

//this.Size = Screen.PrimaryScreen.WorkingArea.Size;

//this.WindowState = FormWindowState.Normal;

}

}

catch (Exception)

Продолжение листинга 3.9

{

MessageBox.Show("Connection Fail.");

}

}

Листинг 3.10 – Метод btnSignUp\_Click (object sender, EventArgs e)

async private void btnSignUp\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(txbEmail.Text) && string.IsNullOrEmpty(txbPassword.Text) && string.IsNullOrEmpty(txbName.Text))

{

MessageBox.Show("enter all date");

return;

}

FirebaseResponse response = \_client.Get("Users/");

int lastIdUser = 0;

if (response.Body == "null")

{

users = new List<User>();

lastIdUser = -1;

goto createUser;

}

users = response.ResultAs<List<User>>();

foreach (var user in users)

{

if (txbEmail.Text == user.Email)

{

MessageBox.Show("You already have an account");

return;

}

}

if(txbPassword.Text != txbConfirmPassword.Text)

{

MessageBox.Show("Passwords are different, enter passwords and try again");

return;

}

lastIdUser = users.Last().Id;

createUser:

User newUser = new User()

{

Id = lastIdUser + 1,

Name = txbName.Text,

Email = txbEmail.Text,

Password = txbPassword.Text,

IsOnline = false,

DateTime = DateTime.Now,

Продолжение листинга 3.10

};

users.Add(newUser);

await \_client.SetAsync("Users/", users);

await \_client.SetAsync($"WhoIsOnline/{newUser.Name}:{newUser.Id}", "null");

txbEmail.Text = string.Empty;

txbPassword.Text = string.Empty;

txbName.Text = string.Empty;

this.Close();

}

Листинг 3.11 – Метод SignIn\_FormClosed(object sender, FormClosedEventArgs e)

private void Registration\_FormClosed(object sender, FormClosedEventArgs e)

{

SignIn signIn = new SignIn();

this.Hide();

signIn.ShowDialog();

}

Класс Chat - это форма переписки между пользователями или в группе. Методы, определенные в данном классе:

* Chat(User thisUser), представленный в листинге 3.12, устанавливает соединение с сервером;
* Chat\_Load(object sender, EventArgs e), представленный в листинге 3.13, подключается к Firebase;
* DisplayGroup(), представленный в листинге 3.14, отображает все доступные группы;
* InitializeSignalR(), представленный в листинге 3.15, инициализирует методы для получения сообщений через SignalR;
* PushId(string connectionId), представленный в листинге 3.16, отправляет connectionId на Firebaseдля отправки сообщений выбранному пользователю;
* DisplayUsers(), представленный в листинге 3.17, отображает всех пользователя;
* DisplayUserMessages(User user), представленный в листинге 3.18, отображает все сообщения с пользователем;
* createLabelMessage(int fromId, string message), создает label в котором находится сообщение;
* btnSend\_Click(object sender, EventArgs e), представленный в листинге 3.20,отправляет сообщения пользователю или в группу;
* SendInGroup(int selectedIndex, string message), представленный в листинге 3.21, отправка сообщения в группу;
* SendMessageToOne(int receiverId, string message), представленный в листинге 3.22, отправка сообщений пользователю;
* Chat\_FormClosed(object sender, FormClosedEventArgs e), представленный в листинге 3.23, отвечает за выключение формы;
* DisplayGroupMessages(int idGroup), представленный в листинге 3.24, отображает сообщения группы;
* listUsers\_SelectedIndexChanged(object sender, EventArgs e) представленный в листинге 3.25, отвечает за выбор пользователя и отображение его последнего посещения приложения.

Листинг 3.12 – Конструктор Chat(User thisUser)

public Chat(User thisUser)

{

InitializeComponent();

this.thisUser = thisUser;

thisUser.IsOnline = true;

indexSelectedUser = -1;

messageDepth = 50;

this.Text += $" ({thisUser.Name})";

connection = new HubConnectionBuilder()

.WithUrl($"http://{ConfigurationManager.AppSettings.Get("serverIp")}:{ConfigurationManager.AppSettings.Get("serverPort")}/chat")

.Build();

}

Листинг 3.13 – Метод Chat\_Load(object sender, EventArgs e)

private void Chat\_Load(object sender, EventArgs e)

{

try

{

\_client = new FireSharp.FirebaseClient(\_config);

if (\_client != null)

{

this.CenterToScreen();

}

}

catch (Exception)

{

MessageBox.Show("Connection Fail.");

return;

}

InitializeSignalR();

DisplayUsers();

DisplayGroup();

}

Листинг 3.14 – Метод DisplayGroup()

private void DisplayGroup()

{

FirebaseResponse response = \_client.Get("Group/");

if (response.Body == "null")

{

MessageBox.Show("don't have group");

return;

}

chatGroups = response.ResultAs<List<ChatGroup>>();

foreach (var group in chatGroups)

{

listUsers.Items.Add(group.Name + " (Group)");

}

}

Листинг 3.15 – Метод InitializeSignalR()

private async void InitializeSignalR()

{

connection.On<int, string>("Receive", (senderUserId, message) =>

{

Invoke((Action)(() =>

{

if (senderUserId != -1)

{

int receiverID = int.Parse(message);

List<User> user = users.Where(u => u.Id == receiverID).ToList();

if (user.Count != 0)

{

user[0].IsOnline = false;

if (user[0].Name == labUserName.Text)

labLastTime.Text = "last seen recently";

}

}

else

{

int receiverID = int.Parse(message);

List<User> user = users.Where(u => u.Id == receiverID).ToList();

if (user.Count != 0)

{

user[0].IsOnline = true;

if (user[0].Name == labUserName.Text)

labLastTime.Text = "online";

}

}

}));

Продолжение листинга 3.15

});

connection.On<int, string, int>("ReceiveToGroup", (senderUserId, message, groupId) =>

{

Invoke((Action)(() =>

{

if(labUserName.Text == chatGroups.FirstOrDefault(g => g.Id == groupId).Name)

{

createLabelMessage(senderUserId, message);

}

}));

});

connection.On<int, string, string>("ReceiveToUser", (senderUserId, message, receiverId) =>

{

Invoke((Action)(() =>

{

var user = users.Where(u => u.Id == senderUserId).ToList()[0];

createLabelMessage(senderUserId, message);

}));

});

try

{

// подключемся к хабу

await connection.StartAsync();

await connection.InvokeAsync("Send", -1, $"{thisUser.Id}");

PushId(connection.ConnectionId);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

Листинг 3.16 – PushId(string connectionId)

async private void PushId(string connectionId)

{

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

if (!string.IsNullOrEmpty(connectionId))

{

Продолжение листинга 3.16

FirebaseResponse response = \_client.Get($"WhoIsOnline/{thisUser.Name}:{thisUser.Id}");

if (response.Body == "null")

{

response = \_client.Get("WhoIsOnline/");

if (response.Body != "null")

{

usersId = response.ResultAs<Dictionary<string, string>>();

}

else

{

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

}

usersId.Add($"{thisUser.Name}:{thisUser.Id}", connectionId);

thisConnectionId = connectionId;

\_client.Set("WhoIsOnline/", usersId);

}

else

{

string str = response.ResultAs<string>();

thisConnectionId = connectionId;

usersId[$"{thisUser.Name}:{thisUser.Id}"] = thisConnectionId;

await \_client.UpdateAsync("WhoIsOnline/", usersId);

}

}

else

{

MessageBox.Show("something doesn't work");

return;

}

}

Листинг 3.17 – Метод DisplayUsers()

private void DisplayUsers()

{

FirebaseResponse response = \_client.Get("Users/");

if (response.Body == "null")

{

MessageBox.Show("don't have users");

return;

}

users = response.ResultAs<List<User>>();

users[users.IndexOf(thisUser)].IsOnline = true;

\_client.Set("Users/", users);

response = \_client.Get("Users/");

int indexDelete = users.IndexOf(thisUser);

Продолжение листинга 3.17

users.RemoveAt(indexDelete);

foreach (var user in users)

{

listUsers.Items.Add(user.Name);

}

}

Листинг 3.18 – Метод DisplayUserMessages(User user)

private void DisplayUserMessages(User user)

{

panelMessages.Controls.Clear();

messageDepth = 0;

FirebaseResponse response = \_client.Get("Chat/");

if (response.Body == "null")

{

MessageBox.Show("nobody don't use this Chat");

return;

}

chatMessages = response.ResultAs<Dictionary<string, ChatMessages>>();

foreach (var message in chatMessages)

{

List<int> idUsers = message.Key.Split(':').ToList().Select(int.Parse).ToList();

if ((idUsers.First() == thisUser.Id && idUsers.Last() == user.Id) || (idUsers.First() == user.Id && idUsers.Last() == thisUser.Id))

{

FromTo = $"{idUsers.First()}:{idUsers.Last()}";

ourMessage = message.Value;

}

}

if (ourMessage == null)

{

MessageBox.Show("you don't send anyone message");

return;

}

foreach (var fromMessage in ourMessage.messages)

{

List<string> strings = fromMessage.Split(":", 2).ToList();

int from = int.Parse(strings[0]);

string message = strings[1];

createLabelMessage(from, message);

}

}

Листинг 3.19 – Метод createLabelMessage(int fromId, string message)

private void createLabelMessage(int fromId, string message)

{

string nameAndMessage;

int messageAndNameLenght;

if (fromId == thisUser.Id)

{

nameAndMessage = $"I:{message}";

messageAndNameLenght = message.Length + 2;

}

else

{

nameAndMessage = $"{users.FirstOrDefault(u => u.Id == fromId).Name}:{message}";

messageAndNameLenght = message.Length + users.FirstOrDefault(u => u.Id == fromId).Name.Length;

}

Label labelMessage = new Label

{

Text = nameAndMessage,

Size = new Size(250, (messageAndNameLenght < 15) ? 30 : (messageAndNameLenght < 55 ? messageAndNameLenght / 15 \* 30 : (messageAndNameLenght < 100 ? messageAndNameLenght / 20 \* 30 : messageAndNameLenght / 25 \* 30))),

BorderStyle = BorderStyle.Fixed3D,

};

if (fromId == thisUser.Id)

labelMessage.Location = new Point(splitContainer1.Panel2.Width - labelMessage.Width - 100, messageDepth);

else

labelMessage.Location = new Point(10, messageDepth);

messageDepth += (labelMessage.Height + 10);

panelMessages.Controls.Add(labelMessage);

}

Листинг 3.20 – Метод btnSend\_Click(object sender, EventArgs e)

async private void btnSend\_Click(object sender, EventArgs e)

{

if (listUsers.SelectedIndex == -1)

{

MessageBox.Show("select user");

return;

}

if (string.IsNullOrEmpty(txbMessage.Text))

{

MessageBox.Show("enter your message");

return;

}

if(indexSelectedUser >= users.Count)

{

Продолжение листинга 3.20

SendInGroup(indexSelectedGroup, txbMessage.Text);

}

else

{

FirebaseResponse response = \_client.Get("Chat/");

if (response.Body == "null")

{

MessageBox.Show("no one don't use chat");

goto sendMessage;

//return;

}

if (users[indexSelectedUser].IsOnline)

SendMessageToOne(users[indexSelectedUser].Id, txbMessage.Text);

createLabelMessage(thisUser.Id, txbMessage.Text);

//await connection.InvokeAsync("Send", thisUser.Id, txbMessage.Text);

sendMessage:

if (ourMessage == null)

{

ourMessage = new ChatMessages()

{

user1ID = thisUser.Id,

user2ID = users[indexSelectedUser].Id,

messages = new List<string>()

};

}

ourMessage.messages.Add($"{thisUser.Id}:{txbMessage.Text}");

if (chatMessages.ContainsKey($"{ourMessage.user1ID}:{ourMessage.user2ID}"))

chatMessages[$"{ourMessage.user1ID}:{ourMessage.user2ID}"].messages = ourMessage.messages;

else

chatMessages.Add($"{ourMessage.user1ID}:{ourMessage.user2ID}", ourMessage);

await \_client.UpdateAsync("Chat/", chatMessages);

}

txbMessage.Text = string.Empty;

}

Листинг 3.21 – Метод SendInGroup(int selectedIndex, string message)

async private void SendInGroup(int selectedIndex, string message)

{

chatGroups[selectedIndex].Messages.Add($"{thisUser.Id}:{message}");

Продолжение листинга 3.21

await \_client.UpdateAsync($"Group/{selectedIndex}", chatGroups[selectedIndex]);

await connection.InvokeAsync("SendToGroup", thisUser.Id, message, 0);

}

Листинг 3.22 – Метод SendMessageToOne(int receiverId, string message)

async private void SendMessageToOne(int receiverId, string message)

{

FirebaseResponse response = \_client.Get($"Users/{receiverId}");

User user = response.ResultAs<User>();

string userConnectionId = \_client.Get($"WhoIsOnline/{user.Name}:{user.Id}").ResultAs<string>();

await connection.InvokeAsync("SendToUser", thisUser.Id, userConnectionId, message);

}

Листинг 3.23 – Метод Chat\_FormClosed(object sender, FormClosedEventArgs e)

async private void Chat\_FormClosed(object sender, FormClosedEventArgs e)

{

FirebaseResponse response = \_client.Get("Users/");

users = response.ResultAs<List<User>>();

//users.IndexOf(thisUser);

users[users.IndexOf(thisUser)].IsOnline = false;

users[users.IndexOf(thisUser)].DateTime = DateTime.Now;

\_client.Set("Users/", users);

DeleteConnectionId();

await connection.InvokeAsync("Send", 0, $"{thisUser.Id}");

}

Листинг 3.24 – Метод DisplayGroupMessages(int idGroup)

private void DisplayGroupMessages(int idGroup)

{

panelMessages.Controls.Clear();

messageDepth = 0;

FirebaseResponse response = \_client.Get($"Group/");

if (response.Body == "null")

{

MessageBox.Show("nobody don't use this group");

return;

}

chatGroups = response.ResultAs<List<ChatGroup>>();

ChatGroup group = chatGroups.Where(c => c.Id == idGroup).ToList()[0];

if (group.Messages.Count == 0)

MessageBox.Show("no one sent messages");

foreach (var messages in group.Messages)

{

Продолжение листинга 3.24

List<string> strings = messages.Split(":", 2).ToList();

int from = int.Parse(strings[0]);

string message = strings[1];

createLabelMessage(from, message);

}

}

Листинг 3.25 – Метод listUsers\_SelectedIndexChanged(object sender, EventArgs e)

private void listUsers\_SelectedIndexChanged(object sender, EventArgs e)

{

if (listUsers.SelectedIndex == indexSelectedUser)

return;

indexSelectedUser = listUsers.SelectedIndex;

if (indexSelectedUser < users.Count)

{

ourMessage = null;

companion = users[indexSelectedUser];

DisplayUserMessages(users[indexSelectedUser]);

labLastTime.Visible = true;

labUserName.Visible = true;

labUserName.Text = users[indexSelectedUser].Name;

DateTime dateTimeUser = users[indexSelectedUser].DateTime;

if (users[indexSelectedUser].IsOnline)

labLastTime.Text = ONLINE;

else if(DateTime.Now.Day == users[indexSelectedUser].DateTime.Day)

labLastTime.Text = $"last seen at {dateTimeUser.Hour}:{dateTimeUser.Minute}";

else if(DateTime.Now.Day - 1 == users[indexSelectedUser].DateTime.Day)

labLastTime.Text = $"last seen yesterday at {dateTimeUser.Hour}:{dateTimeUser.Minute}";

else

labLastTime.Text = $"last seen {dateTimeUser.Day}.{dateTimeUser.Month}.{dateTimeUser.Year} {dateTimeUser.Hour}:{dateTimeUser.Minute}";

}

else

{

indexSelectedGroup = indexSelectedUser - users.Count;

DisplayGroupMessages(indexSelectedGroup);

string nameGroup = chatGroups.FirstOrDefault(g => g.Id == indexSelectedGroup).Name;

labUserName.Text = nameGroup;

labUserName.Visible = true;

labLastTime.Visible = false;

}

}

# 3.2 Сопроводительная документация

Сопроводительная документация по разработанному программному продукту предоставляется в составе технического задания (приложение А) согласно ГОСТ 19.201-78.

Требования к сопроводительной документации устанавливаются государственными стандартами ЕСПД.

# 3.3 Анализ ПО

Для анализа данного программного обеспечения используется анализ метрик кода.

Метрика программного обеспечения – мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций [7].

Microsoft предоставляет встроенное в Visual Studio средство, которое предоставляет возможность сделать анализ ПО. При анализе метрик будут учитываться следующие критерии: индекс удобства поддержки – оценивает простоту обслуживания кода, сложность организации циклов – определяет число ветвей, глубина наследования – определяет число уровней в иерархии наследования объекта, взаимозависимость классов – определяет число классов, на которые есть ссылки, строки кода – приблизительно оценивает число строк исполняемого кода. Результаты метрик кода проекта представлены на рисунке 3.1.

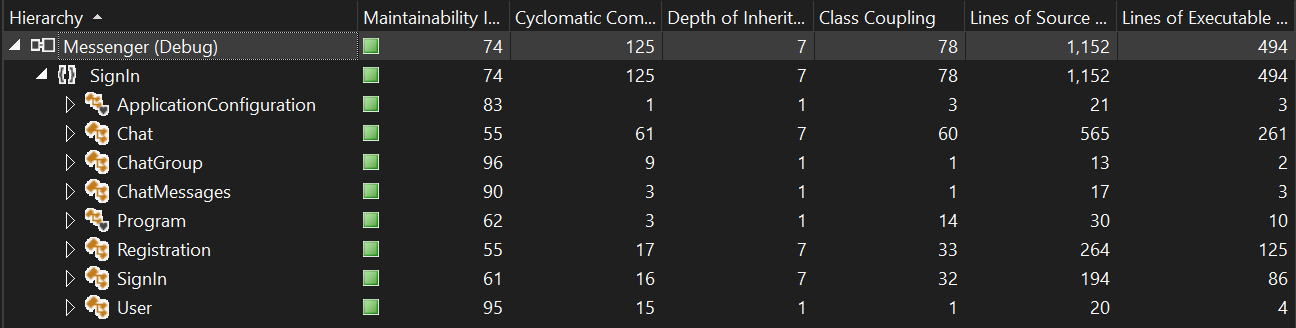


Рисунок 3.1 – Результаты метрик кода проекта

Индекс удобства поддержки в проекте составил 74 из 100, что является хорошим показателем. Сложность организации циклов – 125. Глубина наследования – 7. Взаимозависимость классов – 78. Количество строк кода – 494.

В целом, результаты метрик кода являются достаточно хорошими, хотя некоторые области, могут требовать дополнительного внимания из-за высокой сложности и взаимозависимости классов.

# 4 МЕТОДИКА И РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Тестирование программного обеспечения – процесс соответствия между реальным поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. В более широком смысле, тестирование – одна из техник контроля качества, включающая в себя активности по планированию работ, проектированию тестов, выполнению тестирования и анализу полученных результатов[8].

**Недостатки:**

* тестирование не может производиться на ранних этапах: имеется необходимость ожидание создания пользовательского интерфейса;
* нельзя провести более тщательное тестирование, с покрытием большого количества путей выполнения программы.

При запуске программы появляется форма входа в аккаунт, оно представлено на рисунке 3.2.

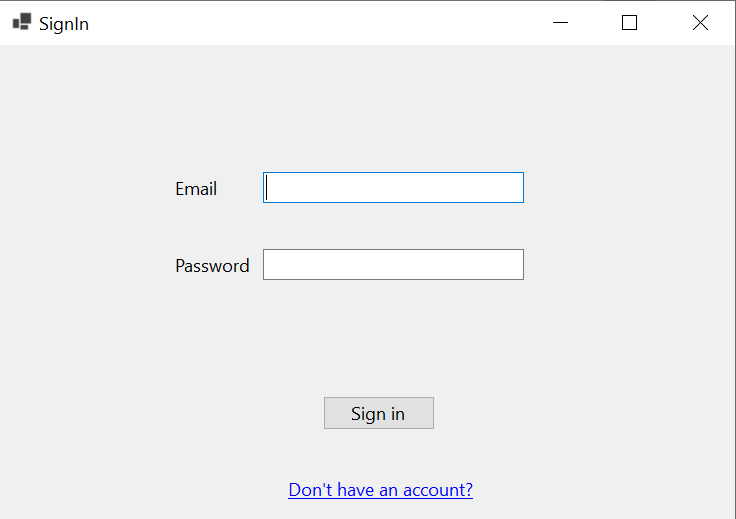


Рисунок 3.2 – форма входа в аккаунт

При нажатии на linklabel «Don’t have an account?» в форме входа в аккаунт, происходит переход в форму регистрации. При вводе данных и нажатие на кнопку «Sign in», происходит проверка данных. При успешной проверке, происхожит переход в форму чата. Форма регистрации представлено на рисунке 3.3, а форма чата – на рисунке 3.4.

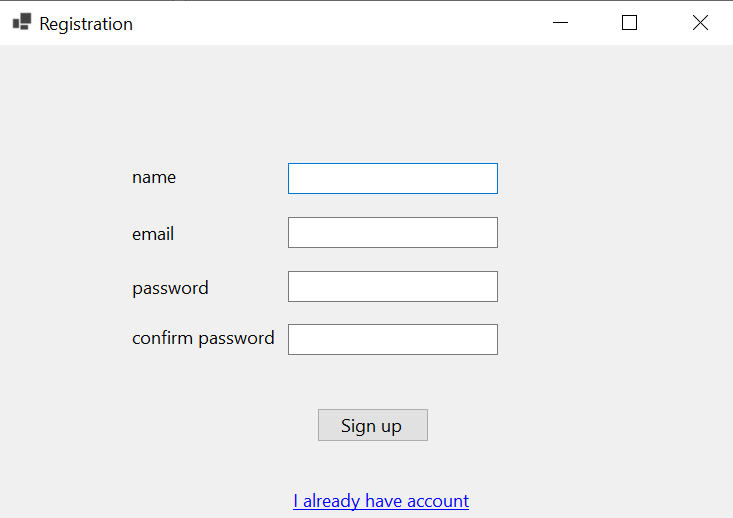


Рисунок 3.3 – Форма регистрации

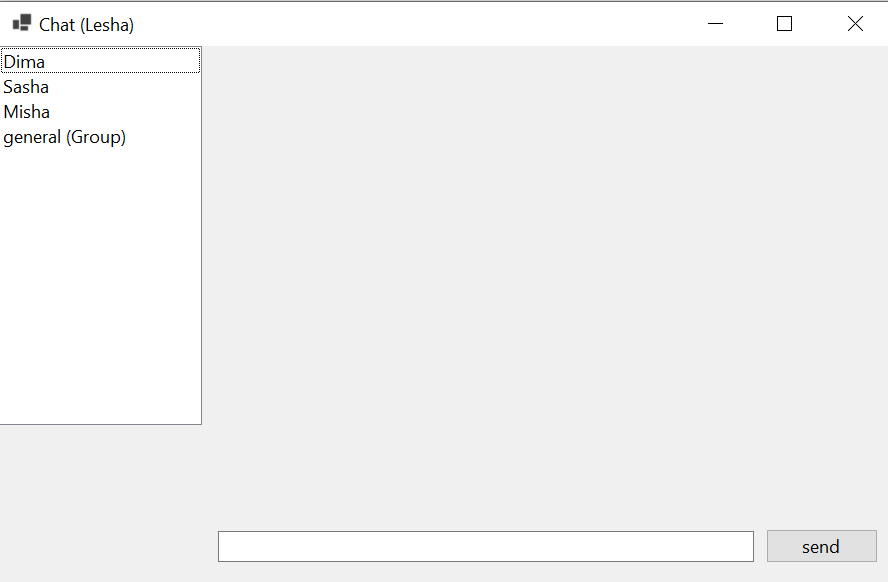


Рисунок 3.4 – Форма чата

При нажатии на имя пользователя или имя группы, отобразятся сообщения переписки с выбранным пользователем или группой. Это продемонстрировано на рисунке 3.5.

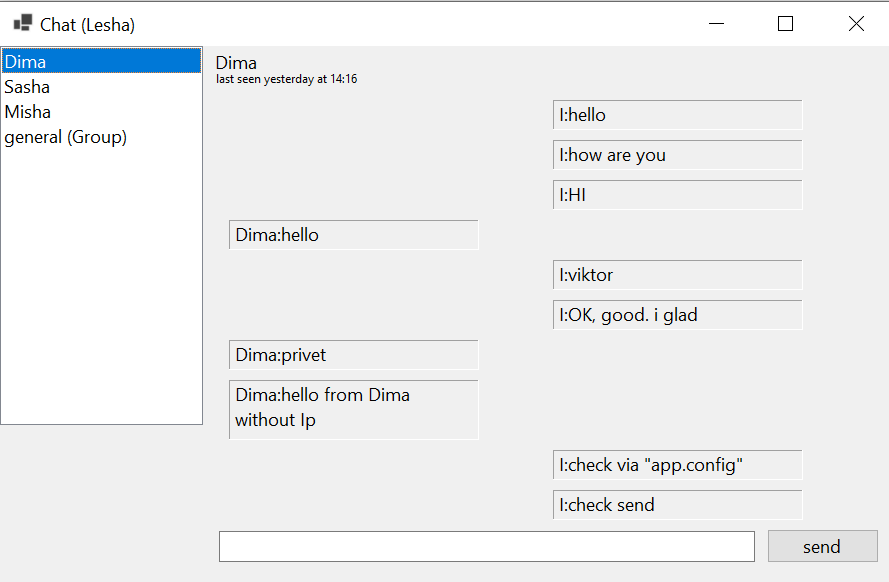


Рисунок 3.5 – отображение сообщений переписки с пользователем

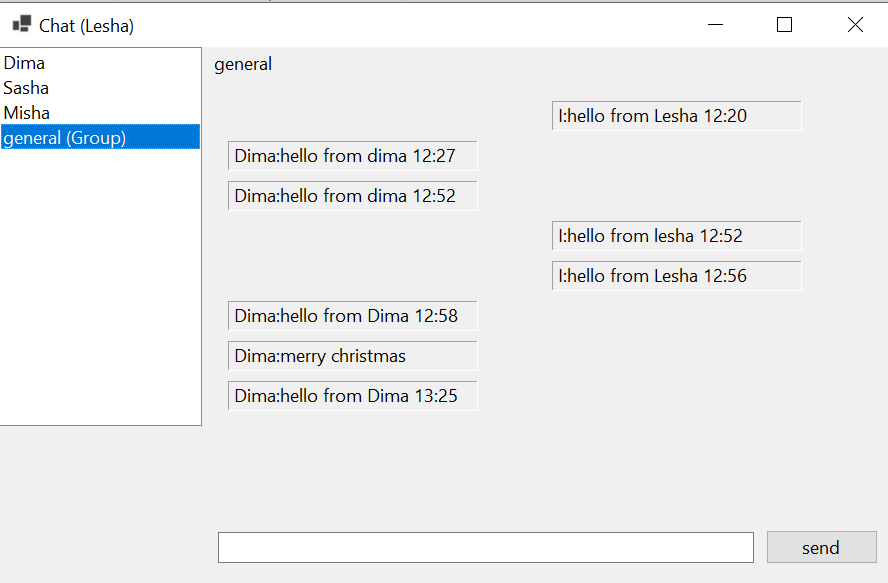


Рисунок 3.5 – отображение сообщений переписки группы

При вводе текста и нажатие на кнопку «send» сообщение отправиться пользователю и отобразится у пользователя, который отправлял. Отображение сообщения у отправителя представлен на рисунке 3.6.

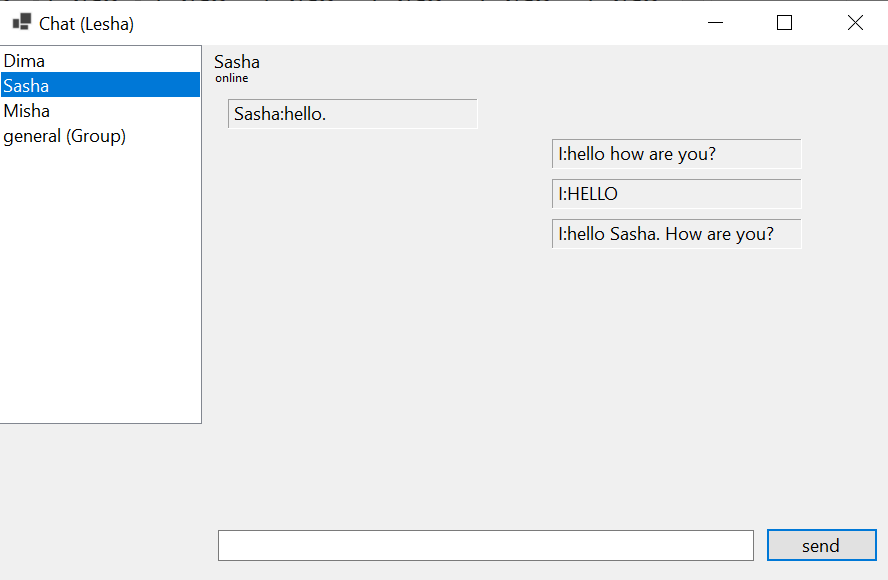


Рисунок 3.6 – Отображение сообщения у отправителя

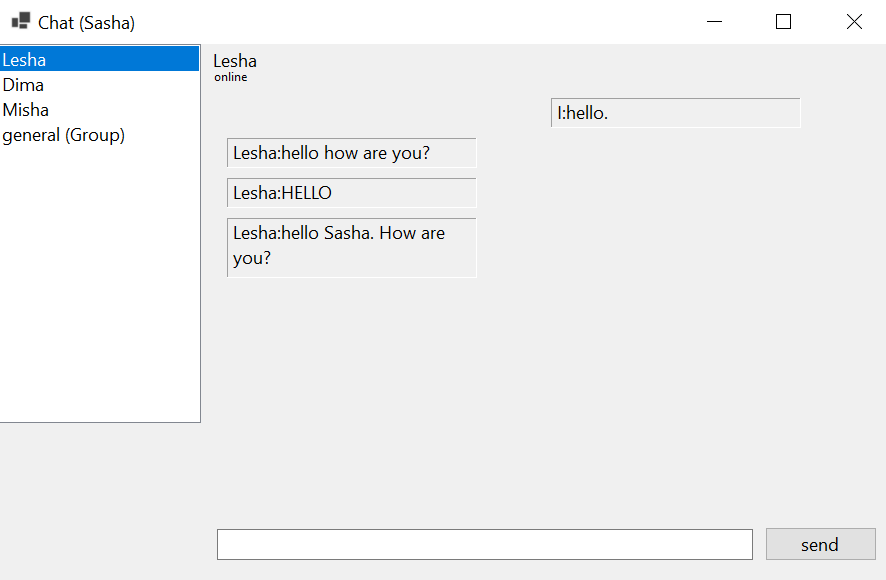


Рисунок 3.7 – Отображение сообщения у получателя

В ходе тестирования программы не было замечено сбоев или аварийного завершения работы игрового приложения, что свидетельствует о его полной работоспособности. Результаты тестирования показаны в таблице В.1 (приложение В).

# Заключение

При разработке данного приложения были пройдены следующие этапы:

– анализ исходных данных;

– программное проектирование;

– реализация ПО;

– анализ ПО;

– тестирование ПО.

В ходе выполнения данного курсового проекта были разработаны следующие возможности:

* отправка и хранение сообщений на Firebase;
* получение сообщений от Firebase;
* отправка и получение в реальном времени с помощью сервера SignalR;

В ходе тестирования не было обнаружено сбоев в программе или аварийного завершения работы ПО, что подтверждает ее готовность к использованию. Это подчеркивает важность тщательного тестирования в процессе разработки ПО, чтобы обеспечить его надежность и стабильность.

Кроме того, во время работы над этим проектом мои навыки использования систем контроля версий, особенно Git, улучшились. Это дает возможность эффективно управлять версиями кода, отслеживать изменения и при необходимости возвращаться к предыдущим версиям. Это еще раз подчеркивает важность использования современных инструментов и методов при разработке программного обеспечения.

Данный проект позволил более углубленно понять как работают мессенджеры.

В заключение хочется отметить, что выполнение данного курсового проекта позволило не только применить и закрепить полученные теоретические знания, но и получить практический опыт разработки программного обеспечения.

# Список использованных источников

1. Kolmogorov: Что такое Firebase и почему стоит с этим познакомиться [Электронный ресурс].  – Режим доступа: https://kolmogorov.pro/what-is-firebase-chto-takoe. Дата доступа: 20.09.2023;
2. Firebase: Firebase Realtime Database [Электронный ресурс].  – Режим доступа: https://firebase.google.com/docs/database?hl=ru. Дата доступа: 06.10.2023;
3. Microsoft: Введение в SignalR [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/aspnet/signalr/overview/getting-started/introduction-to-signalr. Дата доступа: 07.10.2023;
4. Uml2: Что такое Enterprise Architect [Электронный ресурс]. – Режим доступа: https://www.uml2.ru/blog/1/. Дата доступа: 25.10.2023;
5. Wikipedia: Клиент-сервер [Электронный ресурс]. – Режим доступа:https://ru.wikipedia.org/wiki/Клиент\_—\_сервер. Дата доступа: 09.11.2023;
6. Wezom: Проектирование программного обеспечения [Электронный ресурс]. – Режим доступа: https://wezom.com.ua/blog/proektirovanie-programmnogo-obespecheniya. Дата доступа: 15.11.2023;
7. Wikipedia: Метрика программного обеспечения [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Метрика\_программного\_обеспечения. Дата доступа: 24.11.2023;
8. Protesting: Тестирование программного обеспечения [Электронный ресурс]. – Режим доступа: http://www.protesting.ru/testing/. Дата доступа: 30.11.2023;

# Приложение а

**(обязательное)**

**Техническое задание**

**Введение**

Наименование программного продукта – «Мессенджер». Разрабатываемое приложение предназначено для обмена сообщений между пользователями.

Основные функции включают возможность отправки и получения сообщений, между пользователями или в группе. Дополнительно, приложение сохраняет все сообщения, что позволяет возвратиться к переписке, просмотреть предыдущие сообщения и продолжить обсуждаемую тему.

**А.1 Основание для разработки**

«Мессенджер» разрабатывается в рамках курсового проекта студента учреждения образования «Полоцкий государственный университет имени Евфросинии Полоцкой» Чиникайло А.П. Основанием для разработки является выданное задание к курсовому проекту по теме разработки «Мессенджер».

**А.2 Назначение разработки**

Функциональное и эксплуатационное назначение «Мессенджер» – предоставление интерактивной среды с возможностью отправки сообщений другому пользователю или группе.

В первую очередь, данное приложение должно предоставить полезную функциональность, обеспечивая сохранение сообщений и отображение их в следующее использование приложения.

**А.3 Требования к программному продукту**

**А.3.1 Требования к функциональным характеристикам**

При разработке «Мессенджер» выдвинуты следующие требования к функциональным характеристикам:

1. Возможность сохранения и загрузки всех сообщений.
2. Отправка сообщений в реальном времени между пользователями или в группе.
3. Получение сообщений в реальном времени от пользователя или от группы.
4. Программа должна иметь интуитивно понятный интерфейс.

**А.3.2 Требования к надежности**

Данное приложение «Мессенджер» должно надежно функционировать и стабильность работать. При возникновении аппаратных или программных сбоев, приложение должно оповещать пользователя о проблеме. Это включает в себя, от ошиби соединения с Firebase до проблемы с подключением к сети. Все эти функции важны для обеспечения бесперебойной и приятной работай для пользователя.

**А.3.3 Условия эксплуатации**

Эксплуатация программы «Мессенджер» должна осуществляться на персональном компьютере. Минимальные требования к пользователю – умение обращаться с компьютером, знание основ работы в ОС Windows 10.

**А.3.4 Требования к составу и параметрам технических средств**

Для обеспечения устойчивости работы программного средства требуется:

* x64 процессор с тактовой частотой от 1 ГГц и выше;
* 2 ГБ ОЗУ;
* не менее 30 МБ свободного места на жестком диске;

**А.3.5 Требования к информационной и программной совместимости**

Программное средство должно удовлетворять следующему требованию: операционная система Windows 10 и выше.

**А.3.6 Требования к маркировке и упаковке**

Требования к маркировке и упаковке отсутствуют.

**А.3.7 Требования к транспортированию и хранению**

Программное средство должно храниться на электронном носителе в виде исполняемого файла.

**А.4 Требования к программной документации**

Программная документация по приложению «Мессенджер» должна быть предоставлена в следующем составе:

* техническое задание. Согласно ГОСТ 19.201-78;
* пояснительная записка. Согласно ГОСТ 19.101-77.

Требования к перечисленным программным документам устанавливаются государственными стандартами ЕСПД.

**А.5 Стадии и этапы разработки**

Разработка программы заключается в следующем:

1. Анализ исходных данных и постановка задачи проектирования, разработка технического задания.
2. Разработка интерфейса, архитектуры и структуры программы.
3. Реализация и тестирование программы.
4. Разработка программной документации.

**А.6 Порядок контроля и приемки**

Контроль и приемка программного средства осуществляется в соответствии с программой и методикой испытаний.

Для проверки корректности приложения применялись следующие программные средства:

* ОС Windows 10;
* среда разработки Visual Studio 2022 Community Edition.

Тестирование программы состояло из проверки корректности работы ранее перечисленных функций. Это включало в себя функциональное тестирование, где проверялась каждая функция приложения, а также интеграционное тестирование, где проверялось, как функции взаимодействуют друг с другом.

Основным методом испытания программы является отправка сообщения. Это включает в себя проверку, что отправленное сообщение сохранилось в базе данных, а также пришло к пользователю и отобразилось у него.

Другой метод испытания программы является получения сообщения. Этот метод состоящий из того, что отправитель отправил сообщение и пользователь получил это сообщение в реальном времени. Одновременно с этим, требуется испытать, что бы пользователь имел возможно получения сообщения из удаленной базы данных.

Кроме того, была проведена проверка о получении сообщения в группе. Оно состоит из проверки того, что все участники группы получат сообщение в реальном времени. И так же, что бы это сообщение было сохранено в удаленной базе данных.

Также было проведено тестирование на устойчивость к ошибкам, чтобы убедиться, что приложение может эффективно обрабатывать и восстанавливаться после возникновения ошибок или сбоев.

Все обнаруженные в процессе тестирования ошибки и недоработки были зарегистрированы, а затем исправлены. После исправления ошибок было проведено повторное тестирование для убеждения в том, что все проблемы были устранены.

# ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Диаграмма вариантов использования**

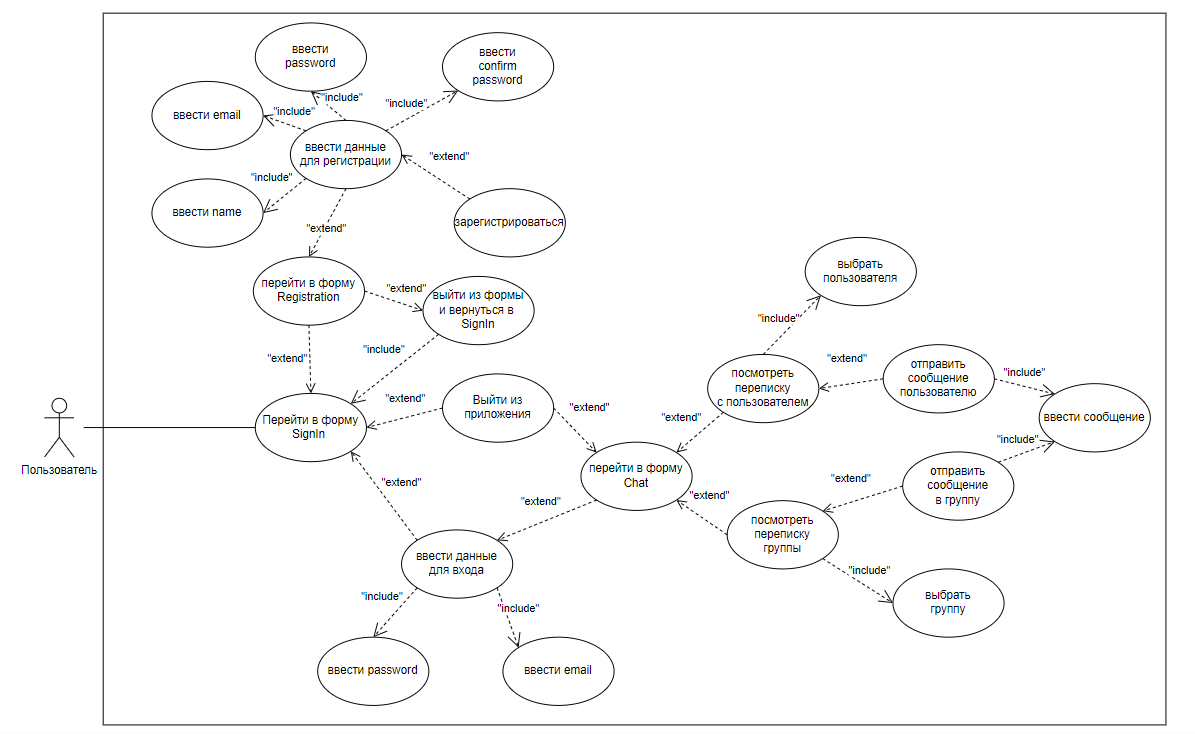


Рисунок Б.1 – Диаграмма вариантов использования

# ПРИЛОЖЕНИЕ В

**(обязательное)**

**Таблица тестирования программы**

Таблица В.1 – Способы проверок с указанием ожидаемых результатов испытаний

|  |  |  |  |
| --- | --- | --- | --- |
| Тестовый вариант | Входные данные | Ожидаемый результат | Результат тестирования |
| Запуск мессенджера | Запуск исполнительного файла | Появление формы  SignIn | Тест пройден успешно |
| Открыть форму  Registration | Нажать на “Don’t have an account” | Переход на форму  Registration | Тест пройден успешно |
| Зарегистрировать пользователя | Ввести необходимые данные для регистрации  И нажать кнопку “Sign up” | Пользователь зарегистрирован | Тест пройден успешно |
| Открыть форму  SignIn из  Registration | Нажать на “I already have an account” | Переход на форму  SignIn | Тест пройден успешно |
| Открыть форму  Chat (Войти в аккаунт) | Ввести необходимые данные для входа  И нажать кнопку “Sign in” | Переход на форму  Chat | Тест пройден успешно |
| Отправить сообщение пользователю | Выбрать пользователя, написать сообщений и нажать на кнопку “send” | Сообщение отправлено и отобразилось у отправителя | Тест пройден успешно |
| Прочитать сообщение у получателя | Войти в аккаунт и открыть диалог с отправителем | Отображение сообщения в чате | Тест пройден успешно |
| Отправить сообщение в группу | Выбрать группу, написать сообщения и нажать кнопку “send” | Сообщение отправлено в чат и отобразилось у отправителя | Тест пройден успешно |
| Прочитать сообщение в группе | Войти в аккаунт и открыть группу | Отображение сообщения в группе | Тест пройден успешно |
| Выход из игры | Выбор в меню игры пункта «Exit» | Закрытие игрового окна | Тест пройден успешно |