Белорусский государственный университет

Институт непрерывного образования

Кафедра прикладной математики и информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ДИПЛОМНОМУ ПРОЕКТУ

на тему

«Программное средство «Сетевой чат»

Выполнил слушатель

группы 15ПО11-04з

Селюк Иван Чеславович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Руководитель Кравцова А.А.

Допущен к защите:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Зав. кафедрой Лесун Б.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, ФИО, дата)

Минск 2013

Белорусский государственный университет

Институт непрерывного образования

Кафедра прикладной математики и информатики

|  |  |
| --- | --- |
| Группа 15ПО11-04з | УТВЕРЖДАЮ  Зав. кафедрой  Б.В. Лесун  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_» \_\_\_\_\_\_\_\_\_ 20\_\_ г. |

ЗАДАНИЕ

на дипломное проектирование

Слушателю Селюку Ивану Чеславовичу.

1 Тема проекта: «Программное средство «Сетевой чат»*.*

2 Срок сдачи законченного проекта «10» октября 2013 г.

3 Исходные данные по проекту: программное средство проектируется в архитектуре клиент-сервер на объектно-ориентированном языке программирования. Необходимо разработать собственную иерархию классов с использованием сокрытия данных (инкапсуляция), наследования, виртуальных методов (полиморфизм), перегрузки и переопределения методов. Так же можно использовать интерфейс программирования приложений (API) предоставляемый операционной системой. Предусмотреть обработку исключительных ситуаций.

Необходимо: разработанное программное средство должно позволять обмениваться текстовыми сообщениями в локальной сети, поддерживать функции регистрации нового пользователя, передачи файлов, сохранения ранее переданных сообщений, входа и выхода из системы зарегистрированных пользователей, иметь возможность настройки сетевых подключений.

4 Состав проекта:

а) пояснительная записка (введение, основная часть (анализ и разработка требований, проектирование программного средства, разработка программного средства, тестирование программного средства, расчет экономических показателей, охрана труда и техника безопасности, руководство пользователя), заключение).

б) графическая часть проекта (7 слайдов – диаграмма вариантов использования, диаграмма взаимодействия при отправке сообщения, диаграмма классов модуля сообщений, диаграмма состояний UDP сервера, выбранные средства разработки, модель взаимодействия потоков при получении сообщений, пользовательский интерфейс).

5 Календарный график работы на весь период проектирования:

09.07.2013 – 15.07.2013 – анализ и разработка требований;

15.07.2013 – 29.07.2013 – проектирование программного средства;

29.07.2013 – 20.08.2013 – разработка приложения;

20.08.2013 – 25.08.2013 – тестирование программного средства;

25.08.2013 – 01.09.2013 – расчет экономических показателей;

01.09.2013 – 30.09.2013 – оформление пояснительной записки;

30.09.2013 – 09.10.2013 – подготовка графического материала;

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Кравцова

Дата выдачи задания «08» июля 2013 г.

Задание принял к исполнению: «08» июля 2013 г.

Подпись слушателя: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

АННОТАЦИЯ

к дипломному проекту Селюка Ивана Чеславовича

слушателя группы 15ПО11-04з по специальности   
1-40 01 73 «Программное обеспечение информационных систем»

на тему

«Программное средство «Сетевой чат»

Ключевые слова: ПРОГРАММНОЕ СРЕДСТВО, СООБЩЕНИЕ, СОКЕТ, ЛОКАЛЬНАЯ СЕТЬ, СЕТЕВОЙ ПРОТОКОЛ, СТЕК СЕТЕВЫХ ПРОТОКОЛОВ.

Предметной областью курсового проектирования является общение пользователей в локальной сети, с помощью передачи текстовых сообщений.

Целью курсового проекта является создание программного средства, которое позволит обмениваться текстовыми сообщениями в локальной сети и, тем самым, организовать систему общения между работниками конкретного предприятия.

При выполнении курсового проектирования были использованы среда визуального программирования Visual Studio 2010, среда проектирования информационных систем Rational Rose 2000.

Пояснительная записка к курсовому проекту включает перечень перечень определений, введение, семь глав и заключение, содержит 69 страниц, в том числе 5 таблиц, 9 рисунков, 7 листингов, 4 приложения, 7 литературных источников.

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Определения…………………………………………………………………..... | 7 |
| Введение………………………………………………………………………... | 9 |
| 1 Анализ и разработка требований к программному средству……………... | 11 |
| 1.1 Анализ предметной области…………………………………………. | 11 |
| 1.2 Разработка требований к программному средству…………………. | 12 |
| 1.2.1 Общее описание……………………………………………….. | 12 |
| 1.2.2 Функциональность системы………………………………….. | 12 |
| 1.2.3 Требования к внешним интерфейсам………………………... | 13 |
| 2 Проектирование программного средства…………………………………... | 15 |
| 2.1 Общие сведения………………………………………………………. | 15 |
| 2.2 Разработка диаграммы вариантов использования………………….. | 15 |
| 2.3 Разработка диаграмм взаимодействия………………………………. | 16 |
| 2.4 Разработка диаграмм классов………………………………………... | 19 |
| 2.5 Разработка диаграмм состояний……………………………………... | 20 |
| 3 Разработка программного средства…………………………………………. | 23 |
| 3.1 Обоснование выбора средств и инструментов разработки………... | 23 |
| 3.2 Разработка модулей программного средства……………………….. | 27 |
| 3.2.1 Разработка модуля пользовательского интерфейса………… | 27 |
| 3.2.2 Разработка модуля компонентов……………………………... | 28 |
| 3.2.3 Разработка модуля отправки и получения сообщений……... | 30 |
| 3.2.4 Разработка модуля отправки и получения файлов………… | 32 |
| 4 Тестирование программного средства……………………………………… | 33 |
| 4.1 Выбор методики тестирования………………………………………. | 33 |
| 4.2 Тестирование программного средства………………………………. | 33 |
| 5 Расчет экономических показателей программного средства…………… | 36 |
| 5.1 Методика расчета себестоимости программного средства……….. | 36 |
| 5.2 Расчет себестоимости программного средства……………………. | 37 |
| 5.3 Анализ эффективности внедрения…………………………………. | 43 |
| 6 Охрана труда и техника безопасности……………………………………. | 45 |
| 6.1 Общие требования безопасности………………………………….. | 45 |
| 6.2 Требования безопасности во время работы на ПК…………………. | 47 |
| 6.2.1 Требования безопасности перед началом работы………… | 47 |
| 6.2.2 Требования безопасности при выполнении работ………… | 48 |
| 6.2.3 Требования безопасности по окончании работы……………. | 50 |
| 6.3 Требования безопасности в аварийных ситуациях………………… | 50 |
| 6.4 Время регламентированных перерывов в течение рабочего дня.. | 51 |
| 7 Руководство пользователя…………………………………………………... | 52 |
| Заключение……………………………………………………………………... | 55 |
| Список использованных источников…………………………………………. | 57 |
| Приложение А………………………………………………………………….. | 58 |
| Приложение Б…………………………………………………………………... | 60 |
| Приложение В………………………………………………………………….. | 62 |
| Приложение Г………………………………………………………………….. | 64 |
| Приложение Д………………………………………………………………….. | 66 |

**ОПРЕДЕЛЕНИЯ**

Локальная вычислительная сеть (англ. Local Area Network, LAN) – компьютерная сеть, покрывающая обычно относительно небольшую территорию или небольшую группу зданий (дом, офис, фирму, институт).

Ethernet – пакетная технология передачи данных преимущественно локальных компьютерных сетей. Стандарты Ethernet определяют проводные соединения и электрические сигналы на физическом уровне, формат кадров и протоколы управления доступом к среде — на канальном уровне модели OSI.

Протокол передачи данных – набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии программного обеспечения, разнесённой в пространстве аппаратуры, соединённой тем или иным интерфейсом.

Стек протоколов – иерархически организованный набор сетевых протоколов, достаточный для организации взаимодействия узлов в сети.

Межпроцессное взаимодействие (англ. Inter-Process Communication, IPC) – набор способов обмена данными между множеством потоков в одном или более процессах. Процессы могут быть запущены на одном или более компьютерах, связанных между собой сетью. IPC-способы делятся на методы обмена сообщениями, синхронизации, разделяемой памяти и удаленных вызовов (RPC). Методы IPC зависят от пропускной способности и задержки взаимодействия между потоками и типа передаваемых данных.

Сокет (англ. socket – разъём) – название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет – абстрактный объект, представляющий конечную точку соединения.

UDP (англ. User Datagram Protocol – протокол пользовательских датаграмм) – один из ключевых элементов Internet Protocol Suite, набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые датаграммами) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных.

Дейтаграмма (англ. datagram), также датаграмма – блок информации, посланный как пакет сетевого уровня через передающую среду без предварительного установления соединения и создания виртуального канала. Датаграмма представляет собой единицу информации в протоколе (protocol data unit, PDU) для обмена информацией на сетевом и транспортном (в случае протокола UDP, UDP-датаграммы) уровнях эталонной модели OSI.

TCP (англ. Transmission Control Protocol) (протокол управления передачей) – один из основных протоколов передачи данных Интернета, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. Выполняет функции протокола транспортного уровня в стеке протоколов TCP/IP. TCP – это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета. В отличие от UDP гарантирует целостность передаваемых данных и уведомление отправителя о результатах передачи.

Многопоточность – свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

Чат – средство обмена сообщениями по компьютерной сети в режиме реального времени, а также программное обеспечение, позволяющее организовывать такое общение.

Система обмена мгновенными сообщениями (англ. Instant messaging, IM) – служба мгновенных сообщений (Instant Messaging Service, IMS), программы онлайн-консультанты (OnlineSaler) и программы-клиенты (Instant Messenger, IM) для обмена сообщениями в реальном времени через Интернет. Могут передаваться текстовые сообщения, звуковые сигналы, изображения, видео, а также производиться такие действия, как совместное рисование или игры. Многие из таких программ-клиентов могут применяться для организации групповых текстовых чатов или видеоконференций.

**ВВЕДЕНИЕ**

С развитием информационных технологий стали возможными ещё более глобальные коммуникации. Историческим «докомпьютерным» предшественником чатов, несомненно, был телефон. Ни почта, ни телеграф не позволяли общаться в режиме реального времени, и не были доступны в домашней обстановке. Изобретение и распространение телефона по планете вызвало настоящую революцию в средствах и способах общения. Возможность поговорить с собеседником на другой стороне Земли казалась настоящим чудом.

Во второй половине XX века начали бурно развиваться компьютеры. Однако долгое время они были большими и слишком дорогими, что препятствовало тому, чтобы расходовать драгоценное машинное время на забавы с обменом сообщениями вместо расчётов атомных бомб. К тому же до конца 60-х годов они не были связаны друг с другом. Предок Интернета, сеть ARPANET, в 1969 году насчитывала только четыре связанных друг с другом научных компьютера. Чуть позже, в 1971 году, была придумана электронная почта, которая стала необычайно популярна ввиду своего удобства. Постепенно появились новые службы сообщений, такие, как списки почтовой рассылки, новостные группы и доски объявлений. Однако в то время сеть ARPANET ещё не могла легко взаимодействовать с другими сетями, построенными на других технических стандартах, что затрудняло её распространение.

Программы для обмена текстовыми строками, несмотря на простоту самой идеи, появились не сразу. Примерно в 1974 году для мэйнфрейма PLATO был разработана программа Talkomatic, потенциально позволявшая общаться между тысячей терминалов системы. В 1980-x появилась система Freelancing' Round table. Однако по-настоящему популярным стал разработанный в 1988 году протокол, названный Internet Relay Chat (IRC), что примерно можно перевести как ретранслируемый интернет-разговор. Где-то в это же время появилось и распространилось само понятие «чат».

Обычно под словом «чат» подразумевается обмен текстовыми сообщениями, реже встречаются голосовой чат, видеочат (или видеоконференция). Существует огромное множество средств общения с использованием сетевых технологий. Например: интернет форумы, Web-чаты, программы для обмена мгновенными сообщениями через Интернет в реальном времени такие, как ICQ, Jabber, AIM, MSN, Yahoo, Google Talk и соответствующие сетевые протоколы. Эти протоколы используют протоколы более низкого уровня, например TCP/IP, как транспорт для передачи данных. Передаваться могут текстовые сообщения, звуковые сигналы, картинки, видео. Такие программы могут применяться для организации телеконференций.

Для этого вида коммуникации необходима клиентская программа, так называемый мессенджер (англ. message – сообщение). Он отличается от электронной почты тем, что позволяет обмениваться сообщениями в реальном времени (instant – мгновенно). Большинство программ позволяет видеть, подключены ли в данный момент абоненты, занесенные в список контактов. В ранних версиях программ все, что печатал пользователь, тут же передавалось. Если он делал ошибку и исправлял ее, это тоже было видно. В таком режиме общение напоминало телефонный разговор. В современных программах сообщения появляются на мониторе собеседника только по окончанию редактирования и отправке.

Как правило, мессенджеры не работают самостоятельно, а подключаются к главному компьютеру данной сети обмена сообщениями, называемому сервером. Поэтому мессенджеры называют ещё клиентскими программами или клиентами (термин взят из понятий о клиент-серверных технологиях).

Широкому кругу пользователей известно некоторое количество популярных сетей обмена сообщениями, таких как ICQ, MSN, Yahoo!. Каждая из этих сетей разработана отдельной группой разработчиков, имеет отдельный сервер, отличается своими правилами и особенностями. Между различными сетями обычно нет никакой взаимосвязи. Таким образом, пользователь сети ICQ не может связаться с пользователем сети MSN. Однако, ничто не мешает быть одновременно пользователем нескольких сетей.

Целью данного дипломного проекта является разработка программного средства «Сетевой чат» для локальной сети, обладающего следующими функциональными возможностями:

- передача текстовых сообщений;

- обмен файлами;

- сохранение и просмотр ранее отправленных сообщений;

- возможность настройки сетевых параметров.

Программное средство при отправке и приеме сообщений будет сочетать в себе функции как сервера, так клиента. Что позволит использовать ее в локальной сети без установки централизованного сервера. В своей работе приложение не будет требовать подключения к сети Internet.

Для обмена данными будут использоваться TCP и UDP сетевые протоколы. Для работы с базой данных будет использоваться библиотека SQLite версии 3.0.

**1 АНАЛИЗ И РАЗРАБОТКА ТРЕБОВАНИЙ   
К ПРОГРАММНОМУ СРЕДСТВУ**

**1.1 Анализ предметной области**

Создание компьютерных сетей вызвано практической потребностью пользователей удаленных друг от друга компьютеров в одной и той же информации. Сети предоставляют пользователям возможность быстрого обмена информацией, совместной работы на сетевых устройствах, и даже одновременной обработки документов.

Развитие компьютерных сетей привело к появлению различных средств общения, так называемых чатов. Обычно под словом «чат» подразумевается обмен текстовыми сообщениями, реже встречаются голосовой чат, видеочат.  
Разрабатывались не только определенные программные средства, но так же и специализированные протоколы передачи сообщений. Одним из самых первых и популярных стал разработанный в 1988 году протокол, названный Internet Relay Chat (IRC), что примерно можно перевести как ретранслируемый интернет-разговор. Впоследствии появились многие другие.

Существующие программные средства для передачи текстовых сообщений можно условно разделить на 2 группы:

- приложения, сочетающие в себе функциональные возможности, как сервера, так и клиента для приема передачи сообщений. Данные программные средства обычно используются в локальных сетях.

- приложения, выполняющие только роль клиента при передаче сообщений. Сервер чаще всего является удаленным и выполняет функции транзита переданной информации от одного клиента к другому. Данные программы-клиенты называются мессенджерами.

Разрабатываемое программное средство «Сетевой чат» будет относиться к первой группе сетевых приложений, т.е. сочетать в себе функции, как сервера для приема сообщений, так и клиента для их отправки. Таким образом, не будет необходимости устанавливать в сети дополнительное серверное приложение.

Чаты в домашних локальных сетях чаще всего используются для общения с друзьями и знакомыми. В корпоративных и офисных сетях чаты предназначены для:

- рассылки объявлений или заданий коллегам и подчиненным;

- напоминаний работникам о проведении совещаний;

- получения оповещений от администратора сети;

Архитектура сетевых чатов, как и архитектура других сетевых приложений, разделяет прикладную систему на службы, которые могут совместно и многократно использоваться множеством приложений. Во время выполнения сетевого приложения поток управления реализуется на одном или на нескольких хостах. Все компоненты такой системы обмениваются информацией, передавая друг другу данные и управление потоком по мере необходимости. Если использовать совместимые протоколы обмена информацией, то можно добиться взаимодействия отдельных компонентов, даже если базовые сети, операционные системы, аппаратные средства и языки программирования являются неоднородными.

**1.2 Разработка требований к программному средству**

**1.2.1 Общее описание**

Программное средство «Сетевой чат» будет представлять собой оконное приложение, предназначенное для отправки и получения текстовых сообщений между пользователями (компьютерами) в локальной сети.

Оно будет сочетать в себе функции, как сервера для приема сообщений, так и клиента для их отправки. Таким образом, не будет необходимости устанавливать в сети дополнительное серверное приложение.

Уровень владения персональным компьютером пользователей данного программного средства должен соответствовать уровню «Опытный пользователь».

Программное средство будет функционировать в среде под управлением операционной системы Windows.

Компьютеры, использующие сетевой чат должны быть подключены к локальной сети Ethernet. При подключении чат будет использовать UDP/IP и TCP/IP стеки сетевых протоколов.

**1.2.2 Функциональность системы**

Программное средство будет поддерживать функцию регистрации нового пользователя. При регистрации необходимо ввести имя пользователя и пароль. В дальнейшем введенные данные изменить будет невозможно.

Функцию входа в систему (логина) пользователей осуществляется каждый раз при запуске программы. При этом пользователю необходимо ввести имя и пароль. При наличии ошибок в вводимых данных пользователь будет оповещен об этом. Программа будет иметь возможность смены текущего пользователя во время работы. Только вошедший в системы пользователь имеет возможность принятия и отправки сообщений.

Отправка сообщений будет осуществляться путем ввода текста и нажатием на кнопку «Отправить».

Отправленные сообщения помещаются в окно отправленных и полученных сообщений.

Все вошедшие в систему клиенты в сети будут отображаться в списке пользователей. Список пользователей отображает только онлайн пользователей в данный момент времени. Для каждого пользователя в списке предусмотрено свое окно отправленных и полученных сообщений.

При выходе из системы или отключении пользователя он автоматически удаляется из списка пользователей у других клиентов.

Будет предусмотрена возможность настройки программного средства для работы в определенной сети (при наличии у пользователя нескольких сетевых интерфейсов) и/или с использованием различных портов для приема и отправки сообщений.

Возможность восстановления предыдущей переписки, сохраненной в базе данных.

Пользователи могут передавать друг другу файлы. Отправка файлов будет осуществляться путем перетаскивания файла на окно ввода сообщений. При этом формируется специальное сообщение, и файл передается пользователю в локальной сети.

При приеме и передаче сообщений программное средство будет использовать сетевые протоколы UDP и TCP.

**1.2.3 Требования к внешним интерфейсам**

Главное окно приложения будет иметь возможность сворачивания на панель задач, и разворачивания на всю ширину рабочего стола. Пользовать может изменить размер главного окна в зависимости от своих предпочтений.

Главное окно разделяется на три области:

- список онлайн пользователей, представляет собой вертикальный список пользователей, отображает их имена, указанные при регистрации;

- окно для ввода новых сообщений, имеет функцию редактирования;

- окно для отображения принятых и отправленных сообщений, функция редактирования отключена.

Размер каждой области можно будет изменить относительно других.

В правом нижнем углу главного окна будет расположена кнопка отправки сообщений.

Главное меню распологается вверху окна под заголовком. Меню имеет следующую структуру:

- Вход в систему («Sign In»);

Вход в систему («Sign In»);

Выход из системы («Sign Out»);

Закрытие приложения («Exit»);

- Настройки («Settings»);

- Информация о программе («About»).

Программное средство будет предназначено для работы в операционных системах Windows XP, Windows Vista, Windows 7.

Требования к оборудованию (определены экспериментально, зависят от выбранной операционной системы):

- CPU AMD Duron 1000 Mhz;

- оперативная память 256 Mb;

- 50 Mb свободного места на жестком диске;

- сетевой адаптер Ethernet;

- подключение к локальной сети.

**2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**2.1 Общие сведения**

Использование объектно-ориентированное проектирования на основе графических диаграмм языка UML позволяет:

- сократить цикла разработки приложения «заказчик – программист – заказчик»;

- увеличить продуктивности работы программистов вследствие уменьшения ошибок и времени на отладку;

- улучшить потребительских качеств программ за счет ориентации на пользователей и бизнес;

- вести большие проекты и группы проектов;

- повторно использовать уже созданное ПО за счет упора на разбор их архитектуры и компонентов;

- использовать язык UML в качестве универсального «мостика» между разработчиками из разных подразделений.

В качестве программного пакета для визуального объектно-ориентированного моделирования систем на основе классов и их взаимодействия выбран визуальный редактор Rational Rose.

Данный редактор позволил быстро разработать необходимые диаграммы, описывающие как архитектуру программного средства «Сетевой чат», так и его функциональные возможности, а так же варианты взаимодействия его структурных компонентов.

**2.2 Разработка диаграммы вариантов использования**

Одним из основных преимуществ применения диаграммы вариантов использования – это предоставляет важной информации. Она показывает, какие функциональные возможности будут заложены в систему, кто конкретно будет с ней взаимодействовать. Изучение всего множества вариантов использования и действующих лиц, позволяет определить сферу применения системы. Диаграмму вариантов использования называют диаграммой прецедентов.

При проектировании диаграммы вариантов использования для разрабатываемого программного средства были определены следующие варианты использования:

- отправка текстовых сообщений;

- получение текстовых сообщений;

- отправка файлов;

- получение файлов;

- настройка сетевого подключения;

- выбор сетевого адаптера;

- регистрация;

- вход и выход из системы;

- сохранение переданных файлов в базе данных.

Диаграмма вариантов использования для программного средства «Сетевой чат» представлена на рисунке 2.1.



**Рисунок 2.1 –** **Диаграмма вариантов использования   
программного средства «Сетевой чат»**

**2.3 Разработка диаграмм взаимодействия**

На диаграмме взаимодействия отображается один из процессов обработки информации в варианте использования. Различают два типа диаграмм взаимодействия – диаграммы последовательности и кооперативные диаграммы. Диаграммы первого типа организованы по времени, они заостряют внимание на управлении. Кооперативные диаграммы отображают поток данных.

С помощью диаграмм взаимодействия мы можем определить классы, которые нужно создать, связи между ними, а также операции и ответственности каждого класса. Диаграммы последовательности полезны для того, что бы понять логическую последовательность событий в сценарии. Кооперативные диаграммы полезны в тех случаях, когда нужно оценить последствия сделанных изменений.

Диаграммы последовательностей используются для моделирования конкретных экземпляров классов, интерфейсов, компонентов и узлов, а также сообщений, которыми они обмениваются, – и все это в контексте сценария, иллюстрирующего данное поведение. Диаграммы взаимодействий могут существовать автономно и служить для визуализации, специфицирования, конструирования и документирования динамики конкретного сообщества объектов, а могут использоваться для моделирования отдельного потока управления в составе прецедента.

В ходе проектирования ИС аналитик поэтапно спускается от общей концепции, через понимание ее логической структуры к наиболее детальным моделям, описывающим физическую реализацию.

С помощь диаграммы прецедентов (вариантов использования) выявляются основные пользователи системы и задачи, которые данная система должна решать. С помощью диаграммы деятельности мы описываем последовательность действий для каждого прецедента, необходимая для достижения поставленной цели. Далее проектируется логическая структура системы с помощью диаграммы классов. Для описания сложного поведения некоторых объектов (экземпляров класса) составляется диаграмма состояний.

Таким образом, аналитиками фиксируются такие поведенческие аспекты как алгоритм действий в рамках одного или нескольких прецедентов, необходимый для достижения определённого результата, а также изменение состояния объектов в ходе выполнения приведенных действий.

Зачастую на этапе спецификации требований необходимо показать не только алгоритм действий или изменение состояния объекта, но и обмен сообщениями между отдельными объектами системы. Данную задачу решает диаграмма взаимодействия. Диаграмма взаимодействия предназначена для моделирования отношений между объектами (ролями, классами, компонентами) системы в рамках одного прецедента. Данный вид диаграмм отражает следующие аспекты проектируемой системы:

- обмен сообщениями между объектами (в том числе в рамках обмена сообщениями со сторонними системами);

- ограничения, накладываемые на взаимодействие объектов;

- события, инициирующие взаимодействия объектов.

Диаграмма последовательности является одной из разновидности диаграмм взаимодействия и предназначена для моделирования взаимодействия объектов системы во времени, а также обмена сообщениями между ними.

Одним из основных принципов ООП является способ информационного обмена между элементами системы, выражающийся в отправке и получении сообщений друг от друга. Таким образом, основные понятия диаграммы последовательности связаны с понятием «Объект» и «Сообщение».

Диаграмма последовательности при отправке текстового сообщения пользователю в сети представлена на рисунке 2.2.



**Рисунок 2.2 –** **Диаграмма последовательности   
при отправке сообщения пользователю в сети**

На основании построенной в Rational Rose диаграммы последовательности можно автоматически получить кооперативную диаграмму. Кооперативная диаграмма при отправке сообщения пользователю в сети представлена на рисунке 2.3.



**Рисунок 2.3 –** **Кооперативная диаграмма   
при отправке сообщения пользователю в сети**

**2.4 Разработка диаграмм классов**

На диаграммах классов отображаются некоторые классы и пакеты системы. Это статические картины фрагментов системы и связей между ними.

Обычно для описания системы создают несколько диаграмм классов. На одних показывают некоторое подмножество классов и отношения между классами подмножества. На других отображают то же подмножество, но вместе с атрибутами и операциями классов. Третьи соответствуют только пакетам классов и отношениям между ними. Для представления полной картины системы можно разработать столько диаграмм классов, сколько требуется.

Диаграммы классов при моделировании объектно-ориентированных систем встречаются чаще других. На таких диаграммах отображается множество классов, интерфейсов, коопераций и отношений между ними. Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Кроме того, диаграммы классов составляют основу еще двух диаграмм – компонентов и развертывания.

В ходе проектирования информационной системы было принято решение использовать объектно-ориентированный подход. При проектировании иерархии классов необходимо исходить из правила, что каждый класс, каждая сущность должно выполнять строго определенную задачу, функцию. Для построения гибкой и расширяемой системы были использованы паттерны объектно-ориентированного проектирования [1]. Диаграмма классов модуля приема и отправки текстовых сообщений представлена на рисунке 2.4.

Существует два вида диаграмм классов:

- статический вид диаграммы рассматривает логические взаимосвязи классов между собой;

- аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов входящих в систему.

Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

- концептуальная точка зрения — диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;

- точка зрения спецификации — диаграмма классов применяется при проектировании информационных систем;

- точка зрения реализации — диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).



**Рисунок 2.4 –** **Диаграмма классов  
модуля отправки и приема текстовых сообщений**

Как видно из приведенной диаграммы в модуле отправки и получения сообщений находится UDP сервер, который считывает данные из сокета и передает их обработчику сообщений (Messages Handler). После обработки сообщения оно помещается в очередь сообщений (Messages Queue). После извлечения сообщения из очереди оно непосредственно отображается в окне пользователя. Объект (Server Settings Holder) хранит в себе настройки UDP сервера. Таким образом, применение конкретного хранителя настроек позволяет повторно использовать один и тот же класс UDP сервера.

**2.5 Разработка диаграмм состояний**

На сегодняшний день при проектировании сложной системы принято делить ее на части, каждую из которых затем рассматривать отдельно. Таким образом, при объектной декомпозиции система разбивается на объекты или компоненты, которые взаимодействуют друг с другом, обмениваясь сообщениями. сообщения описывают или представляют собой некоторые события. Получение объектом сообщения активизирует его и побуждает выполнять предписанные его программным кодом действия.

При данном подходе система становится событийно управляемой, поэтому разработчикам зачастую важно знать, как должен реагировать тот или иной объект на определенные события. Инициаторами событий могут быть как объекты самой системы, так и её внешнее окружение.

Описать поведение отдельно взятого объекта помогает диаграмма состояний. Также зачастую диаграмма состояний используется аналитиками для описания последовательности переходов объекта из одного состояния в другое.

Диаграмма состояний покажет нам все возможные состояния, в которых может находиться объект, а также процесс смены состояний в результате внешнего влияния.

Основными элементами диаграммы состояний являются «Состояние» и «Переход». Диаграмма состояний имеет схожую семантику с диаграммой деятельности, только деятельность здесь заменена состоянием, переходы символизируют действия. Таким образом, если для диаграммы деятельности отличие между понятиями «Деятельность» и «Действие» заключается в возможности дальнейшей декомпозиции, то на диаграмме состояний деятельность символизирует состояние, в котором объект находится продолжительное количество времени, в то время как действие моментально.

На диаграмме состояний отображается жизненный цикл одного объекта, начиная с момента его создания и заканчивая разрушением. С помощью таких диаграмм удобно моделировать динамику поведения класса.

Как правило, диаграммы состояний не требуется создавать для каждого класса. Многие проекты вообще обходятся без них. Если динамика поведения класса важна, то для него полезно разработать диаграмму состояний. Такие классы обычно имеют несколько различных состояний.

В среде Rational Rose на основании диаграмм состояний не генерируется никакого исходного кода. Они нужны для того, чтобы документировать динамику поведения класса.

Состояние может содержать только имя или имя и дополнительно список внутренних действий. Список внутренних действий содержит перечень действий или деятельностей, которые выполняются во время нахождения объекта в данном состоянии. Данный список фиксированный. Список основных действий включает следующие значения:

- «entry» - действие, которое выполняется в момент входа в данное состояние (входное действие);

- «exit» – действие, которое выполняется в момент выхода из данного состояния (выходное действие);

- «do» – выполняющаяся деятельность в течение всего времени, пока объект находится в данном состоянии

- «defer» – событие, обработка которого предписывается в другом состоянии, но после того, как все операции в текущем будут завершены.

На рисунке 2.5 представлена диаграмма состояний сервера приема сообщений.



**Рисунок 2.5 –** **Диаграмма состояний  
объекта класса UDP Server**

Как видно из данной диаграммы объект сервера после создания и инициализации переходит в состояние ожидания, до момента входа пользователя в систему (логина). После логина сервер переходит в состояние приема сообщений. Если пользователь вновь выйдет из системы, сервер перейдет в состояние ожидания. После завершения работы приложения сервер завершает работу и освобождает используемые ресурсы.

**3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**3.1 Обоснование выбора средств и инструментов разработки**

Для создания программного средства «Сетевой чат» была выбрана среда разработки MS Visual Studio 2010.

Интегрированная среда разработки (Integrated Development Environment, IDE) MS Visual Studio является последней по времени выпуска версией популярной и широко используемой среды разработки профессионального программного обеспечения (ПО) производства компании Microsoft.

Объединяя в своем составе все положительные стороны предыдущих версий, данная обеспечивает возможность использования всех преимуществ современной технологий разработки программного обеспечения. В числе основных достоинств MS Visual Studio, оцененных сообществом профессиональных программистов, можно отметить следующие моменты:

- повышение производительности труда разработчиков. Среда разработки MS Visual Studio продолжает традиции корпорации Microsoft в области предоставления эффективных инструментальных средств для разработчиков сложного ПО. Обеспечивая среду разработки для всех языков программирования, дополненную набором окон с интуитивно понятными инструментальными средствами, контекстной справкой и автоматизированными механизмами выполнения разнообразных задач разработки, MS Visual Studio позволяет в сжатые сроки проводить профессиональную разработку программ различного назначения;

- поддержка нескольких языков программирования. В большинстве профессиональных групп разработчиков, как правило, используется несколько языков программирования – для поддержки такой практики в MS Visual Studio впервые была обеспечена возможность использования сразу нескольких языков в рамках одной и той же среды. Благодаря применению общего конструктора для компонентов, для форматов XML и HTML, а также наличию единого отладчика, MS Visual Studio предоставляет разработчикам эффективные средства, независимые от языка программирования. Разработчикам программного обеспечения при использовании MS Visual Studio уже не придется ограничиваться одним языком программирования, адаптируя свою рабочую среду к особенностям этого языка. Более того, MS Visual Studio позволяет программистам многократно использовать уже имеющиеся у них наработки, а также навыки разработчиков, создающих свои программы на разных языках программирования;

- единая модель программирования для всех приложений. При создании приложений ранее разработчикам приходилось использовать различные приемы программирования, которые существенным образом зависели от типа приложения – технологии разработки клиентского программного обеспечения, общедоступных веб-приложений, программного обеспечения для мобильных устройств и бизнес-логики промежуточного уровня значительно различались между собой. Среда разработки MS Visual Studio решает данную проблему, предоставляя в распоряжение разработчиков единую модель создания приложений всех категорий. Эта интегрированная модель обладает привычным и одновременно интуитивно понятным интерфейсом, позволяя разработчикам использовать свои навыки и знания для эффективного создания широкого спектра приложений;

- всесторонняя поддержка жизненного цикла разработки. Среда MS Visual Studio обеспечивает поддержку всего жизненного цикла разработки: начиная с планирования и проектирования через разработку и тестирование и вплоть до развертывания и последующего управления. Обеспечивая возможность легкого расширения среды разработки посредством включения продуктов независимых разработчиков, MS Visual Studio предоставляет всестороннюю адаптируемую среду для создания всех приложений, жизненно необходимых для успешной работы современных компаний.

Написание программного кода часто требует одновременной работы с несколькими конструкторами и редакторами. MS Visual Studio помогает разработчику организовать цифровое окружение благодаря поддержке нескольких мониторов, что упрощает работу над программами.

MS Visual Studio включает в себя поддержку технологии IntelliSense с возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и как отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. MS Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода, добавление новых наборов инструментов. Например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования или инструментов для прочих аспектов процесса разработки программного обеспечения.

В качестве языка программирования был выбран язык C++. Это компилируемый, статически типизированный язык программирования общего назначения.

С++ поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков.

Являясь одним из самых популярных языков программирования, C++ широко используется для разработки программного обеспечения. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений.

В качестве инструмента разработки пользовательского интерфейса была выбрана широко используемая библиотека Qt [5]. Qt – кросс-платформенный инструментарий разработки программного обеспечения на языке программирования C++. На сегодняшний день Qt – это продукт, широко используемый разработчиками всего мира. Среди известных компаний, которые наиболее активно используют Qt можно назвать такие как: Adobe, AT&T, Cannon, HP, Bosch, Boeing, IBM, Motorola, NASA, NEC, Pioneer, Sharp, Siemens, Sony и Xerox и др.

Qt – полностью объектно-ориентированная библиотека. Новая концепция ведения межобъектных коммуникаций, именуемая «сигналы и слоты», полностью заменяет былую не вполне надежную модель обратных вызовов. Также имеется возможность обработки событий, например, нажатия клавиш клавиатуры, перемещения мыши и т.д. Предоставляемая система расширений позволяет создавать модули, расширяющие функциональные возможности приложений.

Сохранение переданных сообщений осуществляется с помощью реляционной СУБД SQLite [7]. SQLite – компактная встраиваемая реляционная база данных Она не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite.

Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени [3].

При разработке сетевых приложений, как правило, выделяют «собственную» и «случайную» сложность. Собственная сложность связана с ключевыми проблемами предметной области, затрудняющими разработку сетевого приложения, включая:

- выбор подходящих механизмов коммуникаций и создание протоколов для их эффективного использования;

- проектирование сетевых служб, которые рационально используют доступные вычислительные ресурсы;

- эффективное использование параллелизма для достижения надежной и высокой производительности системы.

Случайная сложность вытекает из ограничений, связанных с инструментальными средствами и методами, включая:

- отсутствие в операционной системе собственных типобезопасных, переносимых и расширяемых API;

- широко распространенное применение алгоритмической декомпозиции, что делает неоправданно трудными поддержку и развитие сетевых приложений.

Чтобы избежать «случайной» сложности при разработке, была выбрана популярная инструментальная библиотека для программирования сетевых приложений ACE [6]. Адаптивная коммуникационная среда (ACE) свободно доступная, объектно-ориентированная библиотека, с открытым исходным кодом, реализующая основные модели программирования сетевых приложений. ACE предоставляет богатый набор многоразовых C ++ компонентов, которые выполняют общие задачи коммуникационного программного обеспечения в целом ряде платформ.

Задачи коммуникационного программного обеспечение включают демультиплексирования и диспетчеризацию событий, обработка сигналов, инициализацию служб, взаимодействие процессов, разделяемую память управления, маршрутизации сообщений, динамическое конфигурирование распределенных служб, параллельное выполнение и синхронизацию.

ACE упрощает разработку объектно-ориентированных сетевых приложений и сервисов, которые используют взаимодействие процессов и событий, явное динамическое связывание и параллелизма. Кроме того, ACE автоматизирует конфигурирование системы и изменение конфигурации путем динамического связывания сервисов в приложении во время выполнения.

**3.2 Разработка модулей программного средства**

При проектировании структуры программного средства «Сетевой чат» вся система приложения была разделена на несколько модулей, каждый из которых выполняет определенные функции.

**3.2.1 Разработка модуля пользовательского интерфейса**

Модуль пользовательского интерфейса (UI Module) разработан с использованием библиотеки Qt и представляет собой исполняемый (.exe) файл приложения. Он состоит из классов, с помощью которых создается главное окно приложения, а так же элементы управления для удобной работы пользователя с программным средством (главное меню, диалоги настройки, логина, регистрации и др.).

Центральным классом данного модуля является класс MainFraim. Данный класс аккумулирует в себе все действия по созданию и отображению пользовательского интерфейса. Так же он предоставляет методы для управления добавлением и удалением пользователей, добавлением сообщений. В структурном плане класс MainFrame состоит из списка пользователей, окна отправленных и принятых сообщений, окна для ввода сообщений, главного меню. Создание пользовательского интерфейса разделено на несколько методов, каждый из которых отвечает за создание определенной области или объекта в главном окне приложения. Создание интерфейса главного окна приведено в листинге 3.1.

void MainFrame::SetupUI()

{

QWidget\* mainWidget = new QWidget();

mainWidget->setObjectName(MAIN\_WIDGET\_NAME);

mainWidget->setStyleSheet(MAIN\_WIDGET\_STYLE);

CreateMenuBar();

QHBoxLayout\* mainLayout = CreateMainLayout(CreateUsersWidget(),  
 CreateMessagesWidget());

mainWidget->setLayout(mainLayout);

setCentralWidget(mainWidget);

setWindowTitle(MAIN\_TITLE);

setWindowIcon(QIcon(MAIN\_ICON\_PATH));

resize(MAINFRAME\_WIDTH, MAINFRAME\_HEIGH);

}

**Листинг 3.1 –** **Метод SetupUI класса MainFrame**

**3.2.2 Разработка модуля компонентов**

Модуль компонентов (ComponentsModule) включает в себя несколько объектов для обеспечения дополнительной функциональности программного средства «Сетевой чат». Он представлен в виде отдельной статической библиотеки (.lib) которая используется остальными модулями, включая модуль пользовательского интерфейса.

В данном модуле представлены следующие классы: LoginManager (управление регистрацией и логином), SettingsManager (управление настройками приложения), NetUsersManger (управление сетевыми пользователями), DataBaseManager (работа с базой данных приложения).

Программное средство «Сетевой чат» имеет возможность сохранять сообщения принятые и переданные другим пользователям. А затем восстанавливать их из базы данных, если пользователь хочет посмотреть свою прошлую переписку. Для данной функции используется библиотека SQLite версии 3.0, которая позволяет легко работать с реляционными базами данных на языке C++.

LoginManager хранит в себе данные всех зарегистрированных пользователей, считывает и сохраняет данные реестре, предоставляет методы для получения данных, а так же для проверки их на уникальности и правильность.

std::vector<UserDataPtr> LoginManagerImpl::GetUsersData() const

{

std::vector<UserDataPtr> users;

std::for\_each(m\_users.begin(), m\_users.end(),

[&users](const std::pair<std::wstring, UserDataPtr>& it)

{

users.push\_back(it.second);

});

return users;

}

**Листинг 3.2 –** **Получение данных зарегистрированных пользователей**

В листинге 3.2 представлен метод класса для получения данных всех зарегистрированных пользователей на данном компьютере

Кроме этого данный класс хранит в себе два состояния online и offline. Объекты, поведение которых зависит от данных состояний, подписываются на событие изменения данного состояния и оповещаются при его изменении.

Класс SettingsManager хранит в себе и управляет настройками программы, определяет IP-адреса сетевых адаптеров и выбирает наиболее подходящий. Данные настройки записываются в реестр и считываются из него при запуске приложении. Другие объекты могут обратиться к SettingsManager для получения текущих настроек. Сохранение настроек в реестре представлено в листинге 3.3.

void SettingsManager::SaveSettings()

{

CRegKey key;

if (ERROR\_SUCCESS == key.Open(HKEY\_CURRENT\_USER, SETTINGS\_KEYS\_PATH,  
 KEY\_WRITE))

{

key.SetStringValue(CURRENT\_NET\_ADDRESS, m\_currentNetAddress.c\_str());

key.SetDWORDValue(CURRENT\_STATE\_MSG\_PORT, m\_stateMsgPort);

key.SetDWORDValue(CURRENT\_CHAT\_MSG\_PORT, m\_chatMsgPort);

}

}

**Листинг 3.3 –** **Сохранение настроек приложения в реестре**

Объект класса NetUsersManager управляет списком подключенных пользователей, и предоставляет другим объектом информацию об именах, сетевых адресах пользователей.

Из-за того, что невозможно уникально идентифицировать пользователя по его сетевому адресу и имени, было принято решение при регистрации нового пользователя генерировать заведомо уникальный идентификатор. Данным идентификатором является GUID.

GUID (Globally Unique Identifier) – статистически уникальный 128-битный идентификатор. Его главная особенность – уникальность, которая позволяет создавать расширяемые сервисы и приложения без опасения конфликтов, вызванных совпадением идентификаторов.

Хотя уникальность каждого отдельного GUID не гарантируется, общее количество уникальных ключей настолько велико (2128 или 3,4028×1038), что вероятность того, что в мире будут независимо сгенерированы два совпадающих ключа, практически не возможна.

Алгоритм вычисления GUID учитывает текущие дату и время, последовательность «тиков таймера» и её запоминаемое состояние, чтобы можно было обрабатывать и ситуации обратного передвижения часов, простой инкрементальный счетчик, чтобы можно было правильно обрабатывать и очень часто следующие запросы на генерацию очередного GUID, истинно глобально уникальный IEEE-идентификатор машины, извлекаемый из сетевой карты.

Вспомогательная функция для генерирования GUID идентификатора представлена в листинге 3.4.

std::wstring GenerateUUID()

{

UUID uuid = {0};

if (::UuidCreate(&uuid) == RPC\_S\_OK)

{

RPC\_WSTR str = nullptr;

if (::UuidToString(&uuid, &str) == RPC\_S\_OK)

{

std::wstring result(reinterpret\_cast<wchar\_t\*>(str));

::RpcStringFree(&str);

return result;

}

}

return std::wstring();

}

**Листинг 3.4 –** **Вспомогательная функция для генерирования GUID**

**3.2.3 Разработка модуля отправки и получения сообщений**

Модуль отправки и получения сообщений является центральным логическим модулем всей системы. Он спроектирован в виде статический библиотеки, которая используется модулем пользовательского интерфейса.

При отправке и получении сообщений используются UDP сокеты, реализация которых представлена классами ACE\_SOCK\_Dgram и ACE\_SOCK\_Dgram\_Bcast ( отправка широковещательных запросов) библиотеки ACE [6]. В механизме обмена сообщениями участвуют два типа сообщений: сообщения о состоянии клиентов, и непосредственно текстовые сообщения. Первый тип используется для обнаружения пользователей в сети и обмена контактами, другой тип для отправки текстовых сообщений чата.

Центральными классами данного модуля являются классы StateMessagesManager и ChatMessagesManager. В своей реализации они отвечают за отправку и прием сообщений. Для данной задачи они содержат сервер для приема и обработки сообщений и выполняют роль клиентов при их отправке. Метод отправки широковещательного сообщения представлен в листинге 3.5.

Работа менеджеров связана с несколькими потоками, поэтому для их синхронизации используются такие объекты как мьютексы и условные переменные.

Необходимо отметить, что сообщения отправляются в формате xml. Это гарантирует их правильный прием. При нарушении структуры сообщения из-за потери данных или других причин, данное сообщение игнорируется. Таким образом, гарантируется, что все принятые сообщения валидны.

void StateMessagesManager::SendBroadcastMessage(State currentState)

{

net::WSAStartupHolder wsaHolder(MAKEWORD(2, 2));

if (wsaHolder.GetErrorCode() == 0)

{

std::wstring message(CreateMessage(currentState));

ACE\_INET\_Addr currentAddr(TEMP\_PORT, m\_settingsHolder->GetAddress().c\_str());

ACE\_SOCK\_Dgram\_Bcast udpBroadcastSocket(currentAddr);

size\_t result = udpBroadcastSocket.send(message.c\_str(),  
 message.size() \* sizeof(wchar\_t), m\_settingsHolder->GetPort());

udpBroadcastSocket.close();

}

}

**Листинг 3.5 –** **Отправка широковещательных сообщений**

Механизм получения сообщений заключается в приеме и обработке сообщений потоком сервера. Затем сообщение помещается в очередь, после чего оповещается главный поток. Для оповещения главного потока мы создаем скрытое невидимое окно, задача которого принимать оповещения из других потоков. Нотификации происходят с помощью WinAPI функции PostMessage [2]. Данная функция является асинхронной. Главный поток извлекает сообщение из очереди и обрабатывает его. В листинге 3.6 представлена функция обработки сообщений скрытым окном, все действия происходят уже в главном потоке.

LRESULT CALLBACK WindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

MessagesReceiver\* msgReceiver = reinterpret\_cast<MessagesReceiver\*>(wParam);

if (msgReceiver != nullptr)

{

switch(message)

{

case WM\_PROCESS\_STATE\_MSG:

msgReceiver->ProcessStateMessage();

return 0;

case WM\_PROCESS\_CHAT\_MSG:

msgReceiver->ProcessChatMessage();

return 0;

}

}

return DefWindowProc(hWnd, message, wParam, lParam);

}

**Листинг 3.6 –** **Функция обработки сообщений скрытым окном  
в главном потоке**

Для парсинга принятых xml данных используется библиотека с открытым исходным кодом PugiXml. Данная библиотека позволяет быстро и безопасно как создать, так и извлечь данные из xml строк и документов. В листинге 3.7 приведена функция для извлечения сообщения из принятых по сети данных.

ChatMessagePtr ChatMessagesHandler::GetMsgDataFromXml(const std::wstring& xmlMsg)

{

ChatMessagePtr chatMsg(nullptr);

pugi::xml\_document messageDoc;

pugi::xml\_parse\_result result = messageDoc.load(xmlMsg.c\_str());

if (result)

{

pugi::xml\_node messageNode = messageDoc.first\_child();

pugi::xml\_node userNode = messageNode.child(USER\_NODE);

std::wstring uuid = userNode.child\_value(USER\_UUID\_NODE);

pugi::xml\_node dataNode = messageNode.child(DATA\_NODE);

std::wstring message = dataNode.text().as\_string();

chatMsg.reset(new ChatMessage(uuid, message));

}

return chatMsg;

}

**Листинг 3.7 –** **Использования библиотеки PugiXml   
для извлечения данных сообщения**

**3.2.4 Разработка модуля отправки и получения файлов**

При отправке файлов по сети используется идея разбиения файлов на части и передача каждой поочередно. Механизм отправки и получения является асинхронным для обеспечения отзывчивости главного (UI) потока. Все действия инкапсулируются в классе FileTransferManager и в других вспомогательных объектах. Для передачи используется протокол TCP и библиотека ACE (листинг 3.8). Каждое файловое сообщение имеет заголовок, в котором отображается информация об отправителе, размере и имени файла. После получения заголовка файла осуществляется непосредственная его передача. Сохранение файла осуществляется в каталог указанный в настройках программного средства.

void FileTransferManager::SendFile(const std::wstring& uuid, const std::wstring& filePath,

ui::IProgressUIObserver\* observer)

{

if (net::NetUsersManager::Instance()->IsUserExist(uuid))

{

FileInfoPtr fileInfo(GetFileInfo(filePath));

if (fileInfo)

{

std::wstring addr(net::NetUsersManager::Instance()->GetNetUserAddress(uuid));

ACE\_INET\_Addr userAddr(m\_settingsHolder->GetPort(), addr.c\_str());

AsyncFileSender sender(userAddr, fileInfo, uuid, observer); //runs and detaches thread

}

}

}

**Листинг 3.8 – Метод SendFile класса FileTransferManager**

**для асинхронной передачи файлов**

**4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**4.1 Выбор методики тестирования**

Цель испытания программного средства «Сетевой чат» является устранение возможных ошибок и неполадок в работе программного обеспечения. Задачей тестирования приложения было обеспечение высокого качество программного средства. Качество программного обеспечения (ПО) можно определить как совокупную характеристику исследуемого ПО с учётом таких составляющих как: надёжность, сопровождаемость, практичность, эффективность, мобильность, функциональность.

Одним из наиболее важных видов тестирования является функциональное. Функциональное тестирование – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности приложения в определённых условиях решать задачи, нужные пользователям. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает. Был разработан набор функциональных тест-кейсов, сгруппированных в сценарии в соответствии с ролями пользователей системы.

Для тестирования качества и надежности программного кода было использовано юнит-тестирование или тестирование методом «белого ящика». Оно предполагает процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Целью обеспечения модульного тестирования было изолирование отдельных частей программы и демонстрация, что по отдельности эти части работоспособны.

**4.2 Тестирование программного средства**

Для тестирования основных функциональных требований программного средство были разработаны набор тест-кейсов. Они охватывают функции, которые непосредственно предоставляются пользователю после запуска приложения.

Таблица 4.1 – Пример функциональных тест-кейсов приложения

|  |  |  |  |
| --- | --- | --- | --- |
| Модуль | Описание теста | Ожидаемый результат | Статус |
| 1 | 2 | 3 | 4 |
| Главное окно | Запуск приложения в операционной среде Windows. | Открытие главного окна приложения, появление диалогового окна регистрации нового пользователя. | выполнено успешно |
| Логин-модуль | Запуск приложения | Открытие главного окна приложения появление диалогового окна логина нового пользователя. | выполнено успешно |
| Ввод имени и пароля | Логин успешно выполнен | выполнено успешно |
| Логин-модуль | Запуск приложения | Открытие главного окна приложения, появление диалогового окна логина нового пользователя | выполнено успешно |
| Ввод неверного имени и пароля | В логине отказано | выполнено успешно |
| Регистрация | Запуск приложения | Открытие главного окна приложения, появление диалогового окна логина нового пользователя. | выполнено успешно |
| Ввод имени и пароля | Регистрация успешна | выполнено успешно |
| Логин-модуль | Запуск приложения | Открытие главного окна приложения, появление диалогового окна логина нового пользователя. | выполнено успешно |
| Ввод пустого имени и/или пароля | В регистрации отказано | выполнено успешно |
| UI  модуль | Запуск приложения | Открытие главного окна приложения, появление диалогового окна логина нового пользователя. | выполнено успешно |
| Запуск еще одного экземпляра приложения | Открытие главного окна приложения, появление диалогового окна логина нового пользователя | выполнено успешно |
| Сетевой модуль (передача текстовых сообщений) | Запуск приложения | Открытие главного окна приложения появление диалогового окна логина нового пользователя | выполнено успешно |
| Ввод имени и пароля | Логин успешно выполнен | выполнено успешно |
| Выбор пользователя в списке контактов и отправка текстового сообщения | Сообщение доставлено успешно | выполнено успешно |
| Выбор другого пользователя в списке контактов и отправка текстового сообщения | Сообщение доставлено успешно | выполнено успешно |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Сетевой модуль (передача файлов) | Запуск приложения | Открытие главного окна приложения появление диалогового окна логина нового пользователя | выполнено успешно |
| Ввод имени и пароля | Логин успешно выполнен | выполнено успешно |
| Выбор пользователя в списке контактов, перетаскивание файла на окно ввода сообщений и отправка файла | Файл передан успешно | выполнено успешно |
| Выбор другого пользователя в списке контактов, перетаскивание файла на окно ввода сообщений и отправка файла | Файл передан успешно | выполнено успешно |

Проведенное модульное тестирование позволяет осуществлять рефакторинг существующего программного кода, гарантируя, что модуль по-прежнему работает корректно.

При проведении модульного тестирования использовались готовые автоматизированные средства юнит-тестирования из библиотеки Boost Test [4]. Данная библиотека обеспечивает подходящий набор компонентов для написания тестовых программ, организации тестов и создания, как одиночных тестов, так и их наборов, при этом максимально обеспечивая контроль их выполнение.

**5 РАСЧЕТ ЭКОНОМИЧЕСКИХ ПОКАЗАТЕЛЕЙ  
ПРОГРАММНОГО СРЕДСТВА**

**5.1 Методика расчета себестоимости программного средства**

В результате выполненной работы реализована программа для обмена сообщениями между пользователями локальной сети. Так как процесс разработки занимал определенное время, то целесообразно провести среднюю оценку затрат на реализацию программного продукта, а также провести расчеты себестоимости программного средства и его цены.

Под себестоимостью понимаются затраты, необходимые для производства продукта. Для её расчета необходимо оценить затраты времени на разработку программного продукта.

Поскольку процесс написания программ является творческим, вести нормативы для оценки затрат очень сложно, то применяется метод экспертных оценок, суть которого состоит в оценке затрат несколькими экспертами. В данном случае в качестве экспертов выступают разработчик программного продукта и руководитель проекта.

Процесс разработки программного продукта состоит из нескольких этапов:

- проектирование программного средства.

- реализация программного средства.

- тестирование и отладка.

Первый этап включает в себя анализ требований, предъявляемых к программе, поиск необходимой информации в интернете и других источниках, составление алгоритма и др., то есть представляет собой интеллектуальный труд разработчика, который не может быть автоматизирован. Время, затрачиваемое на выполнение первого этапа, зависит от накопленных знаний и квалификации программиста.

Второй этап представляет собой реализацию разработанной программы на используемом языке программирования с учетом выбранных технологий и средств разработки.

Процесс отладки необходим для нахождения ошибок, не найденных на предыдущих этапах разработки, а процесс тестирования позволяет, с достаточной вероятностью, удостовериться в том, что программный продукт удовлетворяет требованиям, выработанным для него на этапе проектирования.

**5.2 Расчет себестоимости программного средства**

На основе экспертных оценок затрат времени определяется средняя величина для каждого из вышеуказанных этапов по формуле (5.1):

|  |  |
| --- | --- |
| , | (5.1) |

где  – средняя оценка,

 – оценка руководителя проекта,

 – оценка разработчика проекта.

Экспертные оценки времени рассчитываются по каждому этапу разработки программного продукта для трех ситуаций: наименее возможная величина затрат (), наиболее вероятная величина затрат (), наиболее возможная величина затрат ().

Результаты расчета средней оценки времени на разработку программного продукта приведены в таблице 6.1.

Таблица 5.1 – Оценка затрат времени на разработку программного продукта

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Этапы разработки программного продукта | Величина затрат | | | | | | | | |
| Наименее возможная , дни | | | Наиболее вероятная , дни | | | Наиболее возможная , дни | | |
|  |  |  |  |  |  |  |  |  |
| Проектирование | 10 | 12 | 11 | 14 | 16 | 15 | 20 | 24 | 22 |
| Написание программы | 6 | 10 | 8 | 10 | 12 | 11 | 12 | 14 | 13 |
| Тестирование и  отладка | 10 | 12 | 11 | 14 | 16 | 15 | 20 | 22 | 21 |
| Итого | 26 | 34 | 30 | 38 | 44 | 41 | 52 | 60 | 56 |

На основании таких оценок рассчитываются математическое ожидание и отклонение, по каждому этапу разработки комплекса программных продуктов.

По формуле (5.2) рассчитываем математическое ожидание:

|  |  |
| --- | --- |
| , | (5.2) |

где  – математическое ожидание для  – го этапа,

, ,  – средние значения затрат.

Стандартное отклонение для каждого этапа разработки программного продукта определяется по формуле (5.3):

|  |  |
| --- | --- |
| , | (5.3) |

где  – стандартное отклонение для  – го этапа.

Общая оценка затрат на разработку программного продукта рассчитывается по формуле (5.4):

|  |  |
| --- | --- |
| , | (5.4) |

где  – общая оценка затрат,

 – математическое ожидание затрат для  – го этапа,

 – количество этапов разработки программного продукта.

Стандартное отклонение в целом по расчету себестоимости программного средства вычисляется по формуле (5.5):

|  |  |
| --- | --- |
| , | (5.5) |

где  – стандартное отклонение общей оценки затрат;

 – стандартное отклонение оценки затрат для  – го этапа.

На основе расчетов математического ожидания и стандартного отклонения вычисляется коэффициент вариации. Данный коэффициент показывает, насколько согласованы эксперты. Ниже приведены формулы (5.6) для расчета коэффициента вариации для  – го этапа и формула (5.7) для вычисления общего коэффициента вариации:

|  |  |
| --- | --- |
| , | (5.6) |

где  – коэффициент вариации для  – го этапа;

 – стандартное отклонение оценки затрат  – го этапа;

 – математическое ожидание затрат для  – го этапа.

|  |  |
| --- | --- |
| , | (5.7) |

где – коэффициент вариации по всем этапам;

 – стандартное отклонение оценки затрат по всем этапам;

 – математическое ожидание затрат по всем этапам.

Результаты расчета , , , а также , , приведены в табли-  
це (5.2).

Таблица 5.2 – Затраты времени на разработку программного продукта

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Этапы разработки программного продукта | Средняя величина затрат  времени по этапам | | | Математич. ожидание  , дни | Стандарт. отклонение  , дни | Коэфф. вариации  , дни |
| Наименее возможная , дни | Наиболее вероятная , дни | Наиболее возможная , дни |
| Проектирование | 11 | 15 | 22 | 15,50 | 1,83 | 0,04 |
| Написание программы | 8 | 11 | 13 | 10,83 | 0,83 | 0,08 |
| Тестирование и отладка | 11 | 15 | 21 | 15,33 | 1,67 | 0,11 |
| Итого | 30 | 41 | 56 | 41,66 | 4,33 | 0,10 |

Так как коэффициент вариации по программному продукту равен 0,10 и не превосходит 0,33, то мнения экспертов согласованы.

Расчет себестоимости программного продукта производится по формуле (5.8):

|  |  |
| --- | --- |
| , | (5.8) |

где С – себестоимость комплекса программных продуктов,

 – среднемесячная заработная плата программиста с учетом районного коэффициента,

 – количество рабочих дней в месяце,

 – трудозатраты на проектирование,

 – трудозатраты на написание программы,

 – трудозатраты на тестирование и отладку,

 – коэффициент, учитывающий страховые взносы,

 – коэффициент, учитывающий накладные расходы,

 – стоимость одного часа машинного времени,

 – стоимость одного часа работы в интернете,

 – количество дней работы на компьютере,

 – количество дней работы в интернете.

Значения данных для расчета себестоимости программного продукта представлены в таблице 5.3.

Таблица 5.3. Значения данных для расчета себестоимости.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Показатель | , тыс.руб | , дни | ,  дни | ,  дни | ,  дни | ,  % | ,  % | , тыс.руб | , дни | , дни |
| Значение | 9000 | 22 | 15,50 | 10,83 | 15,33 | 34 | 8 | 1,25 | 44 | 14 |

Рассчитаем стоимость одного часа машинного времени. Для этого рассчитаем затраты на эксплуатацию персонального компьютера за год по формуле (5.9):

|  |  |
| --- | --- |
| , | (5.9) |

где  – себестоимость одного часа машинного времени;

 – затраты на электроэнергию за год работы;

 – затраты на амортизационные отчисления;

 – затраты на комплектующие материалы;

 – прочие затраты;

 – общее время работы компьютера за год.

Приведем формулы для расчета вышеперечисленных видов затрат. Общее время работы компьютера за год примем 1987 часов. Затраты на комплектующие материалы за год примем 700 тыс. рублей. Затраты на электроэнергию зависят от тарифа на электроэнергию и потребляемой мощности персонального компьютера, а также от времени работы персонального компьютера. Формула (5.10) для расчета затрат на электроэнергию:

|  |  |
| --- | --- |
| , | (5.10) |

где  – затраты на электроэнергию за год работы,

 – общее время работы компьютера за год,

 – тариф электроэнергии за 1кВт,

 – потребляемая мощность ПК по паспортным данным.

Примем потребляемую мощность персонального компьютера и тариф на электроэнергии соответственно:

 = 0,065 кВт./час.

 = 842 руб./кВт.

По формуле (5.10) находим затраты на электроэнергию в год:

 руб.

Амортизационные отчисления в год рассчитываются по формуле (5.11):

|  |  |
| --- | --- |
| , | (5.11) |

где  – затраты на амортизационные отчисления,

– стоимость персонального компьютера (5 055 760 руб.),

 – процент отчисления на амортизацию.

В рамках дипломного проекта установим срок полезного использования компьютера 2 года. Определим процент отчисления на амортизацию:



Произведем расчет амортизационных средств по формуле (5.11):

 руб./год.

Прочие расходы составляют 5% от суммы затрат. Они рассчитываются по формуле (5.12):

|  |  |
| --- | --- |
| , | (5.12) |

По формуле (5.12) находим:

руб./год.

Зная все данные, подсчитаем стоимость одного часа машинного времени по формуле (5.9):

 руб.

Рассчитаем себестоимость программного обеспечения по формуле (5.8):



Цена программного продукта рассчитывается по формуле (5.13):

|  |  |
| --- | --- |
| , | (5.13) |

где  – цена программного продукта,

 – себестоимость программного продукта,

 – рентабельность программного продукта.

Рентабельность программного продукта примем за  = 20% = 0,2. По формуле (5.13) находим:

 руб.

Цена программного продукта с учетом НДС рассчитывается по формуле (5.14):

|  |  |
| --- | --- |
| , | (5.14) |

где  – цена программного продукта с НДС,

 – цена программного продукта,

 – составляет 20% от цены программного продукта.

По формуле (5.14) вычисляется цена программного продукта с учетом НДС:

 руб.

Итого, стоимость данного программного продукта на рынке будет составлять 35 721 160 руб.

**5.3 Анализ эффективности внедрения программного средства**

Внедрение разработанного программного средства может обеспечить пользователю ожидаемый прирост прибыли за счет сокращения трудоемкости решения задачи, являющейся предметом автоматизации и, как результат, снижения текущих затрат, связанных с решением данной задачи.

Основным фактором, обеспечивающим эффективность внедрения приложения, является отсутствие необходимости использования для коммуникации телефонной и факсимильной связи. При этом сохраненная история переданных данных поможет в составлении отчетных данных.

Возможность быстрой отправки сообщений всем пользователям сети позволяет более четко организовать рабочий процесс.

Так же нужно заметить, что использование разработанного программного средства «Сетевой чат» требует наличие только операционной системы Windows и организации локальной вычислительной сети. Активного подключения к сети Internet не требуется. Что по сравнению с другими аналогами программ, предназначенных для общения в сети, делает внедрение разработанного программного средства еще более эффективным.

Таким образом, по результатам оценки экономической эффективности проектирования и внедрения средства «Сетевой чат» сразу можно сказать, что это выгодно. Хоть выгода и косвенная, но, как правило, заметная в средне и долгосрочной перспективе.

Внедрение средств автоматизации передачи информации может привести к корректированию самого бизнес-процесса, так как информацию сотрудники получают быстрее, что увеличивает скорость выполнения различных задач. Сотрудники могут обрабатывать большие объемы информации за свое рабочее время, что можно использовать или для уменьшения затрат на персонал или для быстрого развития бизнеса при неизменности количества сотрудников, занятых обработкой информации.

**6 ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ   
ПРИ РАБОТЕ НА ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ**

**6.1 Общие требования безопасности**

К работе с персональным компьютером (далее ПК) допускаются работники, не имеющие медицинских противопоказаний, прошедшие инструктаж по вопросам охраны труда, с группой по электробезопасности не ниже 1.

Женщины со времени установления беременности и в период кормления грудью к выполнению всех видов работ, связанных с использованием ПК, не допускаются.

При работе с ПК на работников могут оказывать неблагоприятное воздействие следующие опасные и вредные производственные факторы:

- повышенный уровень электромагнитных излучений;

- повышенный уровень ионизирующих излучений;

- повышенный уровень статического электричества;

- повышенная напряженность электростатического поля;

- повышенная или пониженная ионизация воздуха;

- повышенная яркость света;

- прямая и отраженная блесткость;

- повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека;

- статические перегрузки костно-мышечного аппарата и динамические локальные перегрузки мышц кистей рук;

- перенапряжение зрительного анализатора;

- умственное перенапряжение;

- эмоциональные перегрузки;

- монотонность труда.

В зависимости от условий труда, в которых применяются ПК, и характера работы на работников могут воздействовать также другие опасные и вредные производственные факторы.

Организация рабочего места с ПК должна учитывать требования безопасности, удобство положения, движений и действий работника.

Рабочий стол с учетом характера выполняемой работы должен иметь достаточный размер для рационального размещения монитора (дисплея), клавиатуры, другого используемого оборудования и документов, поверхность, обладающую низкой отражающей способностью.

Клавиатура располагается на поверхности стола таким образом, чтобы пространство перед клавиатурой было достаточным для опоры рук работника (на расстоянии не менее чем 300 мм от края, обращенного к работнику).

Чтобы обеспечивалось удобство зрительного наблюдения, быстрое и точное считывание информации, плоскость экрана монитора располагается ниже уровня глаз работника предпочтительно перпендикулярно к нормальной линии взгляда работника (нормальная линия взгляда - 15° вниз от горизонтали).

Для исключения воздействия повышенных уровней электромагнитных излучений расстояние между экраном монитора и работником должно составлять не менее 500 мм (оптимальное 600-700 мм).

Применяемые подвижные подставки для документов (пюпитры) размещаются в одной плоскости и на одной высоте с экраном.

Рабочий стул (кресло) должен быть устойчивым, место сидения должно регулироваться по высоте, а спинка сиденья - по высоте, углам наклона, а также расстоянию спинки от переднего края сиденья. Регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию.

Для тех, кому это удобно, предусматривается подставка для ног.

Рабочее место размещается таким образом, чтобы естественный свет падал сбоку (желательно слева).

Для снижения яркости в поле зрения при естественном освещении применяются регулируемые жалюзи, плотные шторы.

Светильники общего и местного освещения должны создавать нормальные условия освещенности и соответствующий контраст между экраном и окружающей обстановкой с учетом вида работы и требований видимости со стороны работника. Освещенность на поверхности стола в зоне размещения рабочего документа должна составлять 300-500 люкс.

Возможные мешающие отражения и отблески на экране монитора и другом оборудовании устраняются путем соответствующего размещения экрана, оборудования, расположения светильников местного освещения.

При рядном размещении рабочих столов расположение экранов видеомониторов навстречу друг другу из-за их взаимного отражения не допускается.

Для обеспечения безопасности работников на соседних рабочих местах расстояние между рабочими столами с мониторами (в направление тыла поверхности одного монитора и экрана другого монитора) должно быть не менее 2,0 м, а расстояние между боковыми поверхностями мониторов - не менее 1,2 м.

Для снижения уровня напряженности электростатического поля при необходимости применяются экранные защитные фильтры. При эксплуатации защитный фильтр должен быть плотно установлен на экране монитора и заземлен.

Для обеспечения оптимальных параметров микроклимата проводятся регулярное в течение дня проветривание и ежедневная влажная уборка помещений, используются увлажнители воздуха.

При работе с ПК обеспечивается доступ работников к первичным средствам пожаротушения, аптечкам первой медицинской помощи.

Работники при работе с ПК с учетом воздействующих на них опасных и вредных производственных факторов обеспечиваются средствами индивидуальной защиты в соответствии с типовыми отраслевыми нормами для соответствующих профессий и должностей.

При работе с ПК работники обязаны:

- соблюдать режим труда и отдыха, установленный законодательством, правилами внутреннего трудового распорядка организации, трудовую дисциплину, выполнять требования охраны труда, правил личной гигиены;

- выполнять требования пожарной безопасности, знать порядок действий при пожаре, уметь применять первичные средства пожаротушения;

- курить только в специально предназначенных для курения местах;

- знать приемы оказания первой помощи при несчастных случаях на производстве;

- о неисправности оборудования и других замечаниях по работе с ПК сообщать непосредственному руководителю или лицам, осуществляющим техническое обслуживание оборудования.

Не допускается:

- выполнять работу, находясь в состоянии алкогольного опьянения либо в состоянии, вызванном употреблением наркотических средств, психотропных или токсических веществ, а также распивать спиртные напитки, употреблять наркотические средства, психотропные или токсические вещества на рабочем месте или в рабочее время;

- устанавливать системный блок в закрытых объемах мебели, непосредственно на полу;

- использовать для подключения ПК розетки, удлинители, не оснащенные заземляющим контактом (шиной).

**6.2 Требования безопасности во время работы на ПК**

**6.2.1 Требования безопасности перед началом работы**

Перед началом работы с ПК работник обязан:

1. Проветрить рабочее помещение;

2. Проверить:

- устойчивость положения оборудования на рабочем столе;

- отсутствие видимых повреждений оборудования, дискет в дисководе системного блока;

- исправность и целостность питающих и соединительных кабелей, разъемов и штепсельных соединений, защитного заземления (зануления);

- исправность мебели.

3. Отрегулировать:

- положение стола, стула (кресла), подставки для ног, клавиатуры, экрана монитора;

- освещенность на рабочем месте. При необходимости включить местное освещение;

4. Протереть поверхность экрана монитора, защитного фильтра (при его наличии) сухой мягкой тканевой салфеткой;

5. Убедиться в отсутствии отражений на экране монитора, встречного светового потока;

6. Включить оборудование ПК в электрическую сеть, соблюдая следующую последовательность: стабилизатор напряжения (если он используется), блок бесперебойного питания, периферийные устройства (принтер, монитор, сканер и другие устройства), системный блок.

Запрещается приступать к работе при:

- выраженном дрожании изображения на мониторе;

- обнаружении неисправности оборудования;

- наличии поврежденных кабелей или проводов, разъемов, штепсельных соединений;

- отсутствии или неисправности защитного заземления (зануления) оборудования.

**6.2.2 Требования безопасности при выполнении работ**

Во время работы с ПК работник обязан:

- соблюдать требования охраны труда, установленные настоящей Инструкцией;

- содержать в порядке и чистоте свое рабочее место;

- держать открытыми вентиляционные отверстия оборудования;

- соблюдать оптимальное расстояние от экрана монитора до глаз.

Работу за экраном монитора следует периодически прерывать на регламентированные перерывы, которые устанавливаются для обеспечения работоспособности и сохранения здоровья, или заменять другой работой с целью сокращения рабочей нагрузки у экрана.

Время регламентированных перерывов в течение рабочего дня (смены) устанавливается в зависимости от его (ее) продолжительности, вида и категории трудовой деятельности.

При 8-часовой рабочей смене и работе с ПК регламентированные перерывы устанавливаются:

- для I категории работ через 2 часа от начала рабочей смены и через 2 часа после обеденного перерыва продолжительностью 15 минут каждый;

- для II категории работ через 2 часа от начала рабочей смены и через 1,5-2 часа после обеденного перерыва продолжительностью 15 минут каждый или продолжительностью 10 минут каждый час работы;

- для III категории работ через 1,5-2 часа от начала рабочей смены и через 1,5-2 часа после обеденного перерыва продолжительностью 20 минут каждый или продолжительностью 15 минут через каждый час работы.

Продолжительность непрерывной работы с ПК без регламентированного перерыва не должна превышать 2 часов.

Во время регламентированных перерывов для снижения нервно-эмоционального напряжения, утомления зрительного анализатора, улучшения функционального состояния, нервной, сердечно-сосудистой, дыхательной систем, а также мышц плечевого пояса, рук, спины, шеи и ног целесообразно выполнять комплексы упражнений.

Работникам с высоким уровнем напряженности труда во время регламентированных перерывов и в конце рабочего дня показана психологическая разгрузка в специально оборудованных комнатах психологической разгрузки.

С целью уменьшения отрицательного влияния монотонности необходимо применять чередование операций.

При работе с текстовой информацией следует отдавать предпочтение физиологически наиболее оптимальному режиму представления черных символов на белом фоне.

Не следует оставлять оборудование включенным без наблюдения. При необходимости прекращения на некоторое время работы корректно закрываются все активные задачи и оборудование выключается.

При работе с ПК не разрешается:

- при включенном питании прикасаться к панелям с разъемами оборудования, разъемами питающих и соединительных кабелей, экрану монитора;

- загромождать верхние панели оборудования, рабочее место бумагами, посторонними предметами;

- производить переключения, отключение питания во время выполнения активной задачи;

- допускать попадание влаги на поверхность оборудования;

- включать сильно охлажденное (принесенное с улицы в зимнее время) оборудование;

- производить самостоятельно вскрытие и ремонт оборудования;

- вытирать пыль на включенном оборудовании;

- допускать нахождение вблизи оборудования посторонних лиц.

**6.2.3 Требования безопасности по окончании работы**

По окончании работы работник обязан:

- корректно закрыть все активные задачи;

- при наличии дискеты в дисководе извлечь ее;

- выключить питание системного блока;

- выключить питание всех периферийных устройств;

- отключить блок бесперебойного питания;

- отключить стабилизатор напряжения (если он используется);

- отключить питающий кабель от сети;

- осмотреть и привести в порядок рабочее место;

- о неисправности оборудования и других замечаний по работе с ПК сообщить непосредственному руководителю или лицам, осуществляющим техническое обслуживание оборудования;

- при необходимости вымыть с мылом руки.

**6.3 Требования безопасности в аварийных ситуациях**

В аварийных (экстремальных) ситуациях необходимо:

1. При повреждении оборудования, кабелей, проводов, неисправности заземления, появлении запаха гари, возникновении необычного шума и других неисправностях немедленно отключить электропитание оборудования и сообщить о случившемся непосредственному руководителю и лицу, осуществляющему техническое обслуживание оборудования;

2. В случае сбоя в работе оборудования ПК или программного обеспечения вызвать специалиста организации, осуществляющего техническое обслуживание данного оборудования, для устранения неполадок;

3. При возгорании электропроводки, оборудования и тому подобных происшествиях отключить электропитание и принять меры по тушению пожара с помощью имеющихся первичных средств пожаротушения, сообщить о происшедшем непосредственному руководителю. Применение воды и пенных огнетушителей для тушения находящегося под напряжением электрооборудования недопустимо. Для этих целей используются углекислотные огнетушители.

При несчастном случае на производстве необходимо:

- быстро применять меры по предотвращению воздействия на потерпевшего травмирующих факторов, оказанию потерпевшему первой помощи, вызову на место происшествия медицинских работников или доставке потерпевшего в организацию здравоохранения;

- сообщить о происшествии руководителю.

**6.4 Время регламентированных перерывов в течение рабочего дня**

Установлены следующие виды трудовой деятельности:

группа А — работа по считыванию информации с экрана ПК с предварительным запросом;

группа Б — работа по вводу информации;

группа В — творческая работа в режиме диалога с ПК.

При выполнении в течение рабочего дня работ, относящихся к разным группам, за основную работу с ПК следует принимать такую, которая занимает не менее 50% времени в течение рабочего дня (смены). Время регламентированных перерывов в течение рабочего дня приведено в таблице 6.1.

Таблица 6.1 – Время регламентированных перерывов в течение рабочего дня

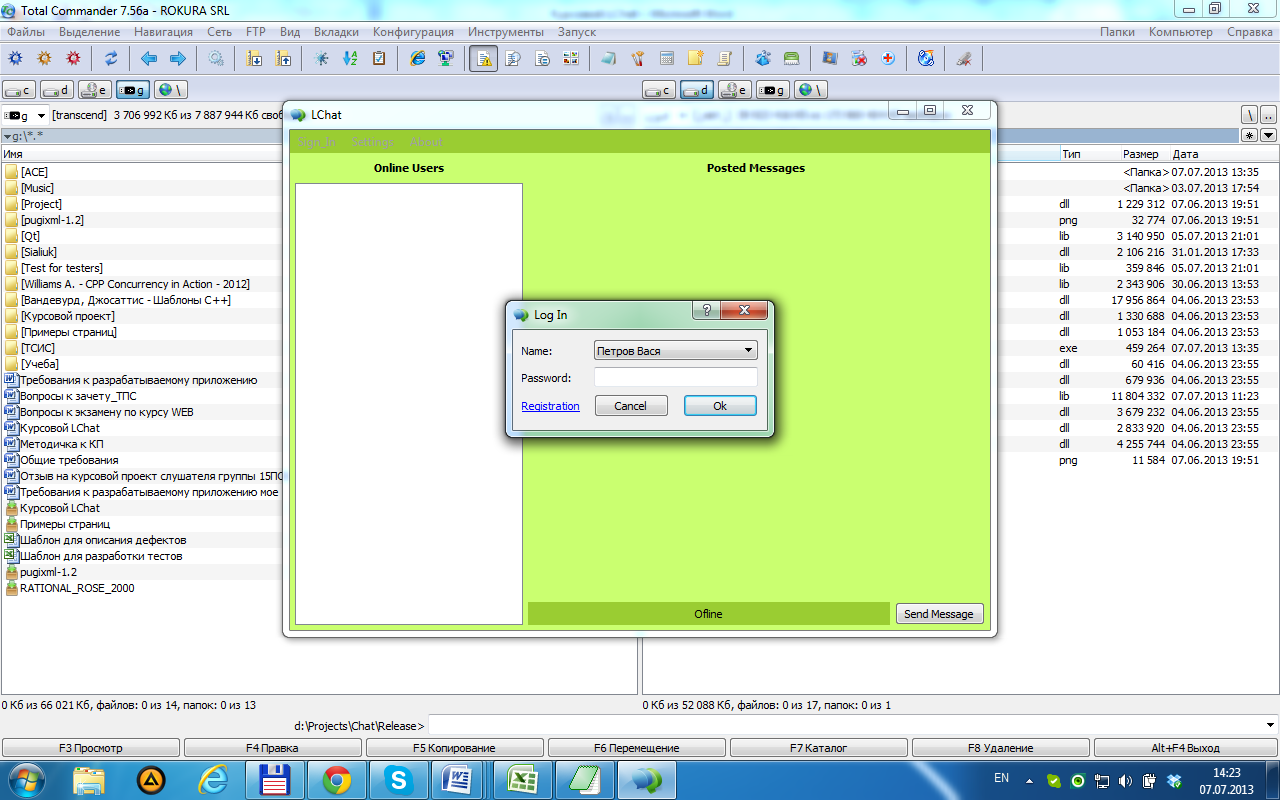
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Категория работы с ПК | Уровень нагрузки за рабочую смену при видах работ с ПК | | | Суммарное время регламентированных перерывов, минут | |
| группа А,  количество знаков | группа Б,  количество знаков | группа В, часов | при  8-часовой смене | при  12-часовой смене |
| I | до 20000 | до 15000 | до 2,0 | 30 | 70 |
| II | до 40000 | до 30000 | дo 4,0 | 50 | 90 |
| III | до 60000 | до 40000 | до 6,0 | 70 | 120 |

**7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Программа, разработанная в данном дипломном проекте, является сетевым чатом и предназначена для общения пользователей локальной сети в режиме реального времени при помощи обмена короткими текстовыми сообщениями, передачи файлов размером до 512 Mb.

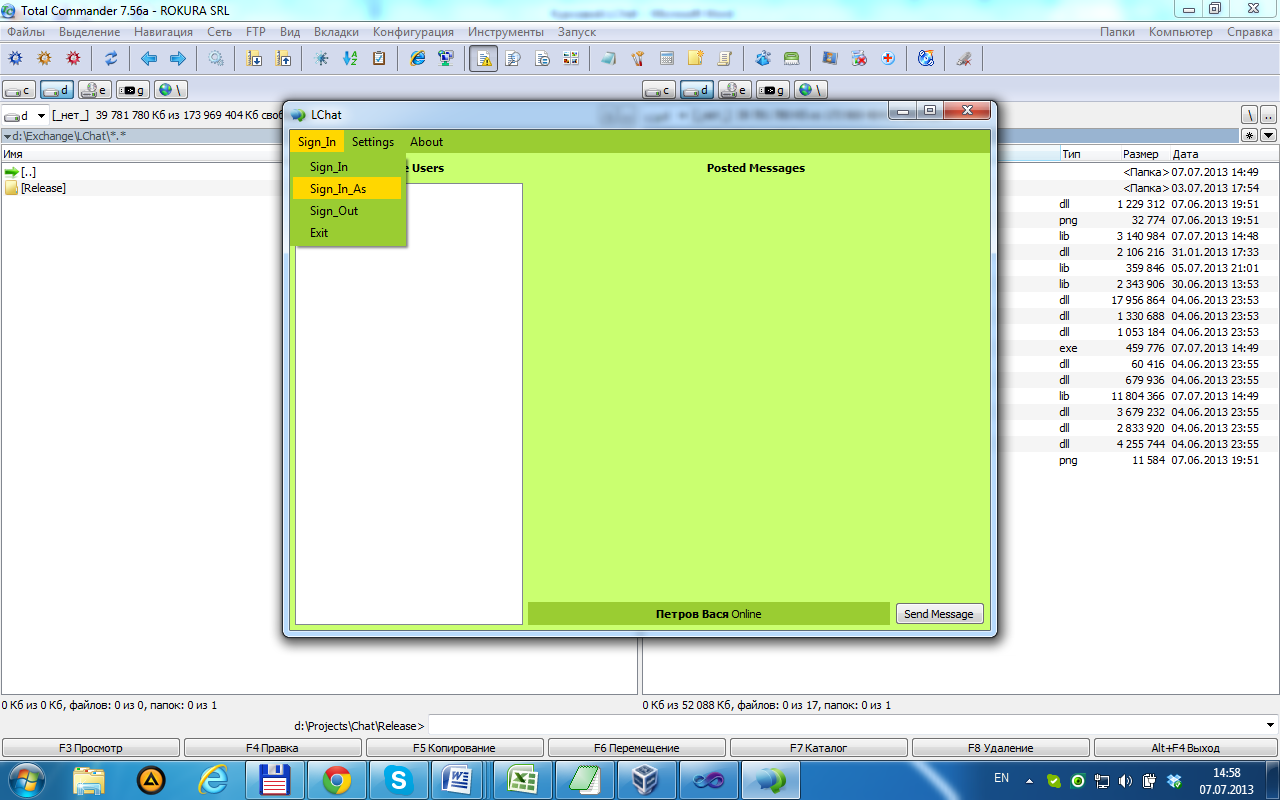
Предусмотрена возможность восстановления переданных сообщений. Для этого при подключении данного пользователя в верху окна в котором отображаются переданные сообщения появиться ссылка Last Conversation. При нажатии на которой произойдет добавление предыдущих сообщений вверх окна.

После запуска приложения появляется главное окно и диалог входа в систему. Если приложение запущено впервые на компьютере появляется диалог регистрации пользователей. Главное окно приложения представлено на рисунке 7.1.

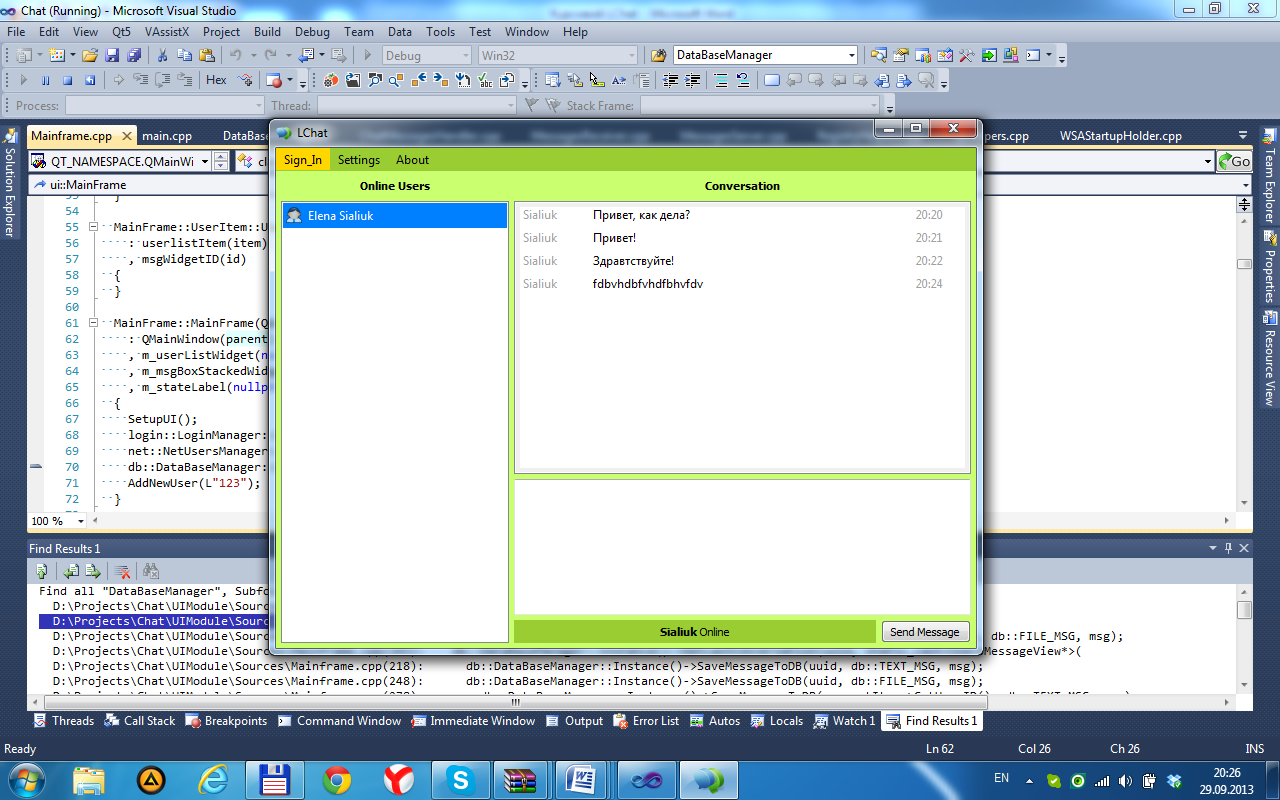


**Рисунок 7.1 –** **Главное окно приложения «Сетевой чат»**

С помощью главного меню пользователь может закрыть приложения, сменить пользователя, изменить настройки программы. Главное меню программного средства представлено на рисунке 7.2.



**Рисунок 7.2 –** **Главное меню приложения «Сетевой чат»**



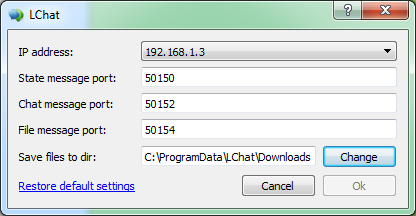
**Рисунок 7.3 – Главное окно приложения «Сетевой чат»   
после авторизации**

После успешного логина или регистрации главное окно активизируется, и пользователь переходит в режим онлайн (рисунок 7.3). Если в сети есть подключенные пользователи, они отобразятся в списке контактов.

Для отправки сообщения достаточно выбрать нужного пользователя в списке контактов набрать сообщение и нажать кнопку «Send Messаge».

Для отправки файлов необходимо перетянуть файл на область для ввода новых сообщений. После этого сразу же начнется передача данного файла.

Диалог настроек приложения позволяет сменить текущий IP адрес, и номера портов, и место сохранения переданных файлов в случае необходимости. Диалог настроек представлен на рисунке 7.4.



**Рисунок 7.4 –** **Диалог настроек** **приложения «Сетевой чат»**

Программное средство предназначено для работы в операционных системах Windows XP, Windows Vista, Windows 7, и имеет следующие требования к оборудованию:

- CPU AMD Duron 1000 Mhz;

- оперативная память 256 Mb;

- 50 Mb свободного места на жестком диске;

- сетевой адаптер Ethernet;

- подключение к локальной сети.

**ЗАКЛЮЧЕНИЕ**

В данной работе было разработано клиент-серверное программное средство «Сетевой чат» с возможностью обмена короткими текстовыми сообщениями, передачи файлов, сохранением переданных сообщений в базе данных.

Для разработки приложение использовалась интегрированная среда разработки Visual Studio. Данная система упрощает создание, отладку и развертывание приложений на различных платформах, включает встроенную поддержку модели «разработка через тестирование», а также инструменты отладки, которые обеспечивают создание высококачественных решений.

В качестве вспомогательных средств разработки использовались свободно распространяемые библиотеки на языке C++ такие как, Qt, ACE, SQLite, PugiXml, BOOST.

В ходе проектирования приложения было принято решение о реализации приложения в виде одной программы, выполняющей роль, как клиента, так и сервера. Был разработан примерный алгоритм работы каждого модуля приложения. Заключительным этапом работы явилась непосредственная реализация модуля отправки и получения сообщений и файлов.

В данный момент на рынке программного обеспечения очень большое количество программ для обмена сообщениями между пользователями, развитие которых не стоит на месте. Можно перечислить некоторые: ICQ, IRC, Skype, Jabber и др.

Если сравнивать программное средство, которое разработан в данном дипломном проекте, то можно выявить, что оно в чем-то уступает, а в чем-то превосходит гигантов рынка.

Например, ICQ и Skype требует активного подключения к интернету, в разработанном приложении пользователь может авторизоваться и работать в локальной сети без использования Интернета. Также клиенту не приходится задумываться о размещении серверной части, т.к. «Сетевой чат» выполняет роль и клиента и сервера.

У ICQ, Skype, IRC очень много разнообразных клиентов, в том числе и кроссплатформенных, что неоспоримо является плюсом данных проектов, наше приложение разработано для платформ операционной системы Windows. В тоже время, использование кроссплатформенных библиотек позволит в дальнейшем легко портировать «Сетевой чат» на другие платформы.

Все перечисленные проекты не стоят на месте, развиваются, и это требует немалых средств при развития и в жесткой конкуренции, ко многим добавляют совсем ненужные функции и возможности, «Сетевой чат» содержит только необходимые функции и обладает малыми, но емкими возможностями, что позволяет существенно снизить затраты на разработку, сопровождение и обслуживание.

После сравнения разработанного программного средства с гигантами рынка, можно сделать вывод, что данный проект будет выгоден как для малых предприятий, так и для больших. Причем расходы будут минимальными, что существенно повышает вероятность выбора этого продукта для использования.

Полученный программный продукт работоспособен, имеет необходимую функциональность – возможность передачи текстовых сообщений и файлов между двумя компьютерами, подключенными к локальной вычислительной сети, а также может сохранять и восстанавливать уже переданные сообщения.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]; Питер; под общ. ред. Э. Гамма. – СПб, 2007. – 366 с.

2. Рихтер, Дж. Windows для профессионалов. Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии Windows / Дж. Рихтер. – СПб: Питер, 2001. – 752 с.

3. Шмидт, Д. Программирование сетевых приложений на С++ / Д. Шмидт, С. Хьюстон. – Москва: Бином-Пресс, 2003. – 304 с.

4. Boost C++ Libraries [Electronic resource]. – 2004. – Mode of access: http://www.boost.org. – Date of access: 25.06.2013.

5. Qt Project [Electronic resource]. – 2013. – Mode of access: http://qt-project.org/. – Date of access: 26.06.2013.

6. The Adaptive Communication Environment (ACE) [Electronic resource]. – 2003. – Mode of access: <http://www.cs.wustl.edu/~schmidt/ACE.html>. – Date of access: 20.06.2013.

7. SQLite Project [Electronic resource]. – 2004. – Mode of access: <http://sqlite.org/>. - Date of access: 30.08.2013.

**ПРИЛОЖЕНИЕ А**

**Исходный код класса UDP Server**

UDPMessageServer::UDPMessageServer(IMessagesHandler\* handler,  
 IServerSettingsHolder\* settingsHolder)

: m\_startupHolder(MAKEWORD(2, 2))

, m\_msgHandler(handler)

, m\_settingsHolder(settingsHolder)

, m\_shouldShutdown(false)

, m\_needReset(false)

, m\_udpSocket(nullptr, DeleteSocket)

, m\_thread(nullptr)

{

if (m\_startupHolder.GetErrorCode() == 0)

{

Initialize();

}

}

UDPMessageServer::~UDPMessageServer()

{

m\_thread->join();

}

void UDPMessageServer::Run()

{

while (!ShouldShutdown())

{

if (login::LoginManager::Instance()->IsOnline())

{

ACE\_INET\_Addr userAddr;

iovec buffer = {0};

size\_t result = m\_udpSocket->recv(&buffer, userAddr, 0,  
 &TIMEOUT);

if ((result > 0) && buffer.iov\_base)

{

std::wstring message((wchar\_t\*)buffer.iov\_base,  
 (wchar\_t\*)(buffer.iov\_base + buffer.iov\_len));

if (!message.empty())

{

m\_msgHandler->HandleMessage(message, userAddr);

}

delete[] buffer.iov\_base;

}

}

else

{

WaitForEvents();

}

if (NeedReset())

{

InitSocket();

}

}

}

void UDPMessageServer::Shutdown()

{

{

Lock lock(m\_mutex);

m\_shouldShutdown = true;

}

m\_condVariable.notify\_one();

}

void UDPMessageServer::Reset()

{

Lock lock(m\_mutex);

m\_needReset = true;

}

void UDPMessageServer::Notify()

{

m\_condVariable.notify\_one();

}

void UDPMessageServer::Initialize()

{

InitSocket();

m\_thread.reset(new Thread(&UDPMessageServer::Run, this)); //start thread

}

void UDPMessageServer::InitSocket()

{

ACE\_INET\_Addr serverAddr(m\_settingsHolder->GetPort(),  
 m\_settingsHolder->GetAddress().c\_str());

m\_udpSocket.reset(new ACE\_SOCK\_Dgram(serverAddr));

}

bool UDPMessageServer::ShouldShutdown()

{

Lock lock(m\_mutex);

return m\_shouldShutdown;

}

bool UDPMessageServer::NeedReset()

{

Lock lock(m\_mutex);

return m\_needReset;

}

void UDPMessageServer::WaitForEvents()

{

Lock lock(m\_mutex);

m\_condVariable.wait(lock);

}

**ПРИЛОЖЕНИЕ Б**

**Исходный код класса ChatMessageManager**

ChatMessagesManager\* ChatMessagesManager::Instance()

{

static ChatMessagesManager manager;

return &manager;

}

ChatMessagesManager::ChatMessagesManager()

: m\_msgQueue(new ChatMessagesQueue())

, m\_settingsHolder(new ChatServerSettingsHolder(sm::SettingsManager::Instance()))

{

login::LoginManager::Instance()->Subscribe(this);

}

ChatMessagesManager::~ChatMessagesManager()

{

m\_server->Shutdown();

}

void ChatMessagesManager::Send(const std::wstring& uuid, const std::wstring& txtMessage)

{

net::WSAStartupHolder wsaHolder(MAKEWORD(2, 2));

if ((wsaHolder.GetErrorCode() == 0)

&& net::NetUsersManager::Instance()->IsUserExist(uuid))

{

std::wstring message(CreateMessage(txtMessage));

std::wstring addr(net::NetUsersManager::Instance()->GetNetUserAddress(uuid));

ACE\_INET\_Addr userAddr(m\_settingsHolder->GetPort(), addr.c\_str());

ACE\_INET\_Addr currentAddr(TEMP\_PORT, m\_settingsHolder->GetAddress().c\_str());

ACE\_SOCK\_Dgram udpSocket(currentAddr);

udpSocket.send(message.c\_str(), message.size() \* sizeof(wchar\_t), userAddr);

udpSocket.close();

}

}

void ChatMessagesManager::Activate(MessagesReceiver\* receiver)

{

if (!m\_server)

{

m\_msgHandler.reset(new ChatMessagesHandler(receiver, m\_msgQueue.get()));

m\_server.reset(new UDPMessageServer(m\_msgHandler.get(),  
 m\_settingsHolder.get())); //init socket and run server

}

}

ChatMessagesQueue\* ChatMessagesManager::GetMessagesQueue()

{

return m\_msgQueue.get();

}

void ChatMessagesManager::ResetServer()

{

m\_server->Reset();

}

std::wstring ChatMessagesManager::CreateMessage(const std::wstring& txtMessage)

{

try

{

login::UserDataPtr user(login::LoginManager::Instance()->GetCurrentUser());

return boost::str(boost::wformat(CHAT\_MESSAGE\_FORMAT) % user->uuid   
 % txtMessage);

}

catch (const std::exception&)

{

return std::wstring();

}

}

void ChatMessagesManager::OnlineStateChanged()

{

if (login::LoginManager::Instance()->IsOnline())

{

m\_server->Notify();

}

}

**ПРИЛОЖЕНИЕ В**

**Исходный код класса AsyncFileSender**

AsyncFileSender::AsyncFileSender(const ACE\_INET\_Addr& userAddr, const FileInfoPtr& file,

const std::wstring& uuid, ui::IProgressUIObserver\* observer)

: m\_thread(&AsyncFileSender::SendFileImpl, userAddr, file, uuid, observer)

{

}

AsyncFileSender::~AsyncFileSender()

{

if (m\_thread.joinable())

{

m\_thread.detach();

}

}

void AsyncFileSender::SendFileImpl(const ACE\_INET\_Addr& userAddr, const FileInfoPtr& fileInfo,

const std::wstring& uuid, ui::IProgressUIObserver\* observer)

{

ProgressUpdater updater(observer, fileInfo->m\_size, uuid);

MessageBlockHolder message(MakeMessageBlocks(fileInfo), MessageDeleter);

net::WSAStartupHolder wsaHolder(MAKEWORD(2, 2));

if (!(wsaHolder.GetErrorCode() == 0))

return;

SocketStream streamSocket(new ACE\_SOCK\_Stream(), StreamDeleter);

ACE\_SOCK\_Connector connector;

if (connector.connect(\*streamSocket, userAddr, &TIMEOUT) == -1)

return;

try

{

SendMessageBlock(streamSocket, message.get(), updater);

}

catch (const Interrupter::Error&)

{

if (observer)

{

observer->OnError();

}

}

}

void AsyncFileSender::SendMessageBlock(const SocketStream& socket, ACE\_Message\_Block\* message,

ProgressUpdater& updater)

{

ACE\_Message\_Block\* header = message;

int count = 0;

for (auto block = message; block != nullptr; block = block->cont(), ++count)

{

size\_t messageSize = block->length();

size\_t isSended = socket->send\_n(block->rd\_ptr(), messageSize);

int eror = errno;

if (isSended == messageSize)

{

updater.Update(messageSize, header == block);

}

else

{

updater.TransferError();

}

}

updater.Finished();

}

ACE\_Message\_Block\* AsyncFileSender::MakeMessageBlocks(const FileInfoPtr& fileInfo)

{

std::string header(strhlp::WstrToStr(MakeMessageHeader(fileInfo)));

header.resize(MTU\_SIZE, '\0');

std::vector<MessageBlockPtr> blocks;

MessageBlockPtr headerBlock(new ACE\_Message\_Block(MTU\_SIZE));

size\_t headerSize = header.size();

int copyResult = headerBlock->copy(header.c\_str(), headerSize);

blocks.push\_back(std::move(headerBlock));

const size\_t MTUChunkCount = fileInfo->m\_size / MTU\_SIZE;

const size\_t residualSize = fileInfo->m\_size % MTU\_SIZE;

FileHolder file(new std::ifstream(fileInfo->m\_path, std::ios::binary), CloseFile);

if (!(\*file.get()))

return nullptr;

for (size\_t count = 0; count != MTUChunkCount; ++count)

{

MessageBlockPtr block(new ACE\_Message\_Block(MTU\_SIZE));

file->read(block->wr\_ptr(), MTU\_SIZE);

block->wr\_ptr(MTU\_SIZE);

blocks.push\_back(std::move(block));

}

if (residualSize > 0)

{

MessageBlockPtr block(new ACE\_Message\_Block(residualSize));

file->read(block->wr\_ptr(), residualSize);

block->wr\_ptr(residualSize);

blocks.push\_back(std::move(block));

}

if (\*file.get())

return LinksMessageBlocks(blocks);

return nullptr;

}

std::wstring AsyncFileSender::MakeMessageHeader(const FileInfoPtr& fileInfo)

{

return boost::str(boost::wformat(FILE\_MESSAGE\_HEADER)

% fileInfo->m\_uuid

% fileInfo->m\_size

% fileInfo->m\_name);

}

**ПРИЛОЖЕНИЕ Г**

**Исходный код класса DataBaseService**

DataBaseService::DataBaseService()

: m\_shouldShutdown(false)

{

}

DataBaseService::~DataBaseService()

{

StopJoinableThread();

}

void DataBaseService::Start()

{

StopJoinableThread();

m\_thread.reset(new boost::thread(&DataBaseService::ProcessRequest, this));

}

void DataBaseService::Stop()

{

Lock lock;

m\_shouldShutdown = true;

}

void DataBaseService::PostRequest(const DBRequestPtr& holder)

{

m\_requestQueue.Enqueue(holder);

}

void DataBaseService::ProcessRequest()

{

std::string uuid(strhlp::WstrToStr(login::LoginManager::Instance()->GetCurrentUser()->uuid).append(".db"));

DataBaseHolder dbHolder(uuid.c\_str());

if (dbHolder.Open())

{

while (!ShouldShutdown())

{

DBRequestPtr reqHolder(m\_requestQueue.DequeueWait(boost::posix\_time::milliseconds(500)));

if (reqHolder)

{

char\* errMsg = nullptr;

int result = sqlite3\_exec(dbHolder.GetHandle(), reqHolder->SqlRequest().c\_str(),

reqHolder->Callback(), reqHolder.get(), &errMsg);

reqHolder->RequestFinished(result == SQLITE\_OK);

sqlite3\_free(errMsg);

}

}

}

}

bool DataBaseService::ShouldShutdown()

{

Lock lock;

return m\_shouldShutdown;

}

void DataBaseService::StopJoinableThread()

{

if (m\_thread && m\_thread->joinable())

{

Stop();

m\_thread->join();

m\_thread.reset();

}

}

**ПРИЛОЖЕНИЕ Д**

**Исходный код класса UsersMessageView**

UsersMessageView::UsersMessageView(IDropResultHandler\* dropHandler)

: QSplitter(Qt::Vertical)

, m\_msgView(nullptr)

, m\_msgEdit(nullptr)

, m\_lastMsg(new db::MessageList())

{

CreateSubControls(dropHandler);

CreateStaticLink();

}

UsersMessageView::~UsersMessageView()

{

}

void UsersMessageView::CreateSubControls(IDropResultHandler\* dropHandler)

{

m\_msgView = new QTableWidget();

m\_msgView->setColumnCount(MessageItemCreator::COLUMN\_COUNT);

m\_msgView->setShowGrid(false);

m\_msgView->setStyleSheet(MESSAGE\_WIDGET\_STYLE);

QHeaderView\* verticalHeader = m\_msgView->verticalHeader();

verticalHeader->setVisible(false);

verticalHeader->setSectionResizeMode(QHeaderView::ResizeToContents);

QHeaderView\* horizontalHeader = m\_msgView->horizontalHeader();

horizontalHeader->setVisible(false);

horizontalHeader->setSectionResizeMode(MessageItemCreator::NAME\_COLUMN, QHeaderView::Fixed);

horizontalHeader->setSectionResizeMode(MessageItemCreator::MESSAGE\_COLUMN, QHeaderView::Stretch);

horizontalHeader->setSectionResizeMode(MessageItemCreator::TIME\_COLUMN, QHeaderView::Fixed);

horizontalHeader->setDefaultSectionSize(DEFAULT\_COLUMN\_WIDTH);

m\_msgEdit = new controls::DnDTextEdit(dropHandler);

addWidget(m\_msgView);

addWidget(m\_msgEdit);

}

void UsersMessageView::CreateStaticLink()

{

controls::StaticLink\* link = new controls::StaticLink();

link->setTextInteractionFlags(Qt::LinksAccessibleByMouse);

link->setText(SetLinkStyle("Last Conversation"));

link->setAlignment(Qt::AlignHCenter);

connect(link, SIGNAL(clicked()), SLOT(ShowLastConversations()));

m\_msgView->insertRow(0);

m\_msgView->setCellWidget(0, MessageItemCreator::MESSAGE\_COLUMN, link);

}

void UsersMessageView::ClearMessageEdit()

{

m\_msgEdit->clear();

m\_msgEdit->document()->clear();

}

bool UsersMessageView::GetTextFromEdit(std::wstring\* msg)

{

QString text(m\_msgEdit->toPlainText());

PrepareMessage(text);

ClearMessageEdit();

if (!text.isEmpty())

{

msg->assign(text.toStdWString());

return true;

}

return false;

}

void UsersMessageView::AppendTxtMessage(const MessageInfo& msg)

{

const int rowNum = m\_msgView->rowCount();

InsertMessageImpl(msg, rowNum, MessageItemCreator(m\_msgView, rowNum));

}

IProgressUIObserver\* UsersMessageView::AppendFileMessage(const MessageInfo& msg)

{

const int rowNum = m\_msgView->rowCount();

FileMessageItemCreator creator(m\_msgView, rowNum);

InsertMessageImpl(msg, rowNum, creator);

return creator.GetProgressObserver();

}

void UsersMessageView::InsertTxtMessageFromDB(const MessageInfo& msg, int rowNum)

{

InsertMessageImpl(msg, rowNum, MessageItemCreator(m\_msgView, rowNum));

}

void UsersMessageView::InsertFileMessageFromDB(const MessageInfo& msg, int rowNum)

{

InsertMessageImpl(msg, rowNum, DBFileMessageItemCreator(m\_msgView, rowNum));

}

void UsersMessageView::InsertMessageImpl(const MessageInfo& msg, int rowNum,

MessageItemCreator& creator)

{

m\_msgView->insertRow(rowNum);

creator.CreateItems(msg);

}

void UsersMessageView::AddLastConversations(const db::MessageListPtr& messages)

{

QCoreApplication::postEvent(this, new AddConversationFromDBEvent(messages));

}

bool UsersMessageView::event(QEvent\* ev)

{

if (ev->type() == AddConversationFromDBEvent::type)

{

AddConversationFromDBEvent\* event = static\_cast<AddConversationFromDBEvent\*>(ev);

SaveLastConversations(event->m\_messages);

return true;

}

return QWidget::event(ev);

}

void UsersMessageView::SaveLastConversations(const db::MessageListPtr& messages)

{

m\_lastMsg = messages;

}

void UsersMessageView::ShowLastConversations()

{

m\_msgView->removeRow(0);

for (auto iter = m\_lastMsg->rbegin(); iter != m\_lastMsg->rend(); ++iter)

{

switch (iter->first)

{

case db::TEXT\_MSG:

InsertTxtMessageFromDB(iter->second, 0);

break;

case db::FILE\_MSG:

InsertFileMessageFromDB(iter->second, 0);

break;

}

}

}

**РЕЦЕНЗИЯ**

**на дипломный проект**

**слушателя группы 15ПО11-04з Селюка Ивана Чеславовича**

**по специальности**

**1-40 01 73 «Программное обеспечение информационных систем»**

**на тему**

**«Программное средство «Сетевой чат»**

В дипломном проекте перед слушателем была поставлена задача разработки программного средства «Сетевой чат» для передачи файлов и текстовых сообщений в локальной сети.

Слушатель проанализировал предметную область, выполнил проектирование и разработку информационной системы для решения поставленной задачи. Разработка приложения велась в среде Visual Studio 2010, на языке программирования – С++. При проектировании информационная система программного средства была разделена на модули в зависимости от выполняемых функций. В работе использовались эффективные библиотеки, как для работы с пользовательским интерфейсом (Qt), так и для передачи данных по сети (ACE). Разработка программного средства велась с использованием многопоточного программирования, использования паттернов объектно-ориентированного проектирования.

Среди недостатков представленного дипломного проекта можно выделить: недостаточный анализ существующего программного обеспечения для передачи сообщений в локальной сети, отсутствие модуля помощи для пользователей в разработанном приложении. Среди достоинств можно выделить: простоту и легкость использования, быструю и простую настройку сетевых параметров для работы приложения.

Разработанное программное средство может быть использовано в локальных сетях предприятий, а так же в домашних сетях для общения пользователей и передачи информации.

Диплом заслуживает высокой оценки, а студент Селюк И.Ч. – присвоения ему квалификации инженера-программиста.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вывод и оценка проекта: | | Допускается к защите | | | |
|  | |  |  |  |  |
|  | | |  |  |  |
| Рецензент: д.т.н., профессор Уласюк Н.Н. | | |  |  |  |
|  | (ФИО) | | (подпись) |  | (дата) |

**ОТЗЫВ**

**на дипломный проект**

**слушателя группы 15ПО11-04з Селюка Ивана Чеславовича**

**по специальности**

**1-40 01 73 «Программное обеспечение информационных систем»**

**на тему**

**«Программное средство «Сетевой чат»**

Тема дипломного проекта является актуальной, так как ввиду быстрых темпов развития информационных технологий и компьютерных сетей использование чатов упрощает и ускоряет передачу информации между пользователями в рамках внутренних локальных и глобальных сетей.

В дипломном проекте перед слушателем была поставлена задача создания программного средства «Сетевой чат», которое бы использовалось для передачи файлов и текстовых сообщений в локальной сети.

В ходе работы над дипломным проектом слушателем была изучена специальная литература по разработке клиент-серверных и сетевых приложений, самостоятельно спроектирована информационная система с помощью средств автоматизированного проектирования Rational Rose. Разработка приложения велась в среде Visual Studio 2010, на языке программирования С++. В работе модулей приложения использовались библиотеки, как для работы с пользовательским интерфейсом (Qt), так и для передачи данных по сети (ACE). Также была проведена исследовательская работа в области многопоточного программирования и использования паттернов объектно-ориентированного проектирования.

В результате выполнения дипломного проекта было разработано программное средство, которое может быть использовано в локальных сетях предприятий, а также в домашних сетях для общения пользователей и передачи информации между ними.

Дипломный проект Селюка Ивана Чеславовича свидетельствует о высоком уровне его профессиональной подготовки и позволяет присвоить ему квалификацию «инженер-программист».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вывод и оценка проекта: | |  | | |
|  | |  |  |  |
| Руководитель дипломного  проекта: Кравцова А.А. | |  |  |  |
|  | (ФИО) | (подпись) |  | (дата) |