

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
Московский государственный университет технологии и управления
имени К.Г. Разумовского (Первый казачий университет)
Университетский колледж информационных технологий

Специальность 09.02.03 Программирование в компьютерных системах

Курсовой проект

Модуль ПМ.01 Разработка программных модулей программного
обеспечения для компьютерных систем
МДК.01.02 Прикладное программирование

на тему «Разработка Серверного компонента программного комплекса для обмена
текстовыми сообщениями»

Пояснительная записка
УКИТ 09.02.03.2017.304.22ПЗ

Группа	<u>П-304</u>	
Студент	<u>(личная подпись)</u>	Дубин Е.П.
Руководители проекта	<u>(личная подпись)</u>	Глускер А.И.
	<u>(личная подпись)</u>	Гусева Е.Л.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
ОСНОВНАЯ ЧАСТЬ	7
1 Введение в предметную область	7
2 Спецификация	7
2.1 Введение.....	7
2.2 Основание для разработки	7
2.3 Назначение разработки.....	7
2.4 Требования к программе или программному изделию	8
2.5 Требования к программной документации	9
2.6 Стадии и этапы разработки	10
2.7 Порядок контроля и приемки	10
3 Разработка программы и методики испытаний	11
3.1 Объект испытаний.....	11
3.2 Цель испытаний.....	11
3.3 Требования к программе	11
3.4 Требования к программной документации	12
3.5 Средства и порядок испытаний	13
3.6 Методы испытаний	14
3.7 Метод проверки требований к исходным кодам в части компиляции.....	20
4 Разработка ПЗ	21
4.1 Введение.....	21
4.2 Назначение и область применения.....	21
4.3 Технические характеристики	21
5 Разработка программы.....	24
6 Тестирование	26
6.1 Тестирование требований к исходному коду.....	26
6.2 Тестирование программы.....	31
ЗАКЛЮЧЕНИЕ.....	33
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	34
ПРИЛОЖЕНИЕ А Исходный код	35

ПРИЛОЖЕНИЕ Б Git log.....	62
ПРИЛОЖЕНИЕ В Issues.....	66
ПРИЛОЖЕНИЕ Г Руководство оператора.....	67
ПРИЛОЖЕНИЕ Д Руководство программиста	69
ПРИЛОЖЕНИЕ Е Доклад и презентация	71

ВВЕДЕНИЕ

Цель проекта

Разработать серверный компонент программного комплекса для обмена текстовыми сообщениями.

Задачи проекта :

- Изучить предметную область;
- Составить ТЗ;
- Составить программу и методику испытаний;
- Составить пояснительную записку;
- Написать код программы;
- Составить руководство пользователя;
- Провести тестирования приложения;
- Составить презентацию.

Актуальность

Актуальность данного курсового проекта для автора заключается в изучении основных функций библиотек boost, отточить навыки использования C/C++, а именно стандарта C++11, требуемого для разработки подобного приложения. Также, для составления соответствующей документации к проекту, требуется знать различные стандарты по написанию отдельных ее компонентов. Помимо этого реализация этого проекта дает опыт работы с большими проектами и работы с сетью.

При написании кода использовались принципы ООП и структурного программирования. Сама разработка велась по принципам стандартной каскадной модели. Для реализации была изучена различная литература по языку, документация библиотеки boost.

Список использованных методов

- Эксперимент;
- Сравнение;
- Формализация;

- Восхождение от абстрактного к конкретному;
- Анализ;
- Моделирование;
- Системный подход;
- Абстрагирование.

Обоснование структуры

1 Введение в предметную область

Данный раздел описывает предметную область, для которой разрабатывается игра.

2 Спецификация

Данный раздел описывает техническое задание, которое устанавливает требования к серверному компоненту программного комплекса для обмена текстовыми сообщениями.

3 Программа и методика испытания

Данный раздел описывает применяемые методы тестирования и тестовые примеры к разрабатываемому серверному компоненту программного комплекса для обмена текстовыми сообщениями.

4 Технический проект

Данный раздел описывает архитектурные решения, который применяются при разработке.

5 Реализация программного продукта

Данный раздел описывает процесс разработки самого серверного компонента программного комплекса для обмена текстовыми сообщениями.

6 Тестирование программного продукта

Данный раздел описывает результаты тестирования серверного компонента программного комплекса для обмена текстовыми сообщениями по программе и методике испытаний.

7 Заключение

Данный раздел подводит итог по проделанной работе.

8 Список используемых источников

В данном разделе приводиться список используемых источников, которые использовались при разработке приложения.

9 Приложение

В данном разделе приводятся дополнительная информация, которая включает в себя руководство оператора, исходный код, протокол системы контроля версий, руководство программиста, доклад.

ОСНОВНАЯ ЧАСТЬ

1 Введение в предметную область

Обычно в приложениях типа «Мессенджер» используется довольно простая архитектура «Клиент-Сервер». Смысл мессенджеров заключается в обмене текстовыми сообщениями между пользователями, посредством сети интернет. Примерами таких приложений могут служить: «Telegram», «WhatsApp», «Viber», «Line» и т.п.

2 Спецификация

В данном разделе представлено техническое задание к проекту, описывающее все цели и требования к компоненту и последующим документам. Также описана основная информация о программном продукте, требования по эксплуатации и разработке и выполняемый им функционал.

2.1 Введение

2.1.1 Наименование программы

Серверный компонент программного комплекса для обмена текстовыми сообщениями

2.1.2 Область применения

Деятельность по предоставлению средств для текстового общения в сети интернет

2.1.3 Объект, в котором используют программу

Программа «Серверный компонент программного комплекса для обмена текстовыми сообщениями» сможет быть использована в любых объектах

2.2 Основание для разработки

2.2.1 Документ, на основании которого ведется разработка

Задание на курсовой проект

2.2.2 Наименование и (или) условное обозначение темы для разработки

Текстовый мессенджер

2.3 Назначение разработки

2.3.1 Функциональное назначение

Предоставление программных средств для передачи текстовых сообщений

2.3.2 Эксплуатационное назначение

Программа предназначена для использования неограниченным кругом лиц

2.4 Требования к программе или программному изделию

2.4.1 Требования к функциональным характеристикам

2.4.1.1 Требования к составу выполняемых функций

Программа должна обеспечивать выполнение следующих функций:

- Аутентификация;
- Передача текстовых сообщений;
- Прием персональных данных пользователя;
- Сохранение персональных данных пользователя;
- Обработка запросов на подключение клиентов;
- Обработка запроса на отключение клиента;
- Принудительное отключение клиента;
- Обработка запроса на проверку статуса пользователя.

2.4.1.2 Требования к организации обмена данными с клиентом

Данные передаются и читаются через сокет

2.4.1.3 Требования к временным характеристикам

- Время передачи пакета не должно превышать 0.5 секунд;
- Время обработки запроса не должно превышать 1 секунды.

2.4.1.4 Требования к надежности

2.4.1.4.1 Требования к контролю файловых ошибок

- Контролировать файловые ошибки, генерируемые файловой системой;
- Контролировать ошибки доступа к файлам с ограниченным доступом.

2.4.1.4.2 Требования к надежности работы с сетью

- Проверка ошибочных пакетов и их обработка соответственно протоколу;
- Контролирование корректности формата пакетов;
- Другие требования к надежности не предъявляются.

2.4.1.5 Условия эксплуатации

2.4.1.5.1 Минимальное количество персонала

Один человек, владеющий английским языком, обладающий практическими навыками по администрированию серверов.

2.4.1.6 Требования к составу и параметрам технических средств

В состав технических средств должен входить компьютер, включающий:

- процессор Pentium или более современный;
- клавиатуру;
- видеокарту;
- монитор;
- жесткий или SSD-диск;
- Сетевое соединение со скоростью более 256Кб/с.

2.4.1.7 Требования к информационной и программной совместимости

Требования к информационным структурам на входе и выходе не предъявляются

- Требования к методам решения не предъявляются;
- Требования к исходным кодам изложены в документе Google C++ Style Guide [Электронный ресурс];
- Программа должна быть написана на языке C/C++ и компилироваться транслятором g++;
- Программа должна работать под управлением операционной системы Ubuntu 12.04+ или Mint 18.02+;
- Требования к защите информации – аутентификация.

2.5 Требования к программной документации

2.5.1 Состав программной документации

Состав программной документации должен включать:

- Техническое задание;
- Пояснительную записку;
- Текст программы;

- Текст программы, осуществляющей автоматическое тестирование программы «Серверный компонент программного комплекса для обмена текстовыми сообщениями»;

- Инструкцию пользователя;
- Программу и методику испытаний.

2.5.2 Специальные требования к пояснительной записке

Пояснительная записка должна содержать UML диаграмму классов, используемых в программе.

2.6 Стадии и этапы разработки

2.6.1 Стадии разработки

Разработка осуществляется в три стадии:

- техническое задание;
- технический проект;
- рабочий проект.

2.6.2 Этапы разработки

На стадии техническое задание осуществляется разработка, согласование и утверждение технического задания в срок до 26 февраля 2016 года Глускером А. И. На стадии технический проект осуществляется разработка, согласование и утверждение пояснительной записки в срок до 5 марта 2016 года Глускером А. И. На стадии рабочий проект осуществляется разработка текста программы, осуществляющей автоматическое тестирование программы «решение квадратного уравнения», разработка, согласование и утверждение программы и методики испытаний, текста программы в срок до 7 марта 2016 года Глускером А. И., после чего осуществляются испытания по результатам которой возможно будет проводиться корректировка программной документации в срок до 12 марта 2016 года.

2.7 Порядок контроля и приемки

Приемосдаточные испытания должны проводиться в соответствии с программой и методикой испытаний, разработанной, согласованной и утвержденной не позднее 30 декабря 2017 года.

В данном разделе были изложены все основные особенности программного продукта и требования к нему. Автор лучше освоил принципы составления технического задания и формулировки требований.

3 Разработка программы и методики испытаний

В данном разделе представлена программа и методика испытаний, которая содержит некоторые более точные требования, а также требования к тестированию.

3.1 Объект испытаний

3.1.1 Наименование

Серверный компонент программного комплекса для обмена текстовыми сообщениями

3.1.2 Область применения

Деятельность по предоставлению средств для текстового общения в сети интернет

3.1.3 Обозначение программы

Текстовый мессенджер

3.2 Цель испытаний

Проверка соответствия программного продукта требованиям технического задания.

3.3 Требования к программе

3.3.1 Требования к функциональным характеристикам

3.3.1.1 Требования к составу выполняемых функций

Программа должна обеспечивать выполнение следующих функций (п. 4.1.1 технического задания):

- Аутентификация;
- Передача текстовых сообщений;
- Прием персональных данных пользователя;
- Сохранение персональных данных пользователя;
- Обработка запросов на подключение клиентов;

- Обработка запроса на отключение клиента;
- Принудительное отключение клиента;
- Обработка запроса на проверку статуса пользователя.

3.3.1.2 Требование к обмену данными с клиентом

Данные передаются и читаются через сокет (п. 4.1.2 технического задания)

3.3.1.3 Требования к временным характеристикам

- Время передачи пакета не должно превышать 0.5 секунд.

(п. 4.1.3.1 технического задания);

- Время обработки запроса не должно превышать 1 секунды.

(п. 4.1.3.2 технического задания).

3.3.2 Требования к надежности

3.3.2.1 Требования к контролю файловых ошибок

- Контролировать файловые ошибки, генерируемые файловой системой. (п. 4.2.1.1 технического задания).

3.3.2.2 Требования к надежности работы с сетью

- Проверка ошибочных пакетов и их обработка соответственно протоколу. (п. 4.2.2.1 технического задания);
- Контролирование корректности формата пакетов.

Другие требования к надежности не предъявляются

3.3.3 Требования к информационной и программной совместимости

Программа должна работать под управлением операционной системы Ubuntu 12.04+ или Mint 18.02+ (п. 4.5.5 технического задания)

3.4 Требования к программной документации

3.4.1 Состав программной документации(п. 5.1 технического задания):

- Техническое задание;
- Пояснительную записку;
- Текст программы;
- Текст программы, осуществляющей автоматическое тестирование

программы «Серверный компонент программного комплекса для обмена текстовыми сообщениями»;

- Инструкцию пользователя;
- Программу и методику испытаний.

3.4.2 Специальные требования к пояснительной записке

Пояснительная записка должна содержать UML и диаграмму классов, используемых в программе (п. 5.2 технического задания)

3.4.3 Специальные требования к тексту программы

- Требования к исходным кодам изложены в документе Google C++ Style Guide [Электронный ресурс];
- Программа должна быть написана на языке C/C++ и компилироваться транслятором g++ (п. 4.5.4 технического задания).

3.5 Средства и порядок испытаний

3.5.1 Технические средства, используемые при проведении испытаний

В состав технический средств входит 2 IBM-совместимый компьютера, включающих:

- процессор Pentium или более современный;
- клавиатуру;
- видеокарту;
- монитор;
- жесткий или SSD-диск;
- Сетевое соединение со скоростью более 256Кб/с.

3.5.2 Программные средства, используемые при проведении испытаний

В состав программных средств входит: Операционная система Linux Mint 18.3

3.5.3 Порядок проведения испытаний

3.5.3.1 Подготовка к проведению испытаний

Подготовка заключается в обеспечении наличия компьютера, описанного в п.

5.1, и программных средств, указанных в п. 5.2, установленных на этом компьютере, а также в обеспечении наличия оператора, управляющего сервером и оператора, управляющего клиентом.

3.5.3.2 Ход проведения испытаний

Ход проведения испытаний документируется в протоколе, где указывается перечень проводимых испытаний, результат каждого испытания и возможно замечания.

3.5.3.3 Состав испытаний:

Проверка состава программной документации в соответствии с методом, описанном в п. 6.2

3.5.3.3.2 Проверка требований к программе

Проверка обеспечения требований к программе (п. 3) в соответствии с методом, описанным в п. 6.1

3.5.3.3.3 Проверка требований к программной документации

- Проверка пояснительной записки (п. 4.2) в соответствии с методом, описанным в п. 6.5;
- Проверка текстов программ (п. 4.3.1) в соответствии с методом, описанным в п.6.7;
- Проверка текстов программ (п. 4.3.2) в соответствии с методом, описанным в п.6.8.

3.6 Методы испытаний

3.6.1 Метод проверки требований к программе

Проверка осуществляется путем запуска программы и провоцирования определенных ситуаций (Таблица 1).

При запуске серверного компонента программного комплекса осуществляется «прослушка» порта и ожидание новых подключений, далее происходит их обработка.

Каждое новое подключение выделяется в отдельный поток и прослушивается сервером, принимая запросы от клиентов, обрабатывает их и отвечает соответствующему клиенту.

Таблица 1 — Набор тестов

Номер	Ситуация	Действия
1	Передача текстового сообщения	Визуально проверить сообщение, которое пришло клиенту с тем, что вывелось в консоли сервера
2	Аутентификация клиента	В консоли сервера появится сообщение об успешной аутентификации
3	Обработка запроса на проверку существования пользователя	У клиента появится иконка, что данный проверяемый пользователь существует
4	Сохранение персональных данных пользователя	При регистрации нового пользователя клиент посылает, введенные, логин и пароль серверу, сервер сохраняет эти данные в файл. Для проверки открыть файл и проверить данные с ожидаемыми.
5	Обработка запросов на подключение клиента	В консоли сервера появится сообщение о подключении нового пользователя
6	Обработка отключения клиента	В консоли сервера появится сообщение об отключении клиента от сервера

3.6.2 Метод проверки требований к составу программной документации

Проверка состава программной документации осуществляется визуально

путем сравнения набора предъявленных документов (в форме распечатки или в рукописной форме) списку, приведенному в п. 4.1.

При этом исходные тексты программ должны быть предоставлены так же и в электронной форме. В случае если набор предъявленных документов соответствует списку, а исходные тексты предоставлены также в электронной форме, то в протокол вносится запись: «Состав программной документации» – соответствует; в противном случае: «Состав программной документации» – не соответствует.

3.6.3 Метод проверки требований к пояснительной записке

– Пояснительная записка должна соответствовать стандарту — ГОСТ 19.105-78

– UML диаграммы классов должны соответствовать стандарту — UML 2.4.1 стандарта ISO/IEC 19505-1

3.6.4 Метод проверки требований к техническому заданию

– Техническое задание должно соответствовать стандарту — ГОСТ 19.201.78

3.6.5 Метод проверки требований к пояснительной записке

– Пояснительная записка должна соответствовать стандарту — ГОСТ 19.105-78

3.6.6 Метод проверки требований к программе и методике испытаний

– Программа и методика испытания должна соответствовать стандарту — ГОСТ 19.101-77

3.6.7 Метод проверки требований к исходным кодам

Исходный код должен соответствовать всем стандартам, описанным в документе Google C++ Style Guide [Электронный ресурс]. Для каждого файла вносится в протокол запись: «Требования к исходным кодам для файла #####» – соответствует/не соответствует (где вместо ##### указывается название файла). Проверка осуществляется по следующим пунктам, взятым из Google C++ Style Guide:

- Каждый заголовочный файл должен оканчиваться .h;
- Файлы предназначенные для включения должны оканчиваться на .inc;
- Все файлы заголовков должны содержать #define guard для предотвращения множественного включения;

- Избегайте использования `forward` объявлений, где это возможно;
- Определяйте функции `inline` только тогда, когда они достаточно малы, например, 10 строк или меньше;
- Используйте стандартный порядок чтения и избегайте скрытых зависимостей: Связанный заголовок, библиотека C, библиотека C++, .h других файлов .h;
- Пространства имен должны иметь уникальные имена, основанные на имени проекта и, возможно, на его пути. Не используйте директивы `using`. Не используйте встроенные пространства имен;
- Когда определения в файле .cc не должны упоминаться вне этого файла, поместите их в неназванное пространство имен или объявите их статическими. Не используйте ни одну из этих конструкций в файлах .h;
- Предпочитают размещение функций не членов в пространстве имен; редко использовать глобальные функции. Предпочитают группировать функции с пространством имен вместо использования класса, как если бы это было пространство имен. Статические методы класса обычно должны быть тесно связаны с экземплярами класса или статических данных класса;
- Поместите переменные функции в максимально узкую область и инициализируйте переменные в объявлении;
- Переменные типа класса со статической продолжительностью хранения запрещены: они вызывают труднодоступные ошибки из-за неопределенного порядка построения и уничтожения. Однако такие переменные допускаются, если они являются `constexpr`: они не имеют динамической инициализации или освобождения памяти, выделенной под переменную(уничтожения);
- Избегайте вызовов виртуальных методов в конструкторах и избегайте инициализации, которая может выйти из строя, если вы не можете сообщить об ошибке;
- Не указывать неявные преобразования. Используйте ключевое слово `explicit` для операторов преобразования и конструкторов с одним аргументом;
- Поддерживайте копирование и / или перемещение, если эти операции

ясны и значимы для вашего типа. В противном случае отключите неявно созданные специальные функции, которые выполняют копии и перемещения;

- Использовать структуру только для пассивных объектов, которые несут данные; все остальное - класс;
- Composition часто более уместен, чем наследование. При использовании наследования сделайте его общедоступным;
- Лишь очень редко используется многократное наследование реализации. Мы допускаем множественное наследование только тогда, когда у большинства базовых классов есть реализация;
- Классам, которые удовлетворяют определенным условиям, допускается, но не требуется, завершаться с суффиксом `interface`;
- Операторы перегрузки допустимы;
- Группируйте аналогичные объявления вместе, размещая `public` части раньше.
- Определение класса обычно должно начинаться с раздела `public`: `section`, затем защищенного `:`, затем `private` `:`. Опустите разделы, которые будут пустыми. Внутри каждого раздела обычно предпочитают группировать похожие виды объявлений вместе и обычно предпочитают следующий порядок: типы (включая `typedef`, `using` и вложенные структуры и классы), константы, фабричные функции, конструкторы, операторы присваивания, деструктор, все другие методы, данных;
- При определении функции порядок параметров: `input`, а затем `output`;
- Предпочитаются небольшие и сфокусированные функции;
- Все параметры, переданные по ссылке, должны быть помечены как `const`;
- Используйте перегруженные функции (включая конструкторы) только в том случае, можно получить представление о том, что происходит, не задумываясь о том, какая именно перегрузка вызывается;
- Аргументы по умолчанию допускаются для не виртуальных функций,

когда гарантируется, что значение по умолчанию всегда имеет одинаковое значение;

- Используйте потоки, где это необходимо;
- Используйте префиксную форму (`++ i`) операторов инкремента и декремента с итераторами и другими объектами шаблона;
- Используйте `const`, когда это имеет смысл. `C++11`, `constexpr` – лучший выбор для некоторых применений `const`;
- В `C++11` используйте `constexpr` для определения истинных констант или для обеспечения постоянной инициализации;
- Из встроенных целых типов `C++` единственное, что используется, – `int`. Если программе нужна переменная другого размера, используйте целочисленный тип точной ширины из `<stdint.h>`, например `int16_t`. Если ваша переменная представляет значение, которое может быть больше или равно 2^{31} (2GiB), используйте 64-битный тип, такой как `int64_t`;
- Избегайте определять макросы, особенно в заголовках. Предпочитают встроенные функции, перечисления и константные переменные. Назовите макросы с префиксом, специфичным для проекта. Не используйте макросы для определения фрагментов `C++ API`;
- Используйте `0` для целых чисел, `0.0` для действительных чисел, `nullptr` (или `NULL`) для указателей и `'\0'` для символов;
- Предпочитайте `sizeof (varname)` в `sizeof (type)`;
- Используйте `auto`, чтобы избежать имен типов, которые являются раздражающими, очевидными или несущественными – случаи, когда тип не помогает ясности читателю;
- При необходимости используйте лямбда-выражения. Предпочитайте явные захваты, когда лямбда выйдет из текущей области;
- Избегайте сложного программирования шаблонов;
- Используйте только утвержденные библиотеки из коллекции библиотек Boost;
- Имена должны быть описательными; избегать аббревиатуры. Дайте как

можно более подробное имя, в пределах разумного. Не беспокойтесь о сохранении горизонтального пространства, так как гораздо важнее сделать код понятным для нового читателя. Не используйте аббревиатуры, которые являются двусмысленными или незнакомыми для читателей вне вашего проекта, и не сокращайте их, удаляя буквы внутри слова;

- Имена файлов должны быть строчными и содержать символы подчеркивания (`_`) или тире (`-`). Следуйте соглашению, которое использует ваш проект. Если нет последовательного локального шаблона, выберите «`_`»;

- Имена типов начинаются с заглавной буквы и имеют заглавные буквы для каждого нового слова без подчеркивания;

- Имена переменных (включая функциональные параметры) и элементы данных имеют строчные буквы, с подчеркиваниями между словами. Элементы данных классов (но не структур) дополнительно имеют завершающие символы подчеркивания;

- Переменные, объявленные `constexpr` или `const`, и значение которых фиксировано для продолжительности программы, называются с ведущим «`k`», за которым следует смешанный случай;

- Регулярные функции имеют смешанный случай; аксессоры и мутаторы могут быть названы как переменные.

- В случае, если стандарт был соблюден, то в раздел «Тестирование» вносится запись о соответствии файла требованиям, в противном случае – о несоответствии.

3.7 Метод проверки требований к исходным кодам в части компиляции

Изложенный ниже метод применяется ко всем файлам, содержащим исходный текст, и входящим в состав программной документации по отдельности. Для каждого файла вносится в протокол запись: «Требования к исходным кодам в части компиляции во C++ для файла #####» – соответствует/не соответствует (где вместо ##### указывается название файла). Проверка осуществляется путем открытия файла

с исходным кодом в среде C++, осуществления компиляции (обе работы делаются в соответствии с документацией к C++).

В случае, если компиляция завершилась успешно в протокол вносится запись о соответствии требованиям, в противном случае – о несоответствии.

В данном разделе была представлена программа и методика испытаний, включающая в себя требования для проверки и тестирования. Был получен опыт в составлении программы и методики испытаний

4 Разработка ПЗ

В данном пункте представлена пояснительная записка и UML диаграмма классов.

4.1 Введение

4.1.1 Наименование программы

Серверный компонент программного комплекса для обмена текстовыми сообщениями.

4.1.2 Условное обозначение темы разработки

Текстовый мессенджер

4.1.3 Документ, на основании которого ведется разработка

Техническое задание на курсовой проект.

4.2 Назначение и область применения

4.2.1 Назначение программы

Предоставление программных средств для передачи текстовых сообщений в сети интернет.

4.2.2 Краткая характеристика области применения

Программа предназначена для использования неограниченным кругом лиц.

4.3 Технические характеристики

4.3.1 Постановка задачи на разработку программы

Предоставление программных средств для передачи текстовых сообщений в сети интернет.

4.3.2 Описание структуры проекта

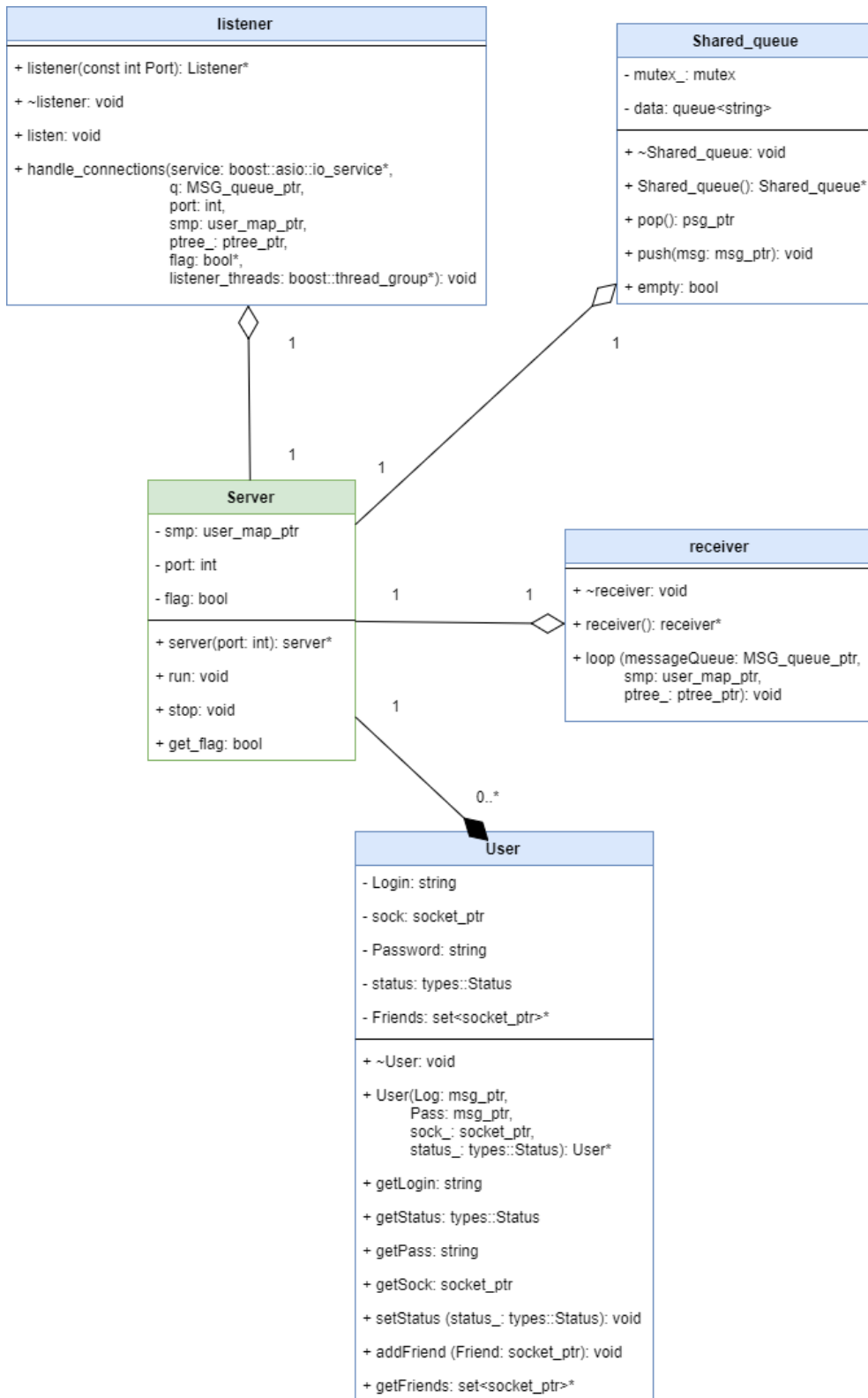


Рисунок 1 — UML диаграмма классов

Класс Server — главный класс, хранящий в себе экземпляр очереди сообщений, receiver и listener, запускающий сервер и останавливающий его по команде stop.

- `smr` – хранит пул пользователей;
- `port` – порт на котором запускается сервер;
- `flag` – флаг остановки сервера;
- `run()` – запуск приложения;
- `stop()` – остановка приложения;
- `get_flag()` – получить текущее состояние флага.

Класс `listener` — класс для прослушки порта сервера, приема новых соединений и приема сообщений от текущих соединений.

- `listen()` – прослушка текущего соединения и прием новых сообщений;
- `handle_connections()` – прослушка порта сервера и прием новых соединений.

Класс `Shared_queue` — класс, реализующий синхронизированную очередь для сообщений.

- `mutex_` – примитив синхронизации для захвата ресурса очереди;
- `data` – очередь сообщений;
- `pop()` – возвращает первое сообщение в очереди, и удаляет его из очереди;
- `push()` – добавляет сообщение в конец очереди;
- `empty()` – проверка очереди на пустоту.

Класс `receiver` — класс, отвечающий за обработку сообщений из очереди сообщений.

- `loop()` – цикл, реализующий обработку сообщений и вызов соответствующих функций.

Класс `User` — класс, отвечающий за хранение данных о подключившихся пользователях и взаимодействия с ними.

- `Login` – Логин пользователя;
- `Password` – Пароль пользователя;
- `sock` – Сокет по которому соединен пользователь;
- `status` – Текущий статус пользователя;

- Friends – Множество пользователей, которых необходимо оповестить, при смене статуса текущего пользователя;
- `getLogin()` – Возвращает логин данного пользователя;
- `getStatus()` – Возвращает статус данного пользователя;
- `getPass()` – Возвращает пароль данного пользователя;
- `getSock()` – возвращает сокет данного пользователя;
- `addFriend()` – добавляет пользователя, в список пользователей для оповещений;
- `getFriends()` – Возвращает список пользователей для оповещений.

4.3.3 Описание и обоснование выбора состава технических и программных средств

При разработке приложения использовалась коллекция библиотек boost.

Остальной состав технических и программных средств полностью описан в техническом задании (пп. 4.4; 4.5.4; 4.5.5).

В данном разделе была представлена пояснительная записка, содержащая некоторые объяснения по области применения, а также UML диаграмма. В ходе составления раздела, был получен опыт в составлении диаграмм классов.

5 Разработка программы

В этом разделе будет описано, как проходил процесс написания кода программы и что это дало автору.

Изначально работа была начата с знанием базового синтаксиса C/C++ без особых знаний стандарта C++11 и опыта работы с boost. Автор начал изучение документации по этой коллекции библиотек и стандарту C++11. После некоторого времени изучения начался этап проектирования. После того, как стала ясна примерная модель приложения, начался этап написания кода. Сперва был создан класс Server, отвечающий за хранение остальных компонентов и ресурсов. Далее этот класс слабо изменялся. Позже был реализован класс listener, отвечающий за прием данных от пользователей и перенаправления их в Lockfree queue из коллекции библиотек boost. С этим не возникало проблем. После этого был реализован класс receiver, отвечающий

за обработку сообщений, находящихся в Lockfree queue. Была написана функция взятия токена из сообщения, разделителем токенов служит символ «:».

Через некоторое время в этой функции были найдены критические ошибки, из-за которых утекло много времени разработки. Далее были написаны основные типы, основанные на умных указателях из коллекции библиотек boost. Это значительно сократило время разработки, поскольку утечки были практически исключены. Позже автор столкнулся с проблемой, что класс receiver не мог узнать какому пользователю отправлять ответ. Было решено создать класс User, хранящий в себе сокет, логин, пароль и другую информацию. В классе Server был создан пул экземпляров класса User, который далее передавался в receiver и listener. Таким образом класс receiver может узнать на какой сокет необходимо отправлять ответ. Позже, при тестах первых прототипов приложения выяснилось, что в Lockfree queue инвалидация указателей, а умные указатели, не имеющие тривиального деструктора, не подходили. Было решено написать свою очередь, которая будет синхронизированной и потокобезопасной, так был реализован класс Shared_queue.

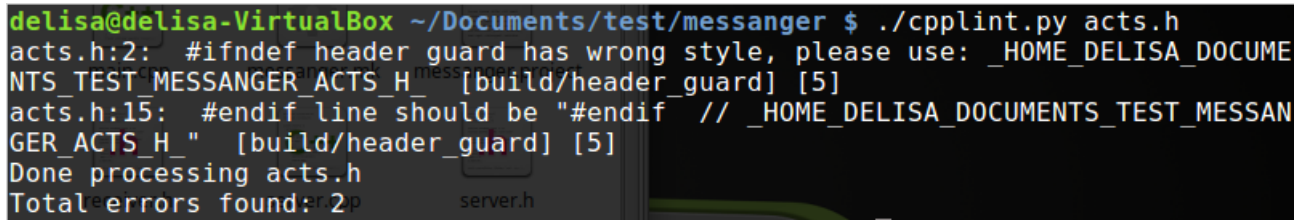
Далее, когда вопрос приема и обработки сообщений был решен, встала следующая проблема, аутентификации пользователей. Для аутентификации был выбран простой механизм, при подключении нового пользователя он передавал данные для регистрации/аутентификации, далее данные сохранялись парами (Login:Password) в property_tree из коллекции библиотек boost. Проверка пароля происходит довольно просто, если в дереве по ключу (Login) лежат данные (Password), не соответствующие проверяемым, то бросалось исключение, которое мы обрабатываем и регистрируем нового пользователя. После добавления нового пользователя в property_tree, данные из дерева записываются в JSON файл. JSON файл читается в property_tree, инициализируя его, только при запуске сервера.

В данном разделе был изложен процесс создания приложения. Благодаря этому проекту автор получил опыт работы с коллекцией библиотек boost, а точнее работу с сетью, работу с JSON/XML, Multithreading, работу с примитивами синхронизации, написания кода в стандарте C++11, а также реализации крупных приложений.

6 Тестирование

6.1 Тестирование требований к исходному коду

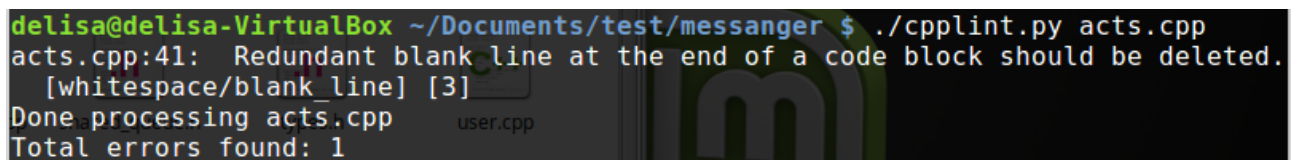
В этом разделе будут представлены результаты тестирования и описан процесс его проведения.



```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py acts.h
acts.h:2: #ifndef header guard has wrong style, please use: _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_ACTS_H_ [build/header_guard] [5]
acts.h:15: #endif line should be "#endif" // _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_ACTS_H_ [build/header_guard] [5]
Done processing acts.h
Total errors found: 2
```

Рисунок 2 – Проверка файла acts.h

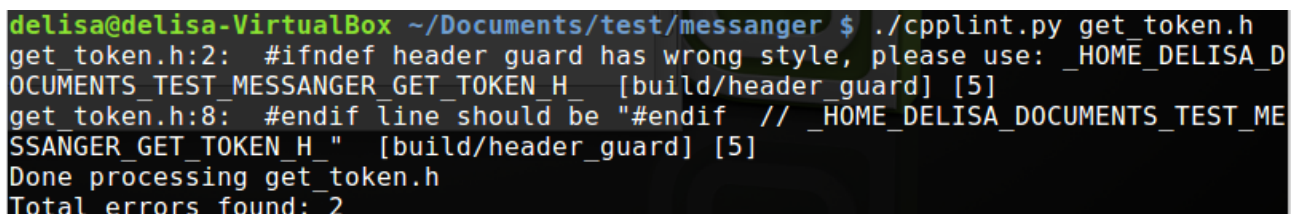
При проверке файла «acts.h» утилитой «cpplint» были выявлены 2 незначительные по приоритету ошибки, которые являются скорее изъятиями самой утилиты (Рисунок 2).



```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py acts.cpp
acts.cpp:41: Redundant blank line at the end of a code block should be deleted. [whitespace/blank_line] [3]
Done processing acts.cpp
Total errors found: 1
```

Рисунок 3 – Проверка файла acts.cpp

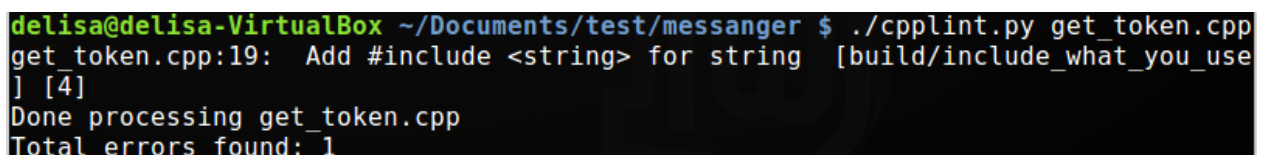
При проверке файла «acts.cpp» утилитой «cpplint» была выявлена 1 незначительная по приоритету ошибка (Рисунок 3).



```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py get_token.h
get_token.h:2: #ifndef header guard has wrong style, please use: _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_GET_TOKEN_H_ [build/header_guard] [5]
get_token.h:8: #endif line should be "#endif" // _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_GET_TOKEN_H_ [build/header_guard] [5]
Done processing get_token.h
Total errors found: 2
```

Рисунок 4 – Проверка файла get_token.h

При проверке файла «get_token.h» утилитой «cpplint» были выявлены 2 незначительные по приоритету ошибки, которые являются скорее изъятиями самой утилиты (Рисунок 4).



```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py get_token.cpp
get_token.cpp:19: Add #include <string> for string [build/include_what_you_use] [4]
Done processing get_token.cpp
Total errors found: 1
```

Рисунок 5 – Проверка файла get_token.cpp

При проверке файла «get_token.cpp» утилитой «cpplint» была выявлена 1 незначительная по приоритету ошибка, она связана с включением заголовочных файлов. Данная ошибка таковой не является, поскольку включение происходит на 1 уровень выше (Рисунок 5).

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py listener.h
listener.h:2: #ifndef header guard has wrong style, please use: _HOME_DELISA_DO
CUMENTS_TEST_MESSANGER_LISTENER_H_ [build/header_guard] [5]
listener.h:33: #endif line should be "#endif // _HOME_DELISA_DOCUMENTS_TEST_ME
SSANGER_LISTENER_H_" [build/header_guard] [5]
listener.h:15: <system_error> is an unapproved C++11 header. [build/c++11] [5]
listener.h:18: public: should be indented +1 space inside class listener [whit
espace/indent] [3]
Done processing listener.h
Total errors found: 4
```

Рисунок 6 – Проверка файла listener.h

При проверке файла «listener.h» утилитой «cpplint» были выявлены 4 незначительные по приоритету ошибки, которые являются скорее изъятиями самой утилиты (Рисунок 6).

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py listener.cpp
listener.cpp:17: Add #include <string> for string [build/include_what_you_use]
[4]
Done processing listener.cpp
Total errors found: 1
```

Рисунок 7 – Проверка файла listener.cpp

При проверке файла «listener.cpp» утилитой «cpplint» была выявлена 1 незначительная по приоритету ошибка, она связана с включением заголовочных файлов. Данная ошибка таковой не является, поскольку включение происходит на 1 уровень выше (Рисунок 7).

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py receiver.h
receiver.h:2: #ifndef header guard has wrong style, please use: _HOME_DELISA_DO
CUMENTS_TEST_MESSANGER_RECEIVER_H_ [build/header_guard] [5]
receiver.h:22: #endif line should be "#endif // _HOME_DELISA_DOCUMENTS_TEST_ME
SSANGER_RECEIVER_H_" [build/header_guard] [5]
receiver.h:14: public: should be indented +1 space inside class receiver [whit
espace/indent] [3]
Done processing receiver.h
Total errors found: 3
```

Рисунок 8 – Проверка файла receiver.h

При проверке файла «receiver.h» утилитой «cpplint» были выявлены 3 незначительные по приоритету ошибки, которые являются скорее изъянами самой утилиты (Рисунок 8).

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py receiver.cpp
receiver.cpp:54: Add #include <string> for string [build/include_what_you_use]
[4]
Done processing receiver.cpp
Total errors found: 1
```

Рисунок 9 – Проверка файла receiver.cpp

При проверке файла «receiver.cpp» утилитой «cpplint» была выявлена 1 незначительная по приоритету ошибка, она связана с включением заголовочных файлов. Данная ошибка таковой не является, поскольку включение происходит на 1 уровень выше (Рисунок 9).

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py server.h
server.h:2: #ifndef header guard has wrong style, please use: _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_SERVER_H_ [build/header_guard] [5]
server.h:24: #endif line should be "#endif // _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_SERVER_H_" [build/header_guard] [5]
server.h:8: public: should be indented +1 space inside class server [whitespace/indent] [3]
server.h:19: private: should be indented +1 space inside class server [whitespace/indent] [3]
server.h:10: Add #include <list> for list<> [build/include_what_you_use] [4]
Done processing server.h
Total errors found: 5
```

Рисунок 10 – Проверка файла server.h

При проверке файла «server.h» утилитой «cpplint» были выявлены 5 незначительные по приоритету ошибки, некоторые из них являются скорее, изъянами самой утилиты (Рисунок 10).

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py server.cpp
Done processing server.cpp
Total errors found: 0
```

Рисунок 11 – Проверка файла server.cpp

При проверке файла «server.cpp» утилитой «cpplint» ошибок обнаружено не было (Рисунок 11).

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py shared_queue.h
shared_queue.h:2: #ifndef header guard has wrong style, please use: _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_SHARED_QUEUE_H_ [build/header_guard] [5]
shared_queue.h:19: #endif line should be "#endif // _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_SHARED_QUEUE_H_" [build/header_guard] [5]
shared_queue.h:5: <mutex> is an unapproved C++11 header. [build/c++11] [5]
shared_queue.h:8: public: should be indented +1 space inside class Shared_queue [whitespace/indent] [3]
shared_queue.h:14: private: should be indented +1 space inside class Shared_queue [whitespace/indent] [3]
shared_queue.h:16: Add #include <string> for string [build/include_what_you_use] [4]
Done processing shared_queue.h
Total errors found: 6
```

Рисунок 12 – Проверка файла shared_queue.h

При проверке файла «shared_queue.h» утилитой «cpplint» были выявлены 6 незначительные по приоритету ошибок, некоторые из них являются скорее, изъянами самой утилиты (Рисунок 12).

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py shared_queue.cpp
shared_queue.cpp:8: Add #include <string> for string [build/include_what_you_use] [4]
Done processing shared_queue.cpp
Total errors found: 1
```

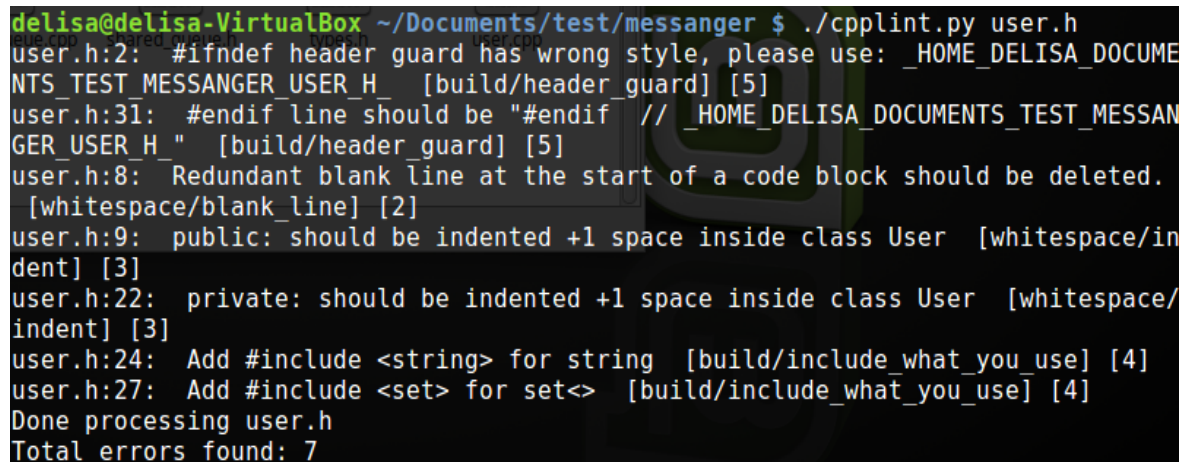
Рисунок 13 – Проверка файла shared_queue.cpp

При проверке файла «shared_queue.cpp» утилитой «cpplint» была выявлена 1 незначительная по приоритету ошибка, она связана с включением заголовочных файлов. Данная ошибка таковой не является, поскольку включение происходит на 1 уровень выше (Рисунок 13).

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py types.h
types.h:2: #ifndef header guard has wrong style, please use: _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_TYPES_H_ [build/header_guard] [5]
types.h:36: #endif line should be "#endif // _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_TYPES_H_" [build/header_guard] [5]
types.h:22: Redundant blank line at the start of a code block should be deleted. [whitespace/blank_line] [2]
types.h:28: Redundant blank line at the end of a code block should be deleted. [whitespace/blank_line] [3]
types.h:34: Namespace should be terminated with "// namespace types" [readability/namespace] [5]
types.h:17: Add #include <string> for string [build/include_what_you_use] [4]
Done processing types.h
Total errors found: 6
```

Рисунок 14 – Проверка файла types.h

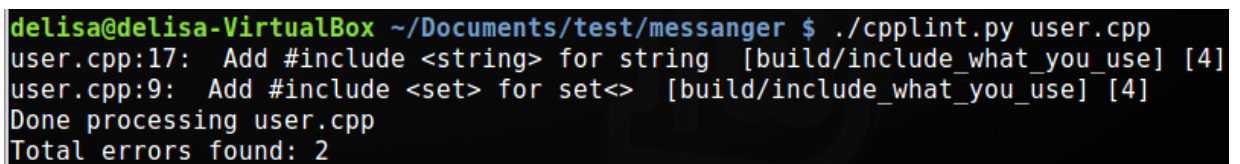
При проверке файла «types.h» утилитой «cpplint» были выявлены 6 незначительные по приоритету ошибок, некоторые из них являются скорее, изъянами самой утилиты (Рисунок 14).

A terminal window showing the output of the cpplint.py utility applied to user.h. The output lists several errors: a wrong header guard style, an endif line issue, a redundant blank line, and indentation problems in the User class. It also suggests adding #include directives for <string> and <set>. The total errors found are 7.

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py user.h
user.h:2: #ifndef header guard has wrong style, please use: _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_USER_H_ [build/header_guard] [5]
user.h:31: #endif line should be "#endif" // _HOME_DELISA_DOCUMENTS_TEST_MESSANGER_USER_H_ [build/header_guard] [5]
user.h:8: Redundant blank line at the start of a code block should be deleted. [whitespace/blank_line] [2]
user.h:9: public: should be indented +1 space inside class User [whitespace/indent] [3]
user.h:22: private: should be indented +1 space inside class User [whitespace/indent] [3]
user.h:24: Add #include <string> for string [build/include_what_you_use] [4]
user.h:27: Add #include <set> for set<> [build/include_what_you_use] [4]
Done processing user.h
Total errors found: 7
```

Рисунок 15 – Проверка файла user.h

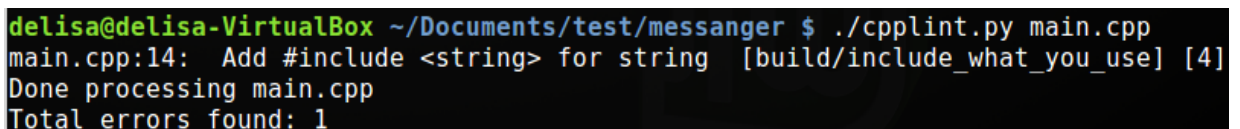
При проверке файла «user.h» утилитой «cpplint» были выявлены 7 незначительные по приоритету ошибок, некоторые из них являются скорее, изъянами самой утилиты (Рисунок 15).

A terminal window showing the output of the cpplint.py utility applied to user.cpp. It identifies two errors: missing #include directives for <string> and <set>. The total errors found are 2.

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py user.cpp
user.cpp:17: Add #include <string> for string [build/include_what_you_use] [4]
user.cpp:9: Add #include <set> for set<> [build/include_what_you_use] [4]
Done processing user.cpp
Total errors found: 2
```

Рисунок 16 – Проверка файла user.cpp

При проверке файла «user.h» утилитой «cpplint» были выявлены 2 незначительные по приоритету ошибки, связанные с включением заголовочных файлов, но данные файлы были включены на 1 уровень выше (Рисунок 16).

A terminal window showing the output of the cpplint.py utility applied to main.cpp. It identifies one error: a missing #include directive for <string>. The total errors found are 1.

```
delisa@delisa-VirtualBox ~/Documents/test/messenger $ ./cpplint.py main.cpp
main.cpp:14: Add #include <string> for string [build/include_what_you_use] [4]
Done processing main.cpp
Total errors found: 1
```

Рисунок 17 – Проверка файла main.cpp

При проверке файла «main.cpp» утилитой «cpplint» была выявлена 1 незначительная по приоритету ошибка, она связана с включением заголовочных файлов. Данная ошибка таковой не является, поскольку включение происходит на 1 уровень выше (Рисунок 17).

6.2 Тестирование программы

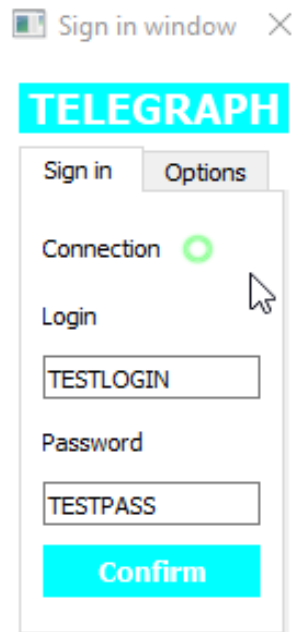


Рисунок 18 – Окно аутентификации

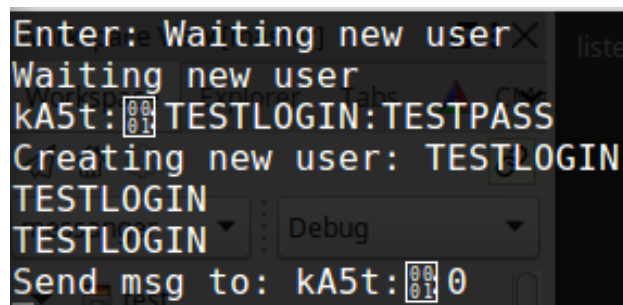


Рисунок 19 – Сообщение о аутентификации пользователя

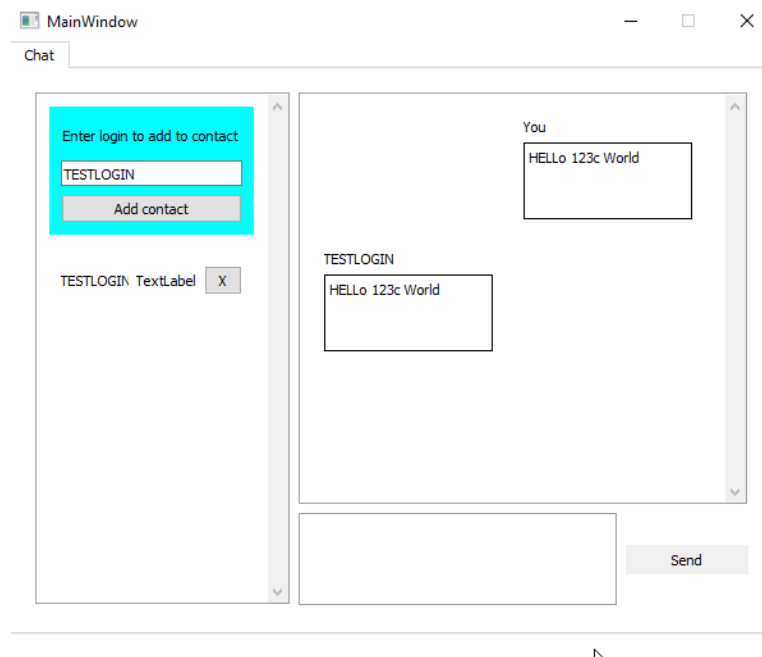


Рисунок 20 – Окно чата

```
TESTLOGIN
TESTLOGIN
Send msg to: kGvl:0
0I4f: TESTLOGIN:HELLO 123c World:TESTLOGIN
TESTLOGIN
TESTLOGIN
Send msg to: 0I4f: TESTLOGIN:HELLO 123c World
```

Рисунок 21 – Сообщение в консоли сервера о обработке запроса на проверку существования пользователя и передаче текстового сообщения

```
stop
Stopping server
STOPED
Press ENTER to continue...
```

Рисунок 22 – Остановка сервера командой

```
{
  "123": "999",
  "admin": "admin",
  "111": "888",
  "TESTLOGIN": "TESTPASS",
  "TEST": "TEST"
}
```

Рисунок 23 – Файл Users.json

```
Enter: Waiting new user>
Waiting new user
Disconnected
```

Рисунок 24 – Сообщение в консоли сервера об отключении клиента

ЗАКЛЮЧЕНИЕ

В итоге цель была достигнута полностью. В ходе работы с проектом был получен опыт в разработке крупных проектов на языке C++, а именно стандарте C++11, опыт использования коллекции библиотек boost, была изучена работа с сетью и устройство архитектуры «Клиент-сервер», также освоена работа с системами контроля версий Git и получен опыт в разработке документации и проектировании архитектуры приложения. Для оформления кода был изучен стандарт Google C++ Style Guide. Выбор языка для разработки по сути можно считать удачным, была достигнута неплохая производительность программы, но на Ruby можно было бы написать быстрее.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Google C++ Style guide [Электронный ресурс] – Режим доступа: URL: <https://google.github.io/styleguide/cppguide.html> (11.12.2017);
- 2 Boost.Asio [Электронный ресурс] – Режим доступа: URL: http://www.boost.org/doc/libs/1_65_1/doc/html/boost_asio.html (11.12.2017);
- 3 Boost.Property_tree [Электронный ресурс] – Режим доступа: URL: http://www.boost.org/doc/libs/1_65_1/doc/html/property_tree.html (11.12.2017);
- 4 Boost.Thread [Электронный ресурс] – Режим доступа: URL: http://www.boost.org/doc/libs/1_66_0/doc/html/thread.html (11.12.2017);
- 5 Std::mutex [Электронный ресурс] – Режим доступа: URL: <http://en.cppreference.com/w/cpp/thread/mutex> (11.12.2017);
- 6 Boost.Shared_ptr [Электронный ресурс] – Режим доступа: URL: http://www.boost.org/doc/libs/1_62_0/libs/smart_ptr/shared_ptr.htm (11.12.2017).

ПРИЛОЖЕНИЕ А

Исходный код

acts.h

/**

@file

@brief Заголовочный файл с описанием свободных функций для работы с пользователями

Данный заголовочный файл содержит описание функций, использующихся для работы контроля аутентификации пользователей и отправки сообщений

*/

#ifndef ACTS_H

#define ACTS_H

#include <types.h>

#include <user.h>

/**

@brief Свободная функция, реализующая аутентификацию

Данная функция проводит операцию аутентификации

@param[in] log Логин пользователя

@param[in] pass Пароль пользователя

@param[in,out] smp Пул пользователей

@param[in] sock Сокет данного пользователя

@param[in,out] ptree_ Дерево зарегистрированных пользователей

@return результат авторизации

@author Delisa

*/

```

bool auth (msg_ptr log,
           msg_ptr pass,
           user_map_ptr smp,
           socket_ptr sock,
           ptree_ptr ptree_);

/**
    @brief Свободная функция, реализующая выход пользователя из онлайн

    Данная функция обеспечивает выход данного пользователя из онлайн
    @param[in] log Логин пользователя
    @param[in,out] smp Пул пользователей

    @author Delisa
*/

void logout (msg_ptr log, user_map_ptr smp);

/**
    @brief Свободная функция, реализующая отправку сообщения

    Данная функция реализует отправку данного ей сообщения,
    пользователю, если последний существует и имеет статус онлайн,
    в пуле пользователей
    @param[in] msg Сообщение
    @param[in] login Логин адресата
    @param[in] log Пул пользователей

    @return Возвращает результат отправления сообщения

    @author Delisa
*/

bool send_ (msg_ptr msg, msg_ptr login, user_map_ptr smp);
#endif // ACTS_H

```

acts.cpp

```
#include <acts.h>
#include <iostream>

bool auth (msg_ptr log,
           msg_ptr pass,
           user_map_ptr smp,
           socket_ptr sock,
           ptree_ptr ptree_)
{
    try {
        int count = 0; ///< Количество проверенных пользователей

        for (auto it = smp->begin (); it != smp->end (); ++it) {
            if (((it->get ()->getLogin ()) == (*log)) &&
                ((it->get ()->getPass ()) == (*pass))) ||
                ((ptree_->get<std::string> ((*log))) == (*pass))) {
#ifdef DEBUG
                std::cout << "LOGGED: " << *log << std::endl;
#endif
                it->get ()->setStatus (types::Online);
                it->get ()->Notify_all ();

                return true; ///< Значит, что пользователь уже существует
            } else {
                count++;
            }
        }

        if ((count == smp->size ()) && (log->length () <= LOG_LEN) &&
            (pass->length () <= PASS_LEN)) {
#ifdef DEBUG
            std::cout << "Creating new user: " << *log << std::endl;
#endif
            boost::shared_ptr<User> tmp (
```

```

        new User (log, pass, sock, types::Online));

smp->push_back (tmp);

try {
    ptree_>put (*log, *pass);
    boost::property_tree::write_json ("Users.json", *ptree_);

} catch (boost::property_tree::ptree_bad_path) {
    std::cerr << "JSON Write error" << std::endl;

    return false;///< Ошибка чтения из файла
}
return true;
};
} catch (...) {
    return false;
}
return false;
}

```

```

void logout (msg_ptr log, user_map_ptr smp)
{
    for (auto it = smp->begin (); it != smp->end (); ++it) {
        if ((it->get ()->getLogin ()) == *log) {
            it->get ()->getSock ()->close ();
            it->get ()->setStatus (types::Offline);
            it->get ()->Notify_all ();
        }
    }
}

```

```

bool send_ (msg_ptr msg, msg_ptr login, user_map_ptr smp)
{
    int bytes = 0;

```

```

for (auto it = smp->begin (); it != smp->end (); ++it) {
    std::cout << (it->get ()->getLogin ()) << std::endl;
    if (((it->get ()->getLogin ()) == (*login)) &&
        (it->get ()->getSock ()->is_open ())) {
        std::cout << (*login) << std::endl;
        bytes = (it->get ()->getSock ())
            ->write_some (boost::asio::buffer ((*msg)));
#ifdef DEBUG
        std::cout << "Send msg to: " << *msg << std::endl;
#endif
        return bytes == sizeof ((*msg));
    }
}
return false;
}

```

Get_token.h

/**

@file

@brief Заголовочный файл с описанием свободной функции получения токена

Данный заголовочный файл содержит описание функции,
реализующей возврат токена из сообщения по его номеру

*/

#ifndef GET_BLOCK_H

#define GET_BLOCK_H

#include <types.h>

#include <boost/thread.hpp>

/**

@brief Свободная функция, реализующая возврат токена по номеру

Данная функция проводит возврат токена по номеру

@param[in] msg Сообщение

@param[in] num Номер токена

@return Возвращает токен по его номеру

@author Delisa

*/

```
msg_ptr get_token (msg_ptr msg, int num);  
#endif // GET_BLOCK_H
```

Get_token.cpp

```
#include <get_token.h>  
#include <iostream>  
msg_ptr get_token (msg_ptr msg, int num)  
{  
    std::string tmpMsg (*msg);  
    std::string tmp ("");  
    int count = 1;  
  
    for (int i = 0;; ++i) {  
        if (tmpMsg[i] == ':') {  
            ++count;  
        } else if ((count > num) || (tmpMsg[i] == '\0')) {  
            break;  
        } else if (count == num) {  
            tmp += tmpMsg[i];  
        }  
    }  
  
    msg_ptr tmp_ptr (new std::string (tmp));  
    return tmp_ptr;  
}
```

listener.h


```
/**
```

```
    @file
```

```
    @brief Заголовочный файл с описанием класса, слушающего порт
```

```
    Данный заголовочный файл содержит описание класса,  
    реализующего прием новых соединений и прослушку текущих соединений
```

```
*/
```

```
#ifndef LISTENER_H
```

```
#define LISTENER_H
```

```
#include <acts.h>
```

```
#include <get_token.h>
```

```
#include <shared_queue.h>
```

```
#include <boost/asio.hpp>
```

```
#include <boost/asio/basic_socket_acceptor.hpp>
```

```
#include <boost/asio/io_service.hpp>
```

```
#include <boost/bind.hpp>
```

```
#include <boost/property_tree/json_parser.hpp>
```

```
#include <boost/property_tree/ptree.hpp>
```

```
#include <boost/system/error_code.hpp>
```

```
#include <boost/thread/thread.hpp>
```

```
#include <system_error>
```

```
/**
```

```
    @brief Класс реализующий прием данных от пользователей
```

```
    Данный класс реализует прием новых соединений и прослушку текущих соединений,  
    а также прием данных и передачу их в очередь сообщений
```

```
    @author Delisa
```

```
*/
```

```
class listener
```

```
{
```

```
    public:
```

```
/**
```

@brief Функция-член, реализующая прослушку текущего соединения

Данная функция прослушивает соединение с определенным пользователем, принимает данные, составляет сообщение и передает в очередь соединений

@param[in] sock Сокет, выделенный для данного пользователя

@param[in,out] messageQueue очередь сообщений

@param[in,out] smp Пул пользователей

@param[in,out] ptree_ Дерево зарегистрированных пользователей

@author Delisa

*/

```
static void listen (socket_ptr sock,  
    MSG_queue_ptr messageQueue,  
    user_map_ptr smp,  
    ptree_ptr ptree_);
```

```
listener () {}
```

```
~listener () {}
```

/**

@brief Функция-член, принимающая новые соединения

Данная функция прослушивает данный ей порт, принимает новые соединения и передает управление данным соединением функции listen, выделяя ее в отдельный поток

@param[in] service Средство ввода-вывода

@param[in,out] q Очередь сообщений

@param[in] port Порт для прослушки

@param[in,out] smp Пул пользователей

@param[in,out] ptree_ Дерево зарегистрированных пользователей

@param[in,out] flag Флаг, сигнализирующей об остановке сервера

@param[in,out] listener_threads Пул для потоков, выделенных на пользователей

@author Delisa

*/

```
static void handle_connections (boost::asio::io_service* service,  
    MSG_queue_ptr q,
```

```

        int port,
        user_map_ptr smp,
        ptree_ptr ptree_,
        bool* flag,
        boost::thread_group* listener_threads);
};
#endif // LISTENER_H

```

listener.cpp

```

#include <listener.h>
#include <iostream>

void listener::listen (socket_ptr sock,
                      MSG_queue_ptr messageQueue,
                      user_map_ptr smp,
                      ptree_ptr ptree_)
{
    int bytes;
    boost::system::error_code ec;

    while (ec == 0) {
        char buff[MSG_LEN] = {'\0'};

        bytes = sock->read_some (boost::asio::buffer (buff), ec);

        if (bytes > 0) {
            msg_ptr msg (new std::string (buff));
#ifdef DEBUG
            std::cout << buff << std::endl;
#endif
            if (static_cast<int> (buff[MSG_TYPE_POS]) == types::log) {
                ptree_>put (*(get_token (msg, types::Login)),
                           *(get_token (msg, types::Data)));
            }
        }
    }
}

```

```

        auth (get_token (msg, types::Login),
              get_token (msg, types::Data), smp, sock, ptree_);
    }
    messageQueue->push (msg);
}
}

```

```

for (auto it = smp->begin (); it != smp->end (); ++it) {
    if ((it->get ()->getSock ()) == sock) {
        it->get ()->getSock ()->close ();
        it->get ()->setStatus (types::Offline);
        it->get ()->Notify_all ();
        break;
    }
}

```

```

sock->shutdown (boost::asio::ip::tcp::socket::shutdown_both, ec);
sock->close ();
}

```

```

void listener::handle_connections (boost::asio::io_service* service,
                                  MSG_queue_ptr q,
                                  int port,
                                  user_map_ptr smp,
                                  ptree_ptr ptree_,
                                  bool* flag,
                                  boost::thread_group* listener_threads)
{
    while (*flag) {
        boost::asio::ip::tcp::acceptor acceptor (
            *service,
            boost::asio::ip::tcp::endpoint (boost::asio::ip::tcp::v4 (), port));

        socket_ptr sock (new boost::asio::ip::tcp::socket (*service));
#ifdef DEBUG

```

```

        std::cout << "Waiting new user" << std::endl;
    #endif

    acceptor.accept (*sock);

    listener_threads->create_thread (
        boost::bind (listener::listen, sock, q, smp, ptree_));
    }

    return;
}

```

main.cpp

```

#include <server.h>
#include <boost/program_options.hpp>
#include <boost/program_options/options_description.hpp>
#include <boost/program_options/variables_map.hpp>
#include <iostream>
#define DEFAULT_PORT 1488 ///< Стандартный порт

void xgets (server* s_ptr)
{
    while (true) {
        char buff[256];
        std::cin.getline (buff, 256);
        std::string com (buff);
        if (com == "stop") {
            s_ptr->stop ();
            delete s_ptr;
            return;
        } else if (!s_ptr->get_flag ()) {
            return;
        }
    }
}

```

```

int main (int argc, char* argv[])
{

    int port = DEFAULT_PORT;
    server* serv = new server (port);
    boost::thread (boost::bind (xgets, serv));
    if (serv) {
        serv->run ();
    }
}

```

receiver.h

/**

@file

@brief Заголовочный файл с описанием класса, обрабатывающего сообщения

Данный заголовочный файл содержит описание класса,

реализующего обработку сообщений и вызова соответствующих функций

*/

#ifndef RECEIVER_H

#define RECEIVER_H

#include <acts.h>

#include <get_token.h>

#include <shared_queue.h>

#include <string.h>

#include <boost/asio.hpp>

#include <boost/asio/buffer.hpp>

#include <boost/thread/thread.hpp>

#include <iostream>

/**

@brief Класс реализующий обработку сообщений

Данный класс реализует обработку сообщений из очереди сообщений

и вызывает соответствующие функции

```

        @author Delisa

*/

class receiver
{
public:
    receiver () {}
    ~receiver () {}
}

/**
    @brief Функция-член, реализующая цикл чтения, обработки и ответа на сообщения

    Данная функция читает сообщения из очереди сообщений, обрабатывает их
    и генерирует соответствующий ответ
    @param[in] messageQueue очередь сообщений
    @param[in,out] smp Пул пользователей
    @param[in,out] ptree_ Дерево зарегистрированных пользователей

    @author Delisa

*/
static void loop (MSG_queue_ptr messageQueue,
                 user_map_ptr smp,
                 ptree_ptr ptree_);
};

#endif // RECEIVER_H

```

receiver.cpp

```

#include <receiver.h>

bool isExist (msg_ptr search_log,
             msg_ptr requester_User,
             user_map_ptr smp,
             ptree_ptr ptree_)

```

```

{
    try {
        std::string FindedUser = ptree_ ->get<std::string> (*search_log);

        if (FindedUser != "") {
            bool flag = false;
            socket_ptr tmp;

            for (auto it = smp->begin (); it != smp->end (); ++it) {
                if ((!flag) && ((it->get ()->getLogin ()) == *requester_User)) {
                    tmp = it->get ()->getSock ();
                    flag = true;
                }
                if (((it->get ()->getLogin ()) == *search_log) && (flag)) {
                    it->get ()->addFriend (tmp);

                    return true;
                }
            }
        }
    } catch (boost::property_tree::ptree_bad_path) {
        std::cerr << "Not Found" << std::endl;
        return false;
    }
}

void receiver::loop (MSG_queue_ptr messageQueue,
                    user_map_ptr smp,
                    ptree_ptr ptree_)
{
    boost::mutex _mutex;

    while (true) {
        if (!messageQueue->empty ()) {
            msg_ptr msg (new std::string (messageQueue->pop ().operator* ()));

```



```

if ((msg != NULL) && ((*msg) != "")) {
    bool result = false;

    if (static_cast<int> ((*msg)[MSG_TYPE_POS]) == types::msg) {
        msg_ptr msg_ (new std::string (msg->substr (
            0, msg->length () -
                (get_token (msg, types::Sender)->length ()) -
                1)));

        result = send_ (msg_, get_token (msg, types::Sender), smp);

    } else {
        msg_ptr newMsg (
            new std::string (*(get_token (msg, types::ID)) + ":"));

        if (static_cast<int> ((*msg)[MSG_TYPE_POS]) == types::iex) {
            result =
                isExist (get_token (msg, types::Login),
                    get_token (msg, types::Data), smp, ptree_);
            (*newMsg) += static_cast<char> (types::iex);

        } else if (static_cast<int> ((*msg)[MSG_TYPE_POS]) ==
            types::log) {
            result = true;
            (*newMsg) += static_cast<char> (types::log);
        }

        if (result) {
            (*newMsg) += ":0";
        } else {
            (*newMsg) += ":1";
        }

        send_ (newMsg, get_token (msg, types::Login), smp);
    }
}

```

```

    }
}
}
}
}

```

server.h

```

/**
    @file
    @brief Заголовочный файл с описанием класса сервер

    Данный заголовочный файл содержит описание класса,
    реализующего запуск и остановку сервера

*/
#ifndef SERVER_H
#define SERVER_H
#include <listener.h>
#include <receiver.h>

/**
    @brief Класс реализующий управление сервером

    Данный класс реализует запуск и остановку сервера

    @author Delisa

*/

class server
{
public:
    server (int port_)
    {
        smp.reset (new (std::list<boost::shared_ptr<User>>));
        port = port_;
    }

```

```

    flag = true;
};
~server (){};

void run ();
void stop ();
bool get_flag () { return flag; };
private:
    bool flag;
    int port;
    user_map_ptr smp;
};
#endif // SERVER_H

```

server.cpp

```

#include <server.h>
#include <iostream>
void server::run ()
{
    try {
        boost::thread_group threads;
        boost::asio::io_service service;
        MSG_queue_ptr q (new Shared_queue ());
        ptree_ptr ptree_ (new boost::property_tree::ptree);

        try {
            boost::property_tree::read_json ("Users.json", *ptree_);
            ptree_ ->put ("admin", "admin");
        } catch (...) {
            ptree_ ->clear ();
        };

        boost::thread_group* listener_threads = new boost::thread_group ();

        threads.create_thread (boost::bind (listener::handle_connections,
                                             &service, q, port, smp, ptree_,

```

```

        &flag, listener_threads));
threads.create_thread (boost::bind (receiver::loop, q, smp, ptree_));

while (flag) {
};

listener_threads->interrupt_all ();
threads.interrupt_all ();

delete listener_threads;
} catch (std::exception& e) {
    std::cerr << e.what () << std::endl;
}
}

void server::stop ()
{
#ifdef DEBUG
    std::cout << "Stopping server" << std::endl;
#endif
    boost::system::error_code errCode;
    for (auto it = smp->begin (); it != smp->end (); ++it) {
        it->get ()->getSock ()->shutdown (
            boost::asio::ip::tcp::socket::shutdown_both, errCode);
        it->get ()->getSock ()->close ();
    }
    flag = false;
}

```

shared_queue.h

```
/**
```

```
    @file
```

```
    @brief Заголовочный файл с описанием класса синхронизированной очереди
```

Данный заголовочный файл содержит описание класса,
реализующего синхронизированную работу с очередью

*/

```
#ifndef SHARED_QUEUE_H
#define SHARED_QUEUE_H
#include <types.h>
#include <mutex>
#include <queue>
```

/**

@brief Класс реализующий синхронизированную очередь

Данный класс реализует синхронизированную работу с очередью

@author Delisa

*/

```
class Shared_queue
```

```
{
```

```
public:
```

```
    Shared_queue (){};
```

```
    ~Shared_queue (){};
```

/**

@brief Функция-член, реализующая добавление элемента в конец очереди

Данная функция добавляет элемент в конец очереди

@param[in] msg Сообщение

@author Delisa

*/

```
void push (msg_ptr msg);
```

/**

@brief Функция-член, реализующая изъятие первого элемента из очереди

Данная функция возвращает копию первого элемента из очереди
и удаляет его из очереди

@return Возвращает сообщение

@author Delisa

*/

```
msg_ptr pop ();
```

/**

@brief Функция-член, реализующая проверку очереди на пустоту

Данная функция возвращает true, если очередь пуста
или false в ином случае

@return возвращает true, если очередь пуста или false в ином случае

@author Delisa

*/

```
bool empty () { return data.empty (); };
```

```
private:
```

```
std::mutex mutex_;//< Мьютекс для синхронизации очереди
```

```
std::queue<std::string> data;//< Очередь для хранения сообщений
```

```
};
```

```
typedef boost::shared_ptr<Shared_queue> MSG_queue_ptr; ///< Очередь сообщений
```

```
#endif // SHARED_QUEUE_H
```

shared_queue.cpp

```
#include <shared_queue.h>
```

```

msg_ptr Shared_queue::pop ()
{
    while (mutex_.try_lock ()) {
        };

    msg_ptr msg (new std::string (data.front ()));

    data.pop ();
    mutex_.unlock ();

    return msg;
}
void Shared_queue::push (msg_ptr msg)
{
    while (mutex_.try_lock ()) {
        };

    data.push (*msg);
    mutex_.unlock ();
}

```

types.h

```

/**
    @file
    @brief Заголовочный файл с описанием основных типов, инклюдов,
    перечислимых типов и констант

    Данный заголовочный файл содержит описание основных типов, инклюдов,
    перечислимых типов и констант
*/
#ifndef TYPES_H
#define TYPES_H
#include <string.h>
#include <boost/asio/ip/tcp.hpp>

```

```

#include <boost/property_tree/json_parser.hpp>
#include <boost/property_tree/ptree.hpp>
#include <set>

#define MSG_LEN 256 ///< Максимальная длина сообщения
#define MSG_TYPE_POS 5 ///< Позиция бита типа сообщения
#define MSG_ID_LEN 4 ///< Длинная идентификатора сообщения

#define LOG_LEN 16 ///< Максимальная длина логина
#define PASS_LEN 16 ///< Максимальная длина пароля

typedef boost::shared_ptr<std::string> msg_ptr; ///< Сообщение
typedef boost::shared_ptr<boost::property_tree::ptree> ptree_ptr; ///< Дерево зарегистрированных пользователей
typedef boost::shared_ptr<boost::asio::ip::tcp::socket> socket_ptr; ///< Сокет
namespace types
{
    ///< Возможные типы сообщений
    enum msgType {

        log = 1, ///< Авторизоваться
        reg = 2, ///< Зарегистрироваться
        iex = 4, ///< Проверка на существование
        ion = 8, ///< Проверка на статус онлайн
        msg = 16 ///< Переслать сообщение

    };
    ///< Возможные номера токенов
    enum Tokens {
        ID = 1, ///< Идентификатор сообщения
        Type = 2, ///< Тип сообщения
        Login = 3, ///< Логин
        Data = 4, ///< Данные(Пароль/текст сообщения)
        Sender = 5 ///< Логин отправителя
    };
};

```



```

/// Возможные статусы пользователей
enum Status {
    Online = 0, ///< Пользователь в сети
    Offline, ///< Пользователь не в сети
    NotAuth ///< Пользователь еще не прошел аутентификацию
};
}

#endif // TYPES_H

```

user.h

```

/**
    @file
    @brief Заголовочный файл с описанием класса пользователя

    Данный заголовочный файл содержит описание класса,
    реализующего хранение и работу с данными пользователя
*/
#ifndef USER_H
#define USER_H
#include <types.h>
#include <list>

/**
    @brief Класс реализующий поведение пользователя

    Данный класс реализует хранение и работу с данными пользователя

    @author Delisa
*/

class User
{
public:

```

```
User (msg_ptr Log, msg_ptr Pass, socket_ptr sock_, types::Status status_);  
~User ();
```

```
/**
```

```
    @brief Функция-член, возвращающая логин пользователя
```

```
    Данная функция возвращает логин пользователя
```

```
    @return Возвращает логин пользователя
```

```
    @author Delisa
```

```
*/
```

```
std::string getLogin () { return Login; }
```

```
/**
```

```
    @brief Функция-член, возвращающая пароль пользователя
```

```
    Данная функция возвращает пароль пользователя
```

```
    @return Возвращает пароль пользователя
```

```
    @author Delisa
```

```
*/
```

```
std::string getPass () { return Password; }
```

```
/**
```

```
    @brief Функция-член, возвращающая сокет пользователя
```

```
    Данная функция возвращает сокет пользователя
```

```
    @return Возвращает сокет пользователя
```

```
    @author Delisa
```

```
*/
```

```
socket_ptr getSock () { return sock; }
```

/**

@brief Функция-член, возвращающая статус пользователя

Данная функция возвращает статус пользователя

@return Возвращает статус пользователя

@author Delisa

*/

```
types::Status getStatus () { return status; };
```

/**

@brief Функция-член, устанавливает статус пользователя

Данная функция устанавливает статус пользователя

@param[in] status_ Новый статус

@author Delisa

*/

```
void setStatus (types::Status status_) { status = status_; }
```

/**

@brief Функция-член, добавляет пользователя, которого необходимо оповещать об изменении статуса данного пользователя

Данная функция добавляет нового пользователя в множество пользователей, которых необходимо оповещать об изменении статуса данного пользователя

@param[in] Friend Сокет пользователя

@author Delisa

*/

```
void addFriend (socket_ptr Friend) { Friends->insert (Friend); }
```

```
/**
```

```
@brief Функция-член, возвращающая множество пользователей,  
которых необходимо оповещать об изменении статуса данного пользователя
```

```
Данная функция возвращает множество пользователей,  
которых необходимо оповещать об изменении статуса данного пользователя  
@return Возвращает множество пользователей (сокетов)
```

```
@author Delisa
```

```
*/
```

```
std::set<socket_ptr>* getFriends () { return Friends; }
```

```
/**
```

```
@brief Функция-член, оповещает всех пользователей из множества Friends
```

```
Данная функция оповещает всех пользователей из множества Friends,  
об изменении статуса данного пользователя
```

```
@author Delisa
```

```
*/
```

```
void Notify_all ();
```

```
private:
```

```
std::string Login; ///< Логин пользователя
```

```
std::string Password; ///< Пароль пользователя
```

```
socket_ptr sock; ///< Сокет пользователя
```

```
types::Status status; ///< Статус пользователя
```

```
std::set<socket_ptr>* Friends; ///< Множество пользователей, которых необходимо оповещать
```

```
};
```

```
typedef boost::shared_ptr<std::list<boost::shared_ptr<User>>> user_map_ptr; ///< Пул пользователей
```

```
#endif // USER_H
```

user.cpp

```
#include <user.h>

User::User (msg_ptr Log,
            msg_ptr Pass,
            socket_ptr sock_,
            types::Status status_)
{
    Login = *Log;
    Password = *Pass;
    sock = sock_;
    status = status_;
    Friends = new std::set<socket_ptr>;
}

User::~~User () { delete Friends; }

void User::Notify_all ()
{
    for (auto it = Friends->begin (); it != Friends->end (); ++it) {
        char tmp = static_cast<char> (status);

        std::string* newMsg = new std::string (Login + ":" + tmp, MSG_LEN);
        it->get ()->write_some (boost::asio::buffer (*newMsg));
        delete newMsg;
    }
}
```

ПРИЛОЖЕНИЕ Б

Git log

Commit 15694fe476539e578bab9de089c9d63599a26f4c

Author: Delisa <delisa.sama@gmail.com>

Date: Mon Dec 11 15:13:12 2017 +0300

Stable 1.1 Doc

commit d3ab9e2afc6eb98c8a278991ee3813807c4a94ba

Author: Delisa <delisa.sama@gmail.com>

Date: Sun Dec 10 02:08:58 2017 +0300

Stable v1.1

commit 619b81596bd5484f4b1a288aa1b60ade04c5126a

Author: Delisa <delisa.sama@gmail.com>

Date: Fri Dec 8 09:51:07 2017 +0300

Stable v1.0.1

commit 2f748070005590554833a385c7a9f25a957338ec

Author: Delisa <delisa.sama@gmail.com>

Date: Tue Dec 5 21:36:33 2017 +0300

V3.0 Stable

commit 271123b260d118df2b32a69bfc66fd2ef2586177

Author: Delisa <delisa.sama@gmail.com>

Date: Tue Dec 5 00:53:55 2017 +0300

Semi-Worked

commit aa349ef2a5707bd1f12e6c370e3a712d9b8ac4b9

Author: Delisa <delisa.sama@gmail.com>

AuthorDate: Mon Dec 4 16:45:32 2017 +0300

Commit: Delisa <delisa.sama@gmail.com>

CommitDate: Mon Dec 4 16:45:32 2017 +0300

Prototype v.2.1

commit d3bb2a030e316455a0341835d5215ff11c873f87

Author: Delisa <delisa.sama@gmail.com>

AuthorDate: Sun Nov 26 23:23:29 2017 +0300

Commit: Delisa <delisa.sama@gmail.com>

CommitDate: Sun Nov 26 23:23:29 2017 +0300

Prototype v.2.0

commit a965c9772ec6929e43f4af617d57578637fcdb9c

Author: Delisa <delisa.sama@gmail.com>

AuthorDate: Fri Nov 24 16:56:08 2017 +0300

Commit: Delisa <delisa.sama@gmail.com>

CommitDate: Fri Nov 24 16:56:08 2017 +0300

Prototype v.1.3

commit 88debfdff7fa0729e81dcbc2a26b27bbdfb27a75

Author: Delisa <delisa.sama@gmail.com>
AuthorDate: Fri Nov 24 01:01:37 2017 +0300
Commit: Delisa <delisa.sama@gmail.com>
CommitDate: Fri Nov 24 01:01:37 2017 +0300

Prototype v.1.2

commit 80b9b09fb5c4174cfe3c64947c6c6e9945d10890
Author: Delisa <delisa.sama@gmail.com>
AuthorDate: Wed Nov 22 21:44:44 2017 +0300
Commit: Delisa <delisa.sama@gmail.com>
CommitDate: Wed Nov 22 21:44:44 2017 +0300

Prototype v.1.1 Fix

commit a1ca5a3d8e7ffe801cd26ee6ae76c5af4dd6adca
Author: Delisa <delisa.sama@gmail.com>
AuthorDate: Wed Nov 22 21:17:36 2017 +0300
Commit: Delisa <delisa.sama@gmail.com>
CommitDate: Wed Nov 22 21:17:36 2017 +0300

Prototype v.1.1

commit 035a6aebad42c0f09a58b5547531075f7dde6812
Author: Delisa <delisa.sama@gmail.com>
AuthorDate: Sat Nov 18 18:52:28 2017 +0300
Commit: Delisa <delisa.sama@gmail.com>
CommitDate: Sat Nov 18 18:52:28 2017 +0300

Prototype v.1

commit efebcdcd9e5db7739c9300dab081af0571bc54e9

Author: Delisa <delisa.sama@gmail.com>

AuthorDate: Sat Nov 18 17:57:06 2017 +0300

Commit: Delisa <delisa.sama@gmail.com>

CommitDate: Sat Nov 18 17:57:06 2017 +0300

Initial commit

ПРИЛОЖЕНИЕ В

Issues

Issues (1–12 of 12)

Title	T	P	Status	Votes	Assignee	Created	Updated
#5: Google C++ Style Guide			RESOLVED		Delisa	2017-11-18	5 days ago
#2: Утечки			RESOLVED		Delisa	2017-11-18	2017-12-06
#12: boost::exception_ptr			RESOLVED		Delisa	2017-12-05	2017-12-06
#4: Онлайн пользователи			RESOLVED		Delisa	2017-11-18	2017-12-03
#7: Хранение набора классов User			RESOLVED		Delisa	2017-11-24	2017-12-03
#10: Список отслеживающих пользователей			RESOLVED		Delisa	2017-11-24	2017-11-26
#11: Проход по коллекции пользователей			RESOLVED		Delisa	2017-11-26	2017-11-26
#6: Аутентификация			RESOLVED			2017-11-24	2017-11-26
#8: Проверка регистрации			RESOLVED		Delisa	2017-11-24	2017-11-26
#9: Оповещения			RESOLVED		Delisa	2017-11-24	2017-11-26
#3: Логика reciever'a			RESOLVED		Delisa	2017-11-18	2017-11-22
#1: server.stop			RESOLVED		Delisa	2017-11-18	2017-11-22

Рисунок 25 – Issues

ПРИЛОЖЕНИЕ Г

Руководство оператора

1 Об этом документе

Настоящий документ – руководство для оператора сервера мессенджера. Документ описывает работу оператора с программной частью. Цель документа – дать читателю понятие об общих элементах управления и детальное руководство к выполнению всех видов операций.

Документ состоит из трех частей:

- Первая глава – ознакомительная. Она дает понять оператору структуру документа, назначения приложения, приводятся технические требования.
- Вторая глава – содержательная. В данной главе объясняется принцип работы боты программы.
- Третья глава – описывает реакции приложения на действия оператора.
- Она необходима для устранения ошибок во время работы.
- Документ составлен в соответствии с требованиями стандарта ГОСТ 19.505-79 «Руководство оператора. Требования к содержанию и оформлению».

1.2 Назначение программы

Данная программа предназначена для предоставления возможности текстового общения клиентами в сети интернет. Данное приложение реализовано с помощью программных средств, которые предоставляет коллекция библиотек Boost на языке C/C++. Предназначено для развертывания на компьютерах, работающих под управлением и Linux.

1.3 Цели создания программы

Целью создания данной программы является дать операторам возможность текстового общения в сети интернет, в удобной форме.

1.4 Условия выполнения программы

1.4.1 Технические условия для оборудования оператора:

Для выполнения основной функции, клиент должен быть подключен к той же

сети, что и сервер.

2 Выполнение программы

2.1 Запуск программы

Запуск сервера осуществляется также, как и запуск любого другого приложения на компьютере через файл исполнения.

2.2 Консоль сервера

В данной консоли будут появляться сообщения, отображающие текущие события, происходящие на сервере. В любой момент можно ввести команду «stop». Приложение сервер приостановит все свои потоки и закроет соединения с клиентами.

3 Сообщения оператору

- Waiting a new user – Обозначает, что сервер ждет подключения нового пользователя;
- Сообщения типа [КОД]:[ТИП]:[ЛОГИН]:[ПАРОЛЬ] – обозначает, что сервер принял запрос на аутентификацию;
- Creating new user: [ЛОГИН] – обозначает, что сервер создает нового пользователя;
- Send msg to: [СООБЩЕНИЕ] – отправка сообщения;
- Сообщения типа [КОД]:[ТИП]:[ЛОГИН]:[СООБЩЕНИЕ] – обозначает, что сервер принял сообщение от клиента;

ПРИЛОЖЕНИЕ Д

Руководство программиста

1 Назначение и условия применения программы

1.1 Назначение программы

Данная программа предназначена для предоставления возможности текстового общения клиентами в сети интернет. Данное приложение реализовано с помощью программных средств, которые предоставляет коллекция библиотек Boost на языке C/C++. Предназначено для развертывания на компьютерах, работающих под управлением и Linux.

1.2 Функции, выполняемые программой

Целью данной программы является дать клиентам возможность текстового общения в сети интернет, в удобной форме.

1.3 Условия необходимые для запуска

Запуск сервера осуществляется также, как и запуск любого другого приложения на компьютере через файл исполнения.

2 Характеристика программы

2.1 Режим работы программы

Режим программы круглосуточный непрерывный.

2.2 Контроль правильности выполнения программы

Работоспособность данной программы можно проверить путем любого взаимодействия с клиентами с сервером или написанием команд.

3 Обращения к программе

Обращение к программе детально описано в руководстве пользователя.

4 Входные и выходные данные

Входные данные адресованные пользователю описаны в руководстве пользователя, данные же адресованные программисту выводятся в стандартный поток вывода и содержат сообщения, которые строятся на протяжении выполнения программы. Они нужны для отладки программы программистом и видны оператору.

Действия, которые необходимо предпринимать зависят от программиста и не нуждаются в методологии.

5 Сообщения

5.1 Программа вывела череду сообщений(формульный вид):

- 1234:(тип сообщения):[логин:[логин:]]сообщение
- 1234:(тип сообщения):ответ

Данная чередка сообщений, где символы квадратных скобок обозначают опциональную часть сообщения, которая зависит от его типа, означает то, что было отправлено сообщение и получен на него ответ, также присутствуют сообщения которые не имеют заранее отправленное сообщение, такие как просто текст, отправленный одним оператором другому. Реакция на такие сообщения зависят от программиста и не нуждаются в методологии.

ПРИЛОЖЕНИЕ Е
Доклад и презентация