МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети

БАКАЛАВРСКАЯ РАБОТА

на тему

Система дистанционной ревизии кода

Студент _____ (подпись, дата) Колесников Вадим Сергеевич

Руководитель _____ (подпись, дата) Дорошенко И.Н.

Нормоконтролёр 18.06.19 (подпись, дата) А.В. Кучин

Работа допущена к защите (протокол зас. кафедры от 31.05.19 г. № 38-09-14)

Заведующий кафедрой _____ (подпись) М.А. Митрохин

Работа защищена с отметкой отлично (протокол заседания ГЭК от 19.06.19 г. № 7)

Секретарь ГЭК _____ (подпись) Е.И. Гурин

Пенза 2019

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«23» 04 2019 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Колесников Вадим Сергеевич гр. 15ВВ1 факультета вычислительной техники

2. Тема работы: «Система дистанционной ревизии кода»

Тема утверждена приказом ПГУ № 554/0 от "29" 04 2019 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ Дорошенко И.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п).
Назначение разработки – система, предназначенная для удалённого рецензирования программного кода в режиме реального времени. Система должна обеспечивать редактирование файлов, их просмотр и комментирование. Под данными терминами следует понимать создание, открытие, редактирование, удаление файла со стороны писателя. Просмотр файла и возможность оставлять комментарии со стороны рецензента.

5. Перечень подлежащих разработке вопросов.
Необходимо провести анализ существующих решений, требований к разрабатываемой системе. Выбрать методы и средства разработки системы. Определить порядок разработки компонентов системы. Произвести отладку и тестирование и оформить выпускную квалификационную работу.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководите
1.	Изучение предметной области	10	01.05.19	
2.	Анализ существующих решений	5	06.05.19	
3.	Анализ методов и средств разработки	10	07.05.19	
4.	Реализация основных компонентов системы	25	15.05.19	
5.	Тестирование и отладка	20	28.05.19	
6.	Оформление выпускной квалификационной работы	30	04.06.19	

Дата выдачи задания "29" 04 2019 г.

Руководитель бакалаврской работы _____ Дорошенко И.Н.
(подпись, дата)

Задание к исполнению принял студент _____ Колесников В.С.
(подпись, дата)

Работу к защите допустить
Декан факультета 18.06.2019 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Бакалаврская работа содержит 58 страниц, 37 рисунков, 13 источников, 2 приложения.

Ключевые слова: программирование, клиент-сервер, ревизия, писатель, рецензент, код.

Структура работы включает в себя введение, 3 главы, заключение, список использованной литературы и приложение.

Цель работы: реализовать систему дистанционной ревизии кода.

Достижение поставленной цели требует решения следующих задач:

1) провести анализ предметной области на примере существующих систем и платформ, обеспечивающих рецензирование кода;

2) выполнить анализ требований к разрабатываемой системе;

3) определить методы и средства, используемые при разработке компонентов системы дистанционной ревизии кода;

4) определить порядок разработки компонентов системы.

В ходе проделанной работы были изучены основные методы повышения качества программного кода и методы разработки оконных приложений MicrosoftWindows. Разработана клиентская часть, обеспечивающая возможность ревизии файлов с программным кодом и их редактирование. Также, разработано серверное приложение, хранящее данные файлы и обеспечивающее доступ к ним. Получены навыки программирования и разработки программ на языкеC# в среде MicrosoftVisualStudio.

СОДЕРЖАНИЕ

1 Характеристика объекта разработки и сбор теоретических сведений 9
1.1 Современные подходы к улучшению программного кода 9
1.1.1 Просмотр кода 10
1.1.2 Парное программирование 12
1.2 Обзор существующих решений 15
1.2.1 Google документы 15
1.2.2 Floobits 16
1.3 Описание архитектуры приложения 19
1.3.1 Клиент-серверная архитектура 20
1.3.2 Обмен данными между узлами 24
1.3.3 Сетевая модель OSI 24
2 Описание процесса разработки системы 27
2.1 Реализациявзаимодействияклиента и сервера 27
2.2 Разработка клиентского приложения 29
2.3 Разработка серверного приложения 36
3 Эксплуатация разработанной системы 39
3.1 Подключение к серверу 40
3.2 Режим «Писатель» 41
3.3 Режим «Рецензент» 46
3.4 Автономный режим 52
Заключение 55
Список использованных источников 57
Приложение А: Листинг программы клиента 59
ПриложениеА.1 – Файл «Writer.cs» 59
ПриложениеА.2 – Файл «BuildWriterRequest.cs» 61
ПриложениеА.3 – Файл «WriterRequest.cs» 64
ПриложениеА.4 – Файл «Program.cs» 66

ПриложениеА.5 – Файл «Reviewer.cs» 67
ПриложениеА.6 – Файл «BuildReviewerRequest.cs» 69
ПриложениеА.7 – Файл «ReviewerRequest.cs» 71
ПриложениеА.8 – Файл «OperationResult.cs» 74
ПриложениеА.9 – Файл «Commands.cs» 74
Приложение Б: Листинг программы сервера 76
ПриложениеБ.1 – Файл «Server.cs» 76
ПриложениеБ.2 – Файл «TheClient.cs» 77
ПриложениеБ.3 – Файл «ClientRequest.cs» 78
ПриложениеБ.4 – Файл «FileProcessing.cs» 83
ПриложениеБ.5 – Файл «OperationResult.cs» 86
ПриложениеБ.6 – Файл «Commands.cs» 87

ВВЕДЕНИЕ

Сегодня трудно представить повседневную жизнь человека без компьютерных технологий. С помощью технологий человек может избавиться от рутинных действий в любой сфере деятельности. В современном мире существует множество программных продуктов. Программные продукты могут быть различными - от простых утилит и программ для домашнего использования добаз данных, хранящихразличного рода информацию о человеке.

В процессе работы любого программного обеспечения могут возникать различного рода ошибки. Ошибка в работе драйвера компьютерной мыши в большинстве случаев не вызовет серьёзных проблем. Случаи, когда выходит из строя программное обеспечение, отвечающее за подачу топлива в двигателе пассажирского самолёта, могут привести к материальным или даже человеческим жертвам.

Согласно вышесказанному, можно сделать вывод, что качество программного обеспечения должно быть высоким, а процесс его разработки должен находиться под наблюдением. Под качеством программного обеспечения подразумевается то, насколько оно удовлетворяет предъявляемым к нему требованиям, зависящим от области применения данного программного обеспечения.

Целью данной работы является разработка системы, позволяющей проводить рецензирование программного кода, а также, обеспечивать возможность редактирования файлов. Система должна позволить проводить рецензию программного кода в режиме реального времени. Пользователь, именуемый как «писатель», работает с файлом и имеет на сервере неразделяемый с другими пользователями каталог для файлов. Пользователь,

именуемый как «рецензент», имеет возможность подключаться к серверу, выбирать писателей для просмотра, а также, вносить комментарии к их программному коду.

Данная разработка призвана упростить процесс ревизии программного кода, который может быть применен в различных областях, в том числе и в образовательных целях.С помощью такой разработки станетудобно работать в удалённом режиме, набрав небольшую группу учеников. Обозреватель сможет контролировать процесс написания кода писателями в режиме реального времени, внося комментарии по мере необходимости. В дополнение к разрабатываемой системе подойдут такие приложения, как Skype, Discord и прочие, поддерживающие аудио или видеосвязь.

Следовательно, разрабатываемая в данной работе система является актуальной и позволит повысить качество программного кода.

К системе предъявлены такие требования, как возможность создания нового файла, открытия файла, изменения файла, удаления файла. Для рецензента должна быть реализована возможность внесения комментариев к коду одного или нескольких писателей, а также, наблюдение за процессом редактирования кода в режиме реального времени. Это означает, что вносимые писателем изменения в файле должны отображаться у всех подключённых к данному писателю рецензентов.

Для достижения поставленной цели необходимо провести анализ существующих решений, технологий для реализации поставленной задачи, инструментов разработки, а затем реализовать данную систему с использованием выбранных технологий и провести тестирование, по итогам которого исправить выявленные недочёты.

1 ХАРАКТЕРИСТИКА ОБЪЕКТА РАЗРАБОТКИ И СБОР ТЕОРЕТИЧЕСКИХ СВЕДЕНИЙ

Данная глава посвящена детальному изучению современныхподходов к улучшению программного кода, проведению анализа существующих, на рынке решений изучаемой проблемы. В этой главе, также, внимание уделяется, разработке структуры приложения, которая включает в себя описание клиент-серверной архитектуры и рассмотрение вопроса взаимодействия компьютеров в сети.

1.1 Современные подходы к улучшению программного кода

Для решения поставленной задачи необходимо рассмотреть современные подходы к улучшению программного кода.

Существует множество способов и методик, направленных на повышение качества кода, которые можно условно разделить на две группы:

1. Программные методы
2. Непрограммные методы

Программные методы подразумевают сканирование кода специальными программами, позволяющими выявлять ошибки. Простейшем примером такой программы является компилятор языка программирования. Существуют, также, различные утилиты для проверки кода на соответствие определённым стандартам языка.

Под непрограммными подразумеваются подходы, требующие, помимо разработчика, участия дополнительных лиц, работающих над улучшением качества программного кода. Данные подходы требуют дополнительных

человеческих ресурсов, но они являются наиболее эффективными в настоящее время.

В данной работе выделены два подхода к улучшению качества программного кода:

Ревизия кода (другие названия: инспекция кода, просмотр кода)

Парное программирование

1.1.1 Просмотр кода

Просмотр кода (англ. codereview) — систематическая проверка исходного кода программы с целью обнаружения и исправления ошибок, которые остались незамеченными в начальной фазе разработки [1].

Просмотр кода также можноусловно поделить на два вида:

Формальный

Неформальный

При формальном просмотре кода координатор назначает дату проведения собрания, секретаря и инспекторов, которые перед собранием должны изучить код. Во время собрания один из инспекторов или координатор зачитывают строчки кода по порядку. Если у кого-то из инспекции возникают вопросы, они задают их автору зачитываемого кода. Если все согласны, что ошибка имеется – секретарь записывает её. Координатор контролирует процесс проведения мероприятия в целом. По окончании выявления ошибок составляется отчёт с результатами по просмотренному коду.

Для организации данной процедуры чаще всего требуется больше двух человек, синхронизация каждого из них по времени и отвлечение от основной

работы на время проведения мероприятия. Но несмотря на всю сложность, данный подход является наиболее эффективным.

При неформальной организации просмотра кода достаточно попросить напарника или одного из коллег посмотреть написанный код. Проверяющий должен вникнуть в суть программы, высказать свои соображения или предложить поправки. Данный метод не требует большого количества проверяющих и занимает меньше времени, но и эффективность ниже.

Причин для проведения ревизий программного кода можно выделить достаточно много. Ниже приведены некоторые из них:

1. Стимулирование разработчика.

Программист, делающий работу, детали которой никто не увидит, с большей вероятностью допустит ошибку. Он может сделать ошибку временно, забыть, а может и вовсе не видеть проблемы, которые возникнут позже. При проведении ревизий программного кода разработчик знает, что его код будет просмотрен другим человеком и начинает допускать меньше ошибок. В данном случае влияние оказывает психологический фактор – человек хочет показать себя с лучшей стороны.

2. Экономия времени на ошибках.

После просмотра кода рецензент, как правило, указывает строку в программном коде, где была допущена та или иная ошибка. Исправление данной ошибки происходит быстрее, так как нет необходимости тратить время на её поиски, особенно если данный участок кода был написан неделями или месяцами ранее.

1.1.2 Парное программирование

Существует методология программирования, называемая экстремальным программированием. Данная методология направлена на повышение качества разрабатываемого продукта (и программного кода, в том числе), ускорение обучения разработчиков, а также, позволяет добиться доверия от заказчиков. Включает множество разновидностей экстремального программирования, в частности – парное программирование.

Парное программирование — техника программирования, при которой исходный код создаётся парами людей, программирующих одну задачу, сидя за одним рабочим местом[6].

При классическом стиле парного программирования два программиста сидят за одним компьютером. Один из них пишет программный код, а второй в это время наблюдает, анализирует действия напарника. В процессе работы у разработчика, наблюдающего за процессом написания кода, могут возникать вопросы разного характера. Решая данные вопросы, пара разработчиков получает на выходе более качественный продукт, содержащий меньше ошибок и недочётов.

Данный метод имеет две разновидности:

Пинг-понг программирование

Удалённое (дистанционное) парное программирование

Пинг-понг программирование — разновидность парного программирования, адаптированная для метода разработки через тестирование[5].

Разработка через тестирование— техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов

разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец код подводится к соответствующим стандартам[6].

Основная мысль такого подхода состоит в написании тестов к программному коду двумя разработчиками. Один из них пишет код, а второй пишет тест. Затем напарники меняются ролями.

Удалённое парное программирование– это разновидность парного программирования, в котором два программиста находятся на расстоянии друг от друга[2], работая через средство совместного редактирования кода, удалённый рабочий стол, или посредством расширения для интегрированной системы разработки.

Парное программирование имеем достаточно много положительных сторон некоторые из которых приведены ниже:

Обучение

В процессе работы в паре программисты непрерывно обмениваются опытом. Использование новых комбинаций горячих клавиш, разговоры о полезных расширениях, удобных текстовых редакторах технологиях помогают развиваться профессионально.

Знания о проекте

Благодаря частой смене пар разработчики знают устройство проектов или всей системы. Это позволяет лучше понимать логику данных проектов, держать единый стандарт разработки, а новые кадры быстрее вникают в работу.

Наставничество

Постановка в пару разработчиков с разным профессиональным уровнем позволяет обучать более слабого.

Концентрация

Разработчики, работая в паре, проводят больше времени в состоянии концентрации, не отвлекаясь на проверку почты, социальных сетей и новостных лент. Это экономит время на восстановление концентрации на решаемой задаче, которое занимает в среднем 25 минут[13].

Улучшение отношений в коллективе

Постоянное общение в процессе работы позволяет лучше узнать характер напарника, его привычки, сильные и слабые стороны. Это главным образом поддерживает «живую атмосферу», позволяет работникам больше общаться, поддерживая необходимый для профессионального и личностного роста обмен информацией.

Несмотря на всю свою эффективность, парное программирование имеет некоторые недостатки.

Во-первых, существует мнение, что процесс написания кода в паре, когда один программист пишет код, а другой следит за этим процессом, гораздо медленнее, чем если бы работа велась в одиночку. Однако, как показывает практика, данный метод, в большинстве случаев, медленнее на 15% или не отличается по времени[12].

Во-вторых, данные преимущества будут работать только при соблюдении определённых правил. Не стоит давать задачу паре, если её с гораздо большей эффективностью можно решить в одиночку. Также, не стоит ставить в пару людей, которые друг другу не нравятся, а при составлении пары хотя бы один из программистов должен иметь опыт в парном программировании.

Таким образом, разрабатываемая система основана на двух методологиях улучшения программного кода, которые рассмотрены выше, а именно – ревизии кода и парном программировании.

1.2 Обзор существующих решений

На данный момент на рынке существует множество удобных сред разработки, сервисов и текстовых редакторов, содержащих удобный пользовательский интерфейс, подсветку синтаксиса для множества языков программирования и прочие функции. Однако ассортимент приложений, позволяющий проводить ревизию кода удалённо, в режиме реального времени, достаточно мал.

Ниже рассмотрены некоторые сервисы и приложения, приведены их положительные и отрицательные стороны с точки зрения дистанционной ревизии кода.

1.2.1 Google документы

Данный сервис представляет собой текстовый процессор, позволяющий работать с документами нескольким пользователям одновременно.

Текстовый процессор— компьютерная программа, используемая для написания и модификации документов, компоновки макета текста и предварительного просмотра документов в том виде, в котором они будут напечатаны[10].

Редактирование пользователями осуществляется после предоставления определённых прав от владельца документа. В настройках доступа есть возможность указать, кто может работать с документом и каким образом: редактировать, комментировать или просматривать. В режиме доступа по

ссылке, к документу может получить доступ любой, кто данной ссылкой владеет.

Важной особенностью данного сервиса является возможность редактирования документа в режиме реального времени с нескольких компьютеров. Сервис позволяет добавлять комментарии к абзацам, которые видны и доступны для обсуждения другим пользователям (см. рис. 1).

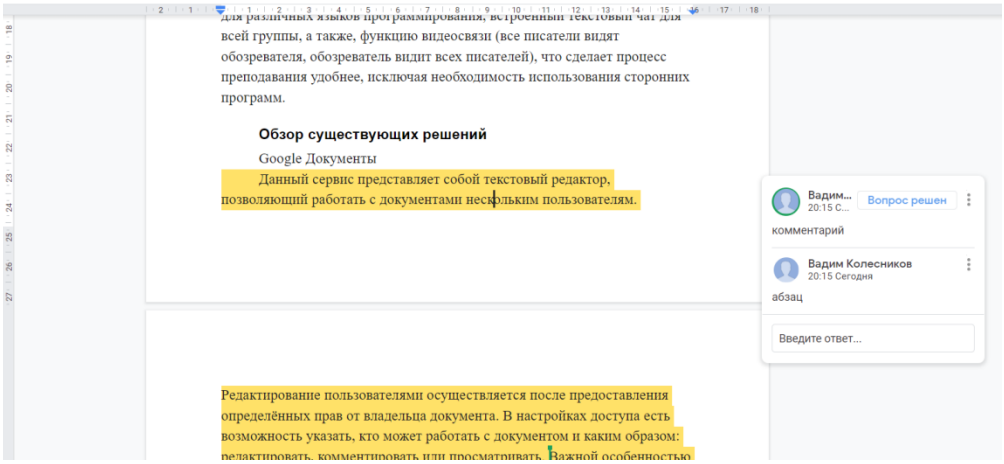


Рисунок 1 - Комментарии google документы

Поскольку данный сервис представляет собой текстовый редактор, он имеет множество инструментов для форматирования текста, но не имеет подсветки синтаксиса для языков программирования, а также позволяет сохранять документ только в текстовых форматах, таких как PDF, DOCX или TXT.

1.2.2 Floobits

Данный сервис представляет собой полноценную платформу для совместного написания программного кода с возможностью связать аккаунт с репозиторием на GitHub. Для совместного редактирования программного кода необходимо создать рабочее пространство и добавить пользователя в данное пространство. Редактирование возможно проводить в браузере. Существует

специальное расширение FloobitsScreenSharing для браузера GoogleChrome, позволяющее вести общение с помощью веб-камеры в процессе работы. Для использования данного расширения необходимо иметь право редактирования. Изображения участников рабочего пространства расположены справа от поля редактирования, в левой части экрана находится иерархия папок и файлов, доступных для редактирования, а снизу расположен онлайн-чат, позволяющий проводить рецензирование в форме текстовых сообщений (см. рис. 2).

В данном проекте реализована возможность создания рабочих пространств. Одно пространство позволяет работать над одним проектом.

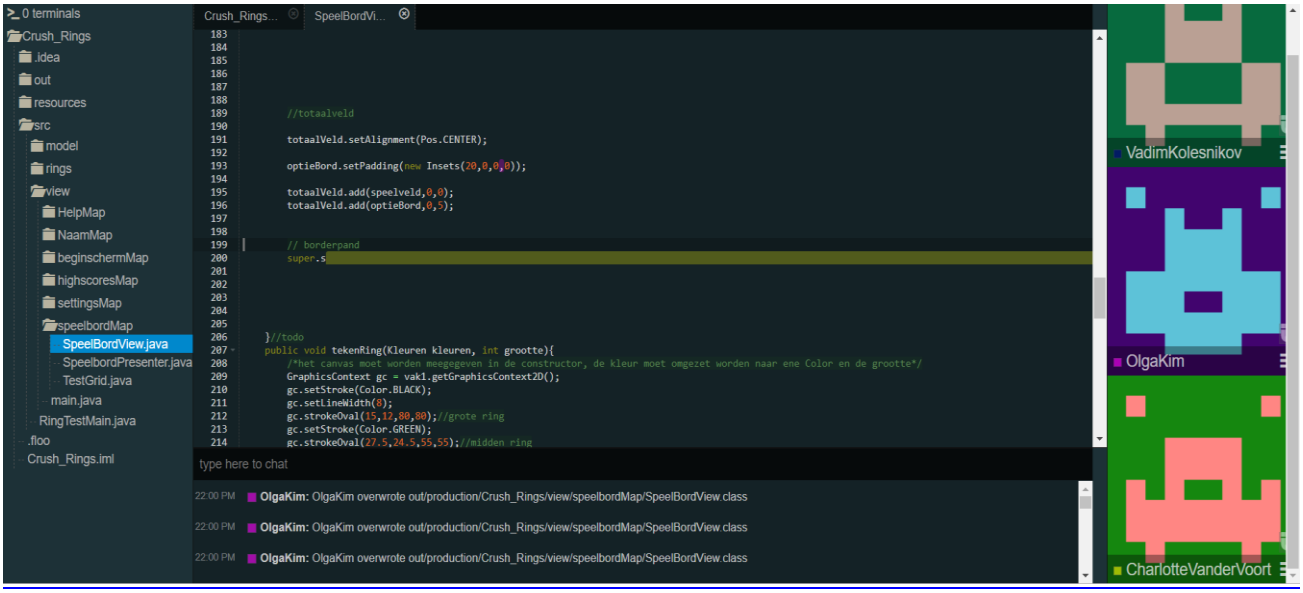


Рисунок 1 - Вид текстового редактора Floobits

Существуют расширения для нескольких наиболее популярных текстовых редакторов, таких как SublimeText 3, Atom, Emacs, IntellijIDEA и Neovim, также позволяющие совместно редактировать программный код и вести непрерывное рецензирование.

Несмотря на все вышеперечисленные преимущества, проект является платным. При бесплатном использовании имеется возможность создать лишь до

пяти рабочих пространств, которые являются публичными. Это означает, что весь проект доступен к просмотру любому пользователю, зарегистрированному на данном портале. Для присоединения к публичному рабочему пространству достаточно знать его название и имя пользователя, которому это пространство принадлежит. Например, имя пользователя – VadimKolesnikov, название рабочего пространства – MySpace. Тогда, соединив последовательно имя пользователя и название рабочего пространства в адресной строке после адреса сайта получим следующий URL-адрес: https://floobits.com/VadimKolesnikov/MySpace/. Вставив его в адресную строку браузера получим желаемый результат (см. рис. 3).

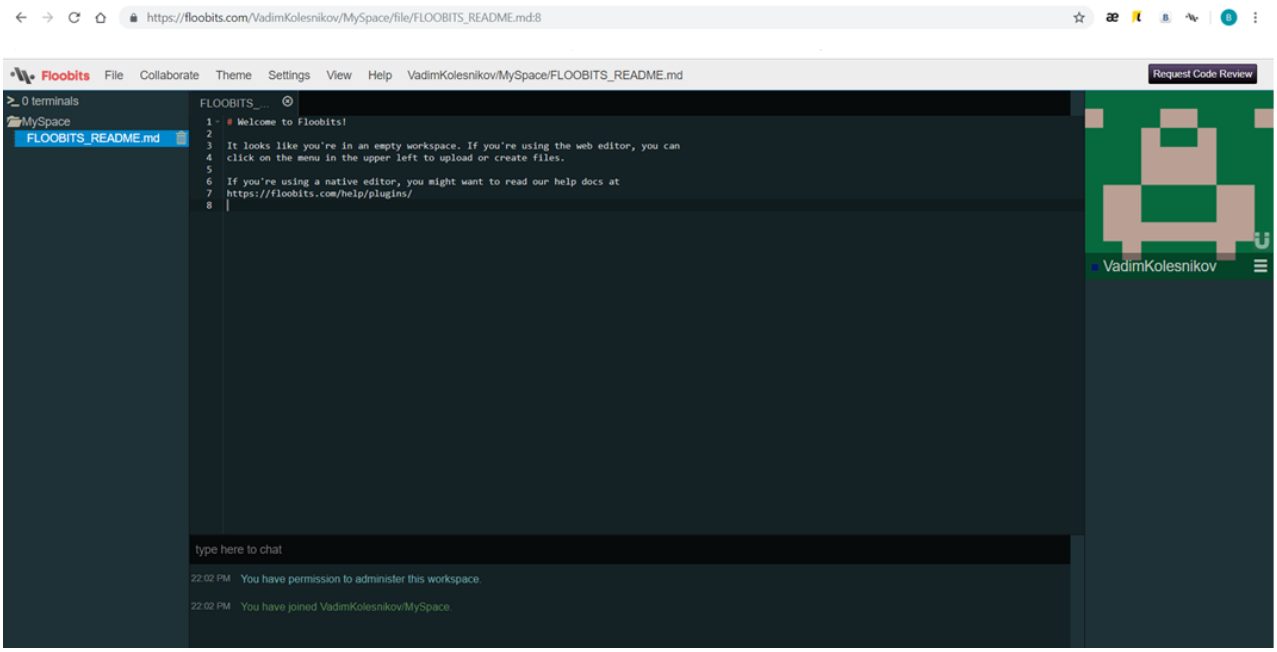


Рисунок 2–Публичное рабочее пространство

При создании приватного рабочего пространства возникает необходимость приобретать другой тарифный план (см. рис. 4):

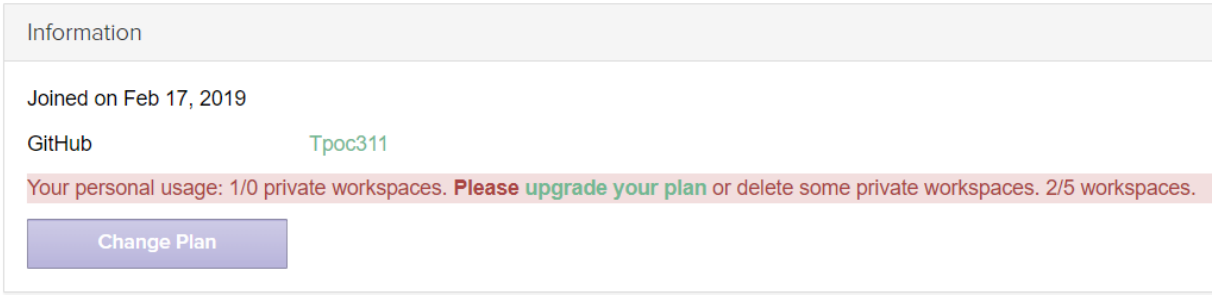


Рисунок 3 - Сообщение после создания приватного рабочего пространства

В противном случае никто не будет иметь доступ к созданному приватному рабочему пространству, кроме владельца. Ниже приведена стоимость увеличения приватных рабочих пространств (см. рис. 5):

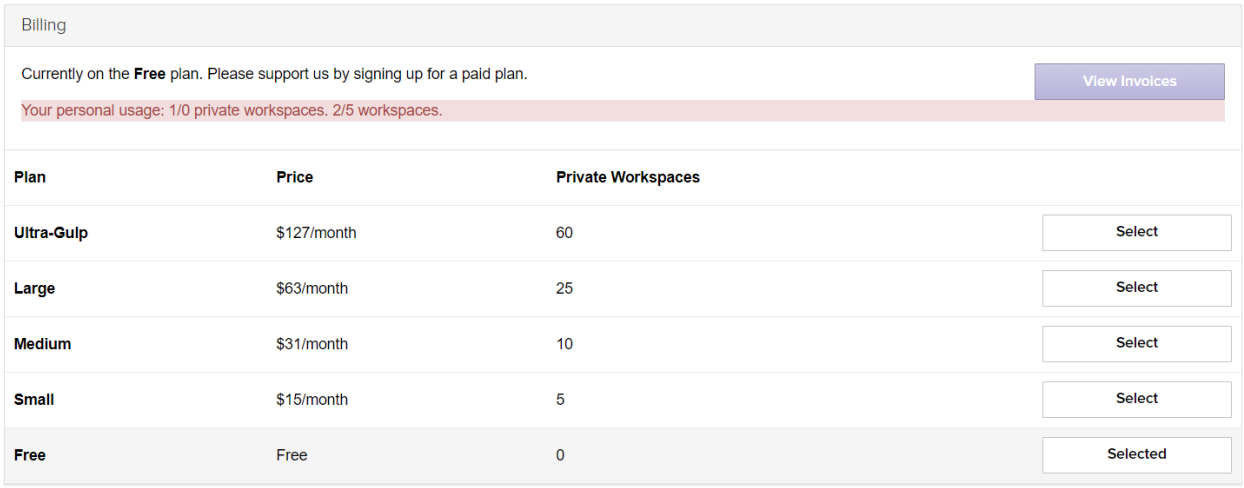


Рисунок 4 - Стоимость увеличения количества приватных рабочих пространств

Таким образом, разрабатываемый проект позволит устранить приведённые выше недостатки аналогов, а именно, будет адаптирован для редактирования и рецензирования программного кода и не будет содержать платного контента.

1.3 Описание архитектуры приложения

Система должна содержать единое хранилище данных, а пользователи должны иметь доступ к этим данным. Следовательно, необходимо рассмотреть

клиент-серверную архитектуру, а также, изучить вопрос о взаимодействии клиентов и сервера между собой.

1.3.1 Клиент-серверная архитектура

Исторически первой появилась модель, которая реализовывалась на универсальной ЭВМ с подключенными к ней терминалами. Управление данными и взаимодействие с пользователями происходили в одной программе. В дальнейшем, с появлением персональных компьютеров и локальных сетей, подход к хранению и обработке данных, а также, взаимодействие с пользователями изменились.

Как правило, компьютеры и программы, входящие в состав какой-либо системы, не являются равноправными. Чаще всего в системе находятся один и более компьютеров, имеющих в своём распоряжении определённые ресурсы (ресурсы могут принимать различный облик – от файлов до периферийных устройств). Такие компьютеры называются серверами. Остальные компьютеры могут получить доступ к этим данным посредством сервера. Такая модель взаимодействия (или по-другому архитектура) называется клиент-серверной или «клиент-сервер».В настоящее время клиент и сервер – это физически разные ЭВМ, но под клиентами и серверами, также, следует понимать программное обеспечение, установленное на этих ЭВМ.

«Клиент — сервер» — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами[7].

В системе может быть несколько серверов. Каждый сервер выполняет определённые функции. Например, сервер баз данных реализует выполнение

запросов клиентов на обработку данных, а файловый сервер распоряжается файловой системой.

Клиентская программа и серверная программа разделяют между собой функции, представленные ниже:

Функции ввода и отображения данных;

Прикладные функции, характерные для данной предметной области;

Функции хранения и управления информационно-вычислительными ресурсами (базами данных, файламии т.д.).

Существуют две основные архитектуры взаимодействия клиент-серверных приложений, внутри которых распределяются приведённые выше функции:

Двухзвенная;

Трёхзвенная.

На рисунке 6 изображена двухзвенная архитектура:

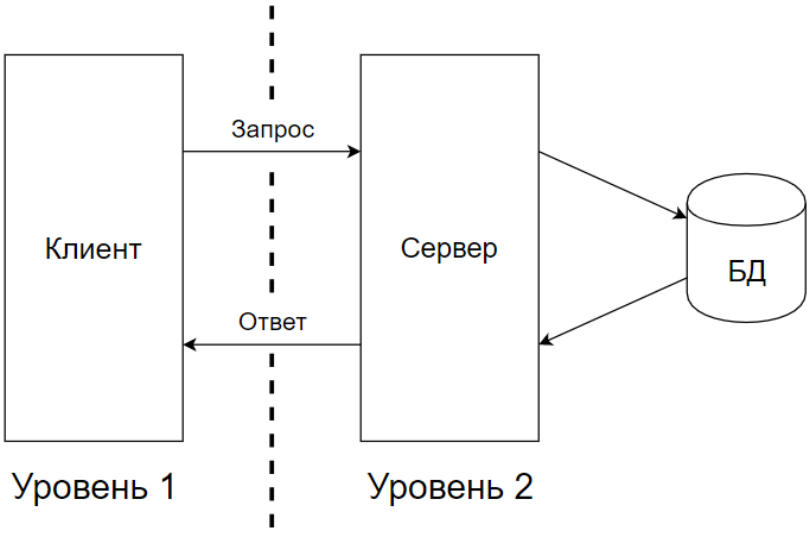


Рисунок 5 - Двухзвенная архитектура

Особенность данной архитектуры состоит в том, что выполнениеприкладных функций ложится на сервер, либо распределяется между сервером и клиентом. Это предъявляет дополнительные требования к производительности клиентов и сервера. В то же время такая архитектура является очень надёжной. На рисунке видно, что данная архитектура разделена на 2 уровня. На первом уровне расположено приложение клиента, позволяющее пользователю сделать необходимый запрос. На втором уровне находится сервер, выполняющий запросы к базе данных (БД).

Ниже приведена трёхзвенная архитектура (см. рис. 7):

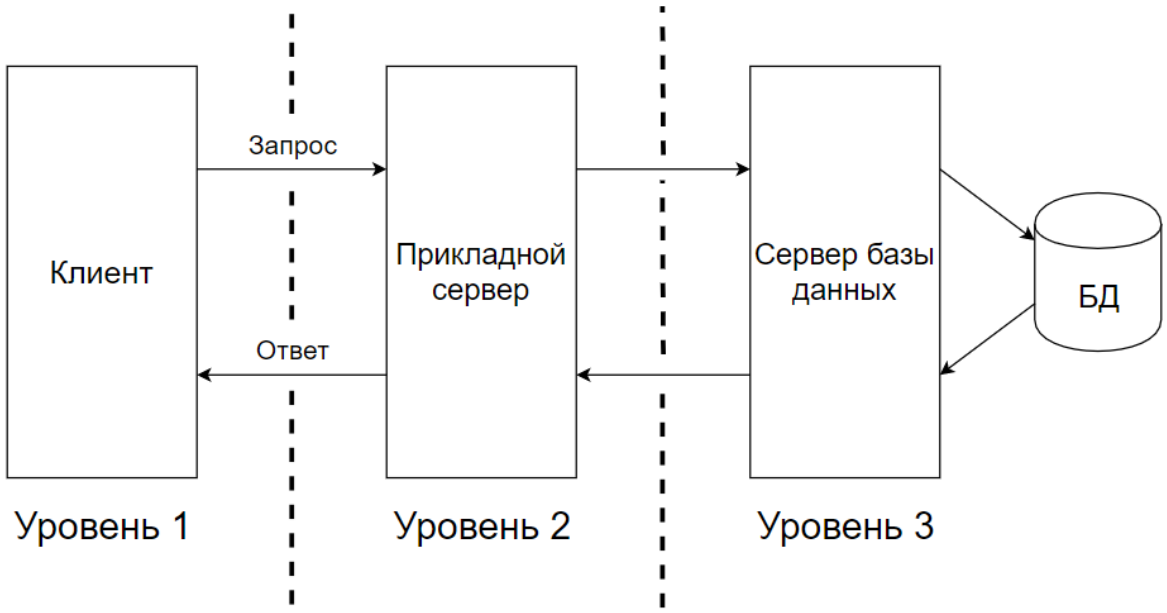


Рисунок 6 - Трёхзвенная архитектура

Данная архитектура отличается от двухзвенной тем, что содержит прикладной сервер. Данный сервер призван уменьшить нагрузку на остальные звенья.Чаще всего, прикладной сервер выполняет необходимые вычисления над данными, полученными от сервера базы данных. Тогда клиент осуществляет только работу с пользователем и отправку запросов, а сервер базы данных

только обработку этих запросов. Остальные функции возложены на прикладной сервер. Трёхзвенная архитектура – частный случай многозвенной архитектуры. Многозвенная архитектура отличается наличием одного или нескольких дополнительных серверов, которые так же выполняют свои функции.

Различают множество типов серверов. Некоторые из них представлены ниже:

Web-серверы;

Серверы баз данных;

Файл-серверы;

Прокси-серверы;

Почтовые серверы.

Web-серверы предназначены для доступа к гипертекстовым документам, но в настоящее время обеспечивают дополнительные функции, например, работа с мультимедиа-файлами.

Серверы баз данных реализуют формирование и отправку запросов к базе данных.

Файл-сервер представляет собой сервер, предназначенный для выполнения файловых операций ввода-вывода и хранящий файлы любого типа. Как правило, обладает большим объемом дискового пространства, реализованном в форме RAID-массива для обеспечения бесперебойной работы и повышенной скорости записи и чтения данных.

Прокси-сервер действует как посредник между клиентом и сервером, помогая пользователям получать информацию из сети. Часто содержат локальный кэш, позволяющий получить запрашиваемую информацию без повторного обращения к сети интернет.

Почтовые серверы обеспечивают отправку и получение электронных почтовых сообщений.

Модель файл-сервер изображена на рисунке 8:

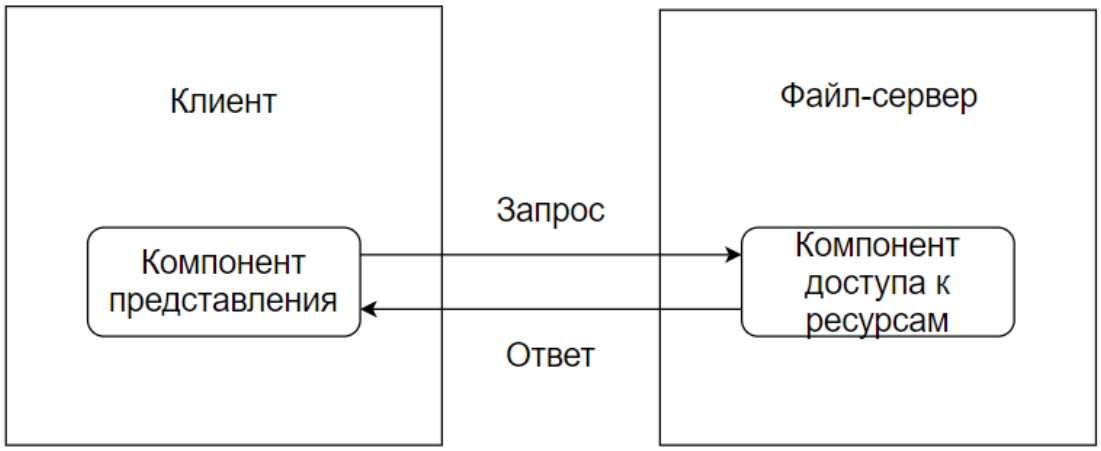


Рисунок 7 – Модель файл-сервер

1.3.2 Обмен данными между узлами

Чтобы обеспечить обмен данными между узлами, необходимо связать их между собой. Данная связь, как правило, обеспечивается путём подключения компьютеров к сети интернет. В сети интернет обмен данными происходит посредством специальных правил — протоколов.

— набор правил и действий (очерёдности действий), позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть устройствами[8].

1.3.3 Сетевая модель OSI

Сетевая модель OSI(opensystemsinterconnectionbasicreferencemodel— Базовая Эталонная Модель Взаимодействия Открытых Систем (ЭМВОС)) —

сетевая модель стека (магазина) сетевых протоколов OSI/ISO (ГОСТ Р ИСО/МЭК 7498-1-99) [9].

Посредством данной модели различные сетевые устройства могут взаимодействовать друг с другом. Модель определяет различные уровни взаимодействия систем. Каждый уровень выполняет определённые функции при таком взаимодействии (см. рис. 9):

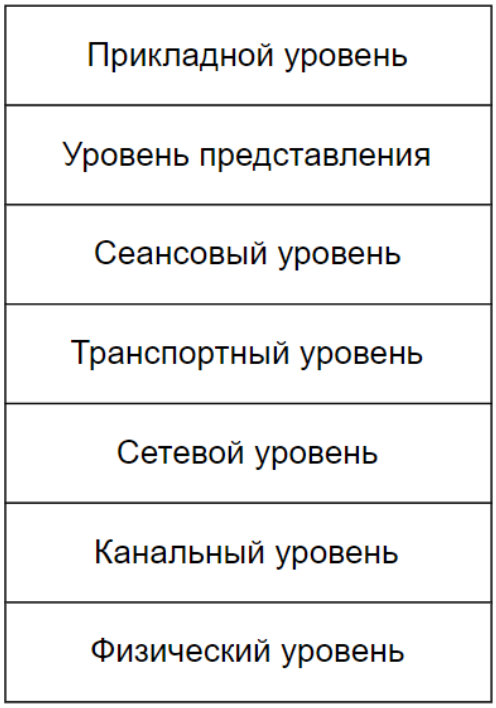


Рисунок 8 – Сетевая модель OSI

Прикладной уровень связывает пользовательские приложения с сетью. На каждом уровне действуют разные протоколы, выполняющие различные функции. Например, для просмотра веб-страниц используется протокол HTTP, при передаче и приёме почты – SMTP, POP3, для приёма и получения файлов – FTP, TFTP и т.д.

Уровень представления содержит протоколы преобразования данных, обрабатываемых на других уровнях.

На сеансовом уровне организуется сеанс связи между компьютерами. Например, устанавливается сеанс аудио или видеосвязи, выбирается способ кодирования сигнала.

Транспортный уровень отвечает за факт доставки сообщения на другой компьютер. На этом уровне выделены протоколы, отвечающие за доставку сообщений по сети. Примером таких протоколов могут служить TCP и UDP протоколы.

На сетевом уровне расположены транспортные протоколы, отвечающие за маршрут доставки данных. На этом уровне находятся такие протоколы, как ICMP, IP, RIP и т.д.

На канальном уровне располагаются протоколы и устройства, отвечающие за обмен данными между устройствами в одном сегменте локальной сети. Примерами протоколов, работающих на данном уровне являются Ethernet, Point-to-PointProtocol (PPP), HDLC.

Физический уровень является самым нижним уровнем и отвечает за передачу потока данных. Протоколы на данном уровне: Bluetooth, Wi-Fi, витая пара и т.д.

2 ОПИСАНИЕ ПРОЦЕССА РАЗРАБОТКИ СИСТЕМЫ

Разрабатываемая система делится на две основные программы – приложение клиента и серверная программа. Для реализации поставленной задачи выбрана среда разработки MSVisualStudio. Клиентская часть реализуется на основе компонента WindowsForms. Серверная часть – консольное приложение. Приложения разрабатываются на объектно-ориентированном языке программирования C#. Клиентское и серверное приложения разрабатываются и тестируются одновременно.

Порядок реализации следующий. Сначала реализуется автономный режим для клиента, обеспечивающий базовые функции тестового редактора и сервер с функцией подключения\отключения клиентов. Далее реализуется графический интерфейс для режима «Писатель», механизм соединения с сервером, а затем механизм редактирования файлов. Последним этапом реализуется режим «Рецензента» в том же порядке, что и режим писателя – графический интерфейс, подключение\отключение к серверу, механизм рецензирования.

2.1 Реализациявзаимодействияклиента и сервера

В процессе анализа поставленной задачи было принято решение об использовании двухзвенной архитектуры. Данное решение связано с тем, что прикладные функции были распределены между клиентом и сервером, поскольку не предъявляют высоких требований к производительности вычислительных машин.

Все файлы пользователей хранятся на сервере. Для каждого нового пользователя на сервере формируется отдельный каталог файлов, поэтому к данным файлам имеет доступ на редактирование только один писатель.

Таким образом, схема взаимодействия клиентов с сервером приобретает следующий вид (см. рис. 10):

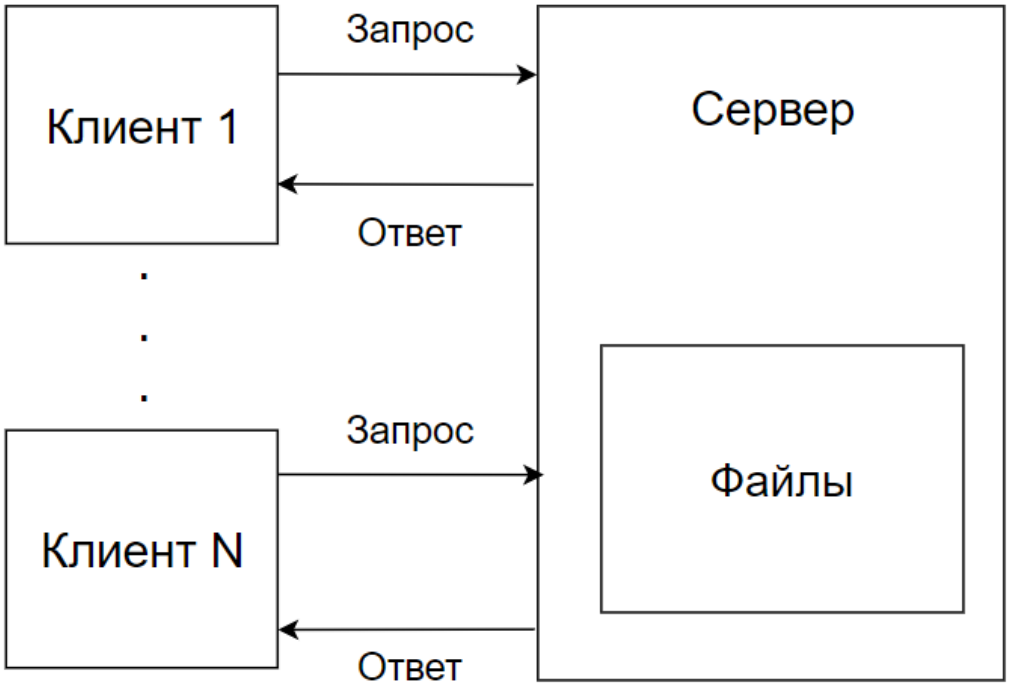


Рисунок10 – Архитектура разрабатываемого приложения.

Обмен данными между клиентом и сервером происходит посредством стека протоколов TCP/IP. Это обусловлено тем, что в процессе работы системы необходимо обеспечить гарантированную доставку данных при передаче между клиентом и сервером.

Таким образом, в данной работе функции хранения и управления файлами возложены на серверную часть, а отображения данных и их ввод происходит на клиентской части.

2.2 Разработка клиентского приложения

Клиентскую часть приложения было решено разбить на три составляющие. Это связано с тем, что функционал для режима рецензента и режима писателя достаточно отличается. Также, было принято решение создать автономныйрежим. (см. рис. 11):

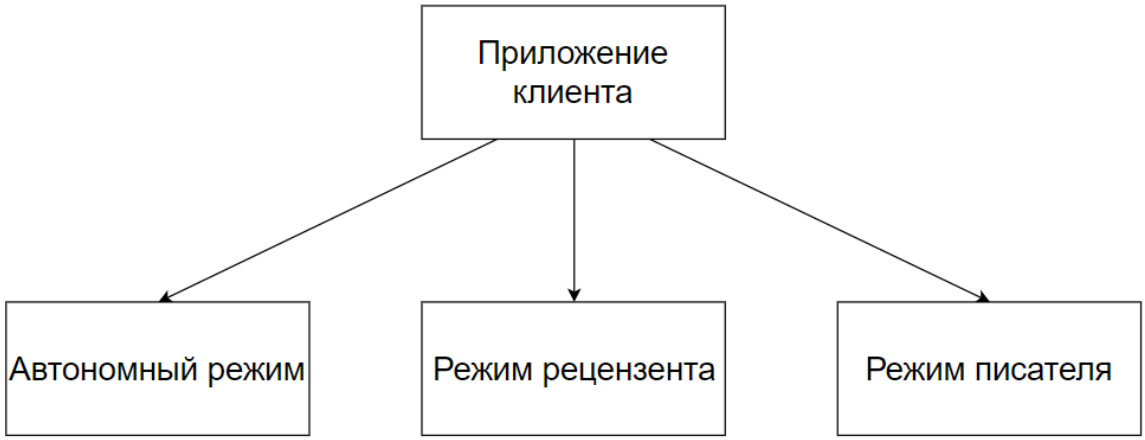


Рисунок 9 - Структура клиентского приложения

Каждая из составляющих частей является в виде отдельного компонента WindowsForms.

Автономный режим должен обеспечивать базовые возможности текстового редактора. Таким образом, функции данного режима следующие:

создание;

открытие файла;

редактирование файла;

сохранение файла.

Приведённые выше функции работают только в локальном режиме, то есть позволяют работать с файлами только на локальном компьютере.

Режим писателя позволяет работать с файлами непосредственно на сервере. Данный режим поддерживает следующие функции:

создание файла;
открытие файла;
редактирование файла;
сохранение файла;
удаление файла.

Режим рецензента позволяет работать с подключенными к серверу писателями. Данный режим поддерживает следующие функции:

выбор нового писателя;
наблюдение за работой писателя;
комментирование работы писателя.

Главное назначение данного режима – наблюдение за писателем в ходе редактирования конкретного файла и комментирование его работы. Возможно наблюдение за несколькими писателями в одно время.

Данный режим реализуется с помощью двух компонентов WindowsForms.

Первый компонент – главная форма, которая запускается при подключении к серверу. Второй компонент может запускаться в нескольких экземплярах. Количество запускаемых форм напрямую зависит от количества работающих на сервере писателей. Один писатель – одна форма.

Схема режима рецензента представлена на рисунке 12:

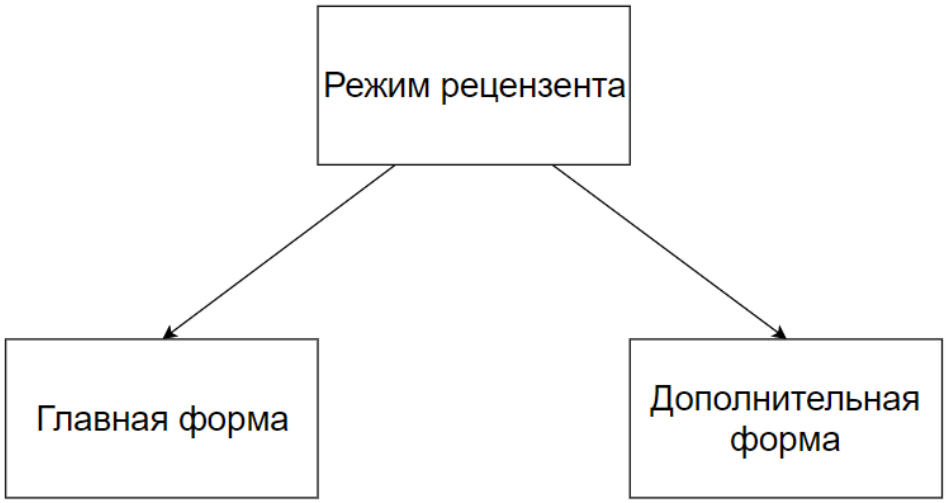


Рисунок 10 - Режим рецензента

Для наблюдения за двумя и более писателями реализована возможность открытия дополнительной формы. Необходимые для этого элементы расположены в главной форме. При закрытии главной формы закрываются все дополнительные и происходит отключение от сервера.

Диаграмма вариантов использования для клиентской части приведена на рисунке 13:

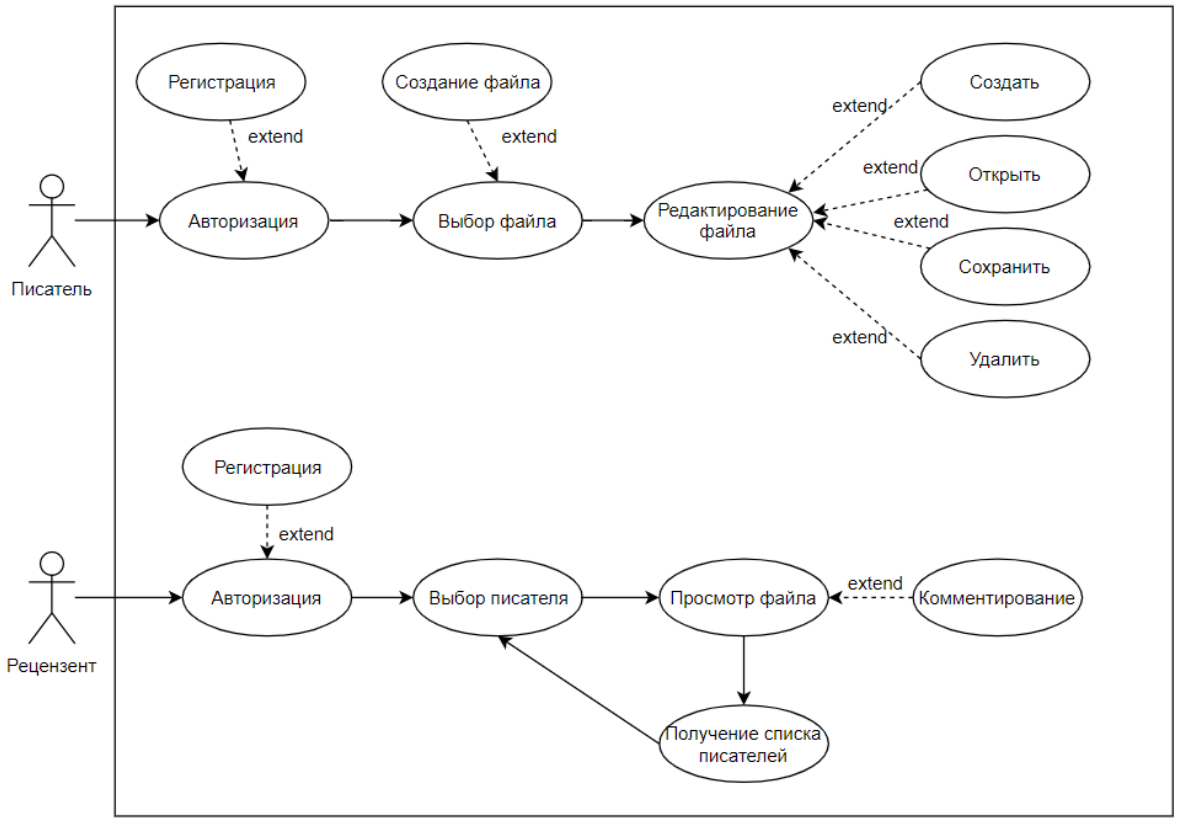


Рисунок13 – Диаграмма вариантов использования для клиентской части

Диаграмма состояния файла показана на рисунке 14:

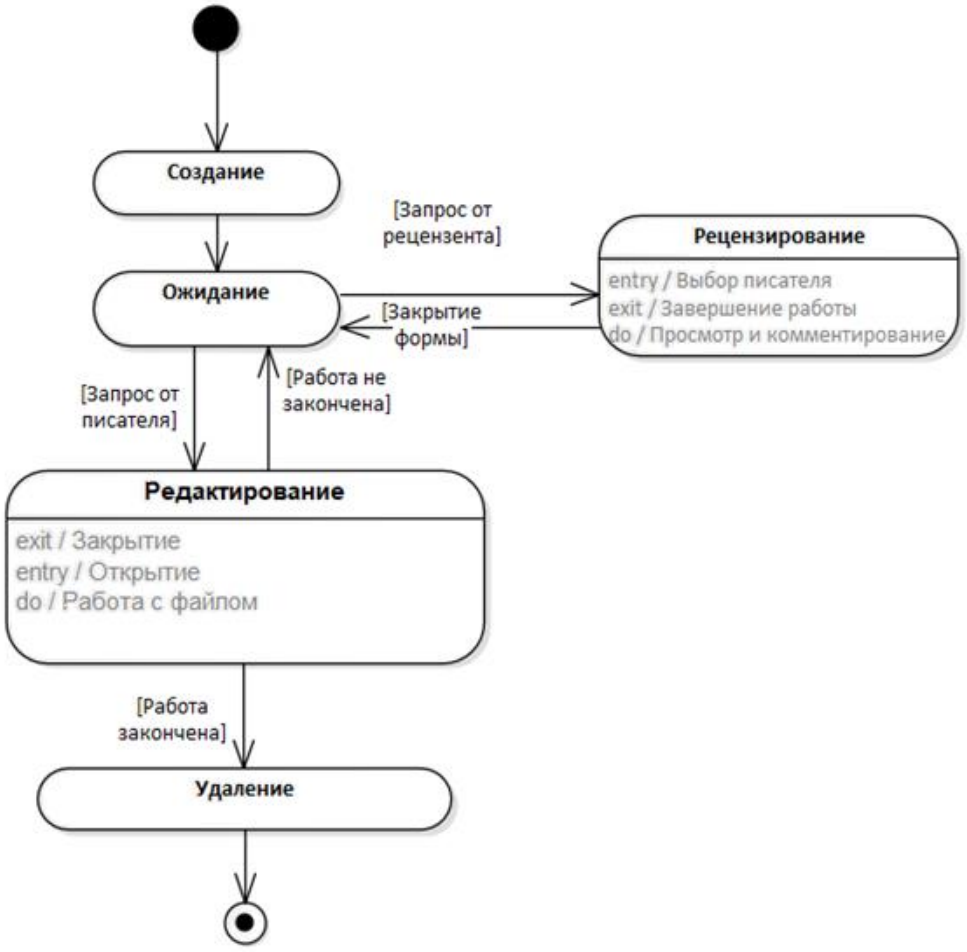


Рисунок 14 – Диаграмма состояния файла

Процесс рецензирования происходит следующим образом. Писатель редактирует файл. В это время рецензенты могут подключаться или отключаться от писателя посредством сервера. В процессе редактирования файла (внесение или удаление данных) изменения отправляются на сервер, а затем рассылаются всем подключенным к писателю рецензентам.

Рецензент видит отображение всех изменений файла писателя на своей форме. В поле «Комментарии» вносится строка текста и отправляется по нажатию «Enter» на сервер (см. рис. 15). С сервера строка перенаправляется писателю.

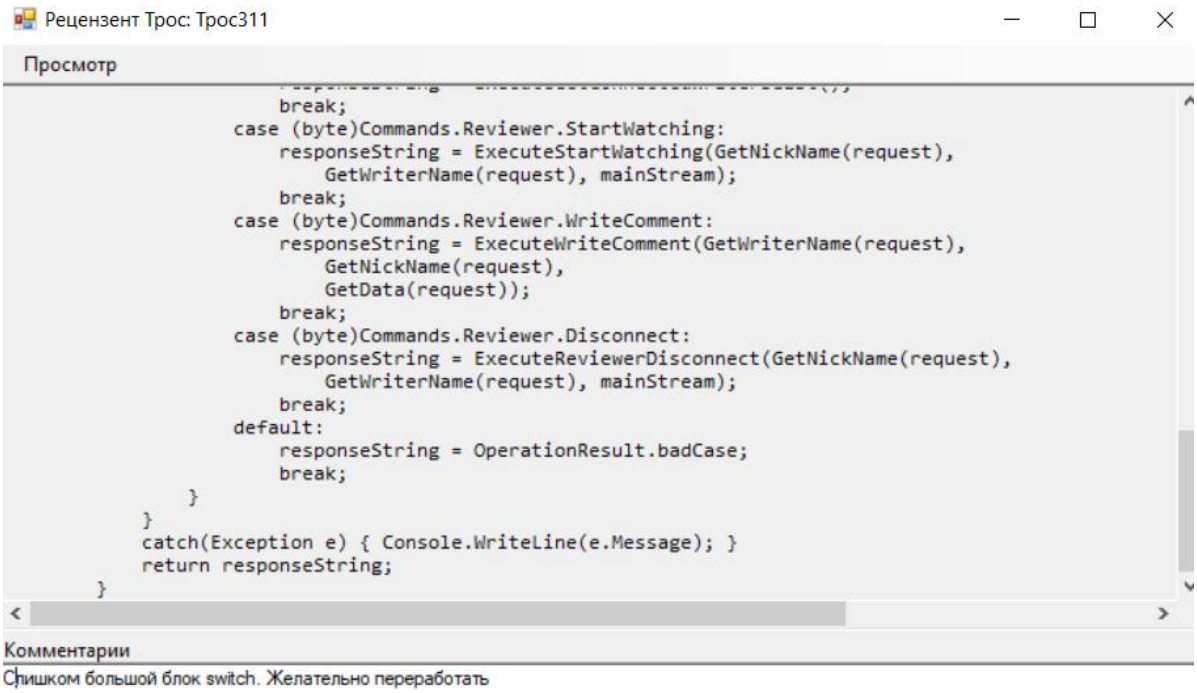


Рисунок15 – Пример рецензирования

Писатель не может редактировать окно комментариев, а рецензент не может редактировать окно отображения изменений.

Диаграмма деятельности для клиентской части приложения приведена на рисунке 16:

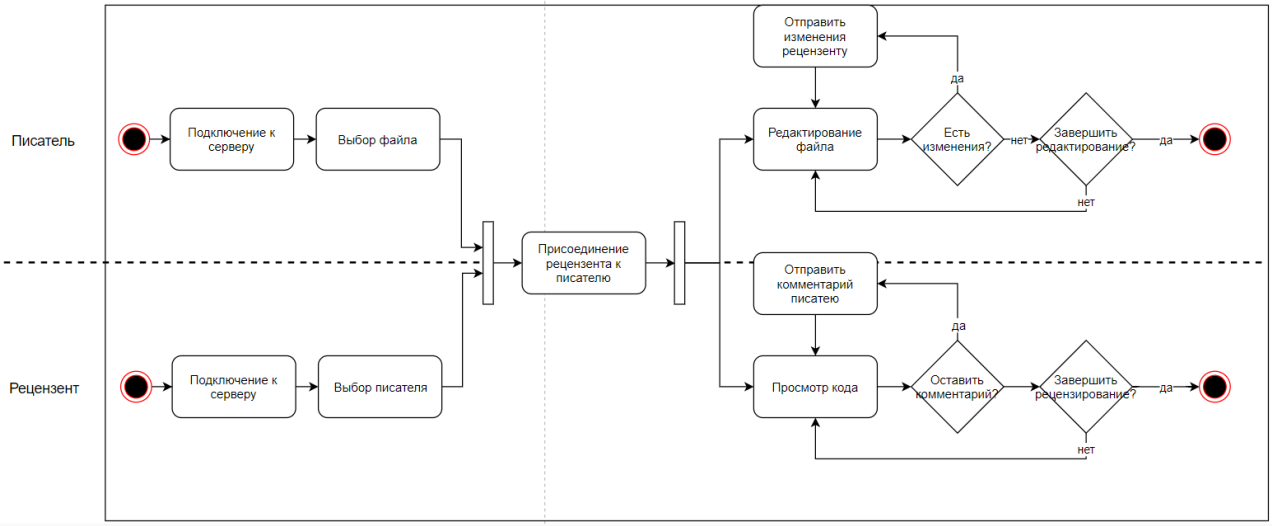


Рисунок16 – Диаграмма деятельности для клиентской части

Перед отправкой на сервер данные на стороне клиента выстраиваются в определённом порядке. Строка разбивается на блоки, состоящие из угловых скобок, между которыми вставляются необходимые серверу данные.Каждый блок имеет свой номер в строке (см. рис. 17):

<command><requestRange><nickName><role><filename><offset><count><filler><data>

0 1 2 3 4 5 6 7 8

Рисунок17 – Структура запроса

Неиспользуемые поля при составлении запроса остаются пустыми. Пример кода, выполняющего построение запроса на подключение для сервера:

```
public string BuildConnect(byte command, Reviewer data) {
        if (Program.requestRange == 0) return null;
        string request = "";
        request += "<" + command.ToString() + ">";
        request += "<" + Program.requestRange.ToString() + ">";
```

```
            request += "<" + data.GetNickName() + ">";
            request += "<" + data.GetRole() + ">";

            request += "<" + ">";
            request += "<" + ">";
            request += "<" + ">";
            request += '<';
            for (int i = request.Length; i < Program.requestRange - 1; i++)
{
                request += "0";
            }
            request += ">";

            return request;
        }
```

2.3 Разработка серверного приложения

Сервер реализован в виде консольного приложения. При запуске сервера начинается прослушивание входящих соединений по указанному порту.На сервере реализован класс Server. Данный класс прослушивает входящие соединения и создаёт объект класса TcpClientдля каждого подключенного клиента.На сервере реализован класс TheClient, содержащий данные клиентов и функции для работы с ними. Данный класс отвечает за работу над одним подключенным клиентом.Классы TheClientи ClientRequestреализуют выделение нитей для клиентов и работу с запросами от них, соответственно.Класс FileProcessingотвечает за работу с файлами.

При подключении клиента, сервер создаёт новую нить для данного клиента и запускает её с методом Processкласса TheClient:

```
public void Process()
        {
            do
            {
                clientStream = clientConnection.GetStream();
                clientRequest = new ClientRequest(this, fileProcessing);
```

```
            byte command = clientRequest.RecieveRequest(clientConnection.Connected);
            string response = clientRequest.ExecuteRequest(command);
            if (response != OperationResult.disconnect && response != OperationResult.nameIsBusy)
{
                clientRequest.SendResponse(response);
            }
            else if (response == OperationResult.nameIsBusy)
            {
                clientRequest.SendResponse(response);
                clientStream.Close();
                clientConnection.Close();
            }
            else if(response == null)
            {
                clientRequest.SendResponse("null response");
            }
            else
            {
clientRequest.SendResponse(response);
DisconnectClient(clientStream, GetNickName());
            }
        } while (clientConnection.Connected);
}
```

МетодProcessожидаетпоступлениязапросовотклиента. Дляпринятияивыполнениязапросовиспользуютсяметодыклассa ClientRequest, RecieveRequestиExecuteRequest, соответственно.

В зависимости от переданной команды сервер выбирает нужные ему данные из строки запроса. Например, для разбора запроса на подключение сервер вызываетследующий метод с параметрами, выбираемыми из запроса:

```
ExecuteWriterConnect(GetNickName(request),
        mainStream,
        GetRequestRange(request),
        GetRole(request));
```

Во время работы с файлами писателей, данные этих файлов хранятся в словаре, где ключ– это имя писателя, а значение – редактируемый файл:

public static Dictionary<string, string> writersData = new Dictionary<string, string>();

Вспомогательные классы CommandsиOperationResultсодержат команды, используемые для обмена информацией между клиентом и сервером, а также, сообщения о результатах тех или иных операций в проекте.

Диаграмма последовательности представлена на рисунке 18:

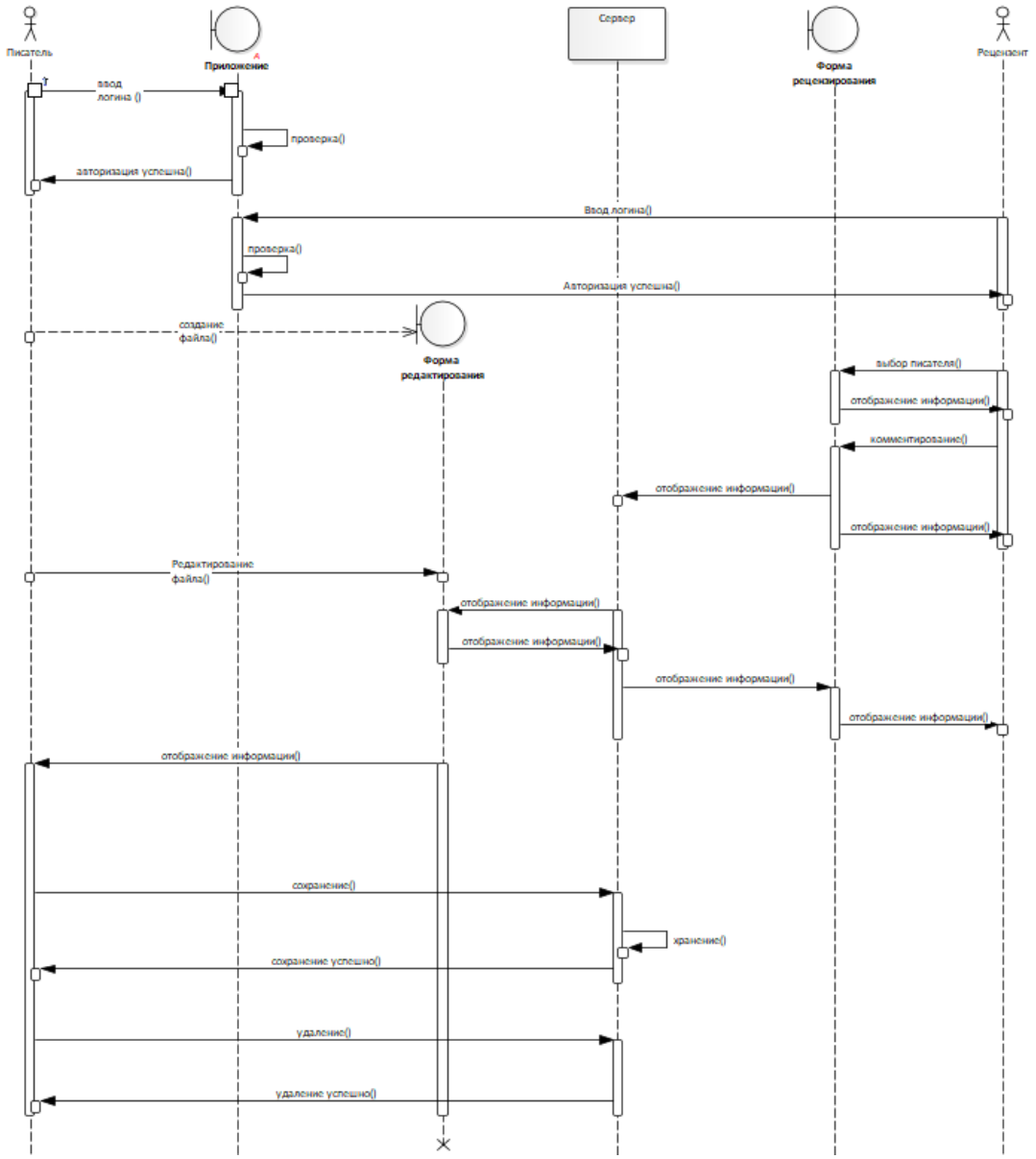


Рисунок11 – Диаграмма последовательности

3 ЭКСПЛУАТАЦИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

Данная система предназначена для развёртывания на нескольких компьютерах локальной или глобальной сети.

Один из компьютеров сети предназначен для работы с серверным приложением. Остальные компьютеры – клиентские.

Диаграмма развёртывания представлена на рисунке 19:

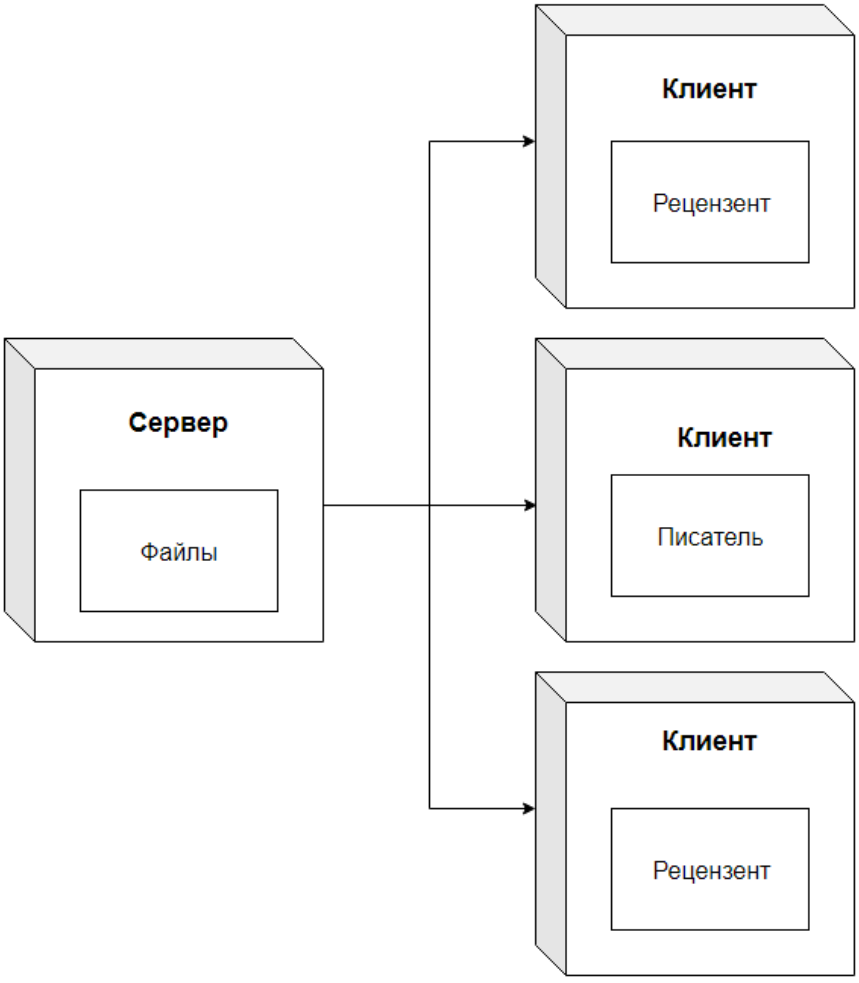


Рисунок19 – Диаграмма развёртывания приложения

Для работы с приложением сначала необходимо подключиться к серверу. После подключения в зависимости от выбранного режима представляется

возможность редактировать файлы с программным кодом, а также, проводить рецензию данных файлов.

3.1 Подключение к серверу

После запуска клиентской части отображается следующая форма (см. рис. 20):

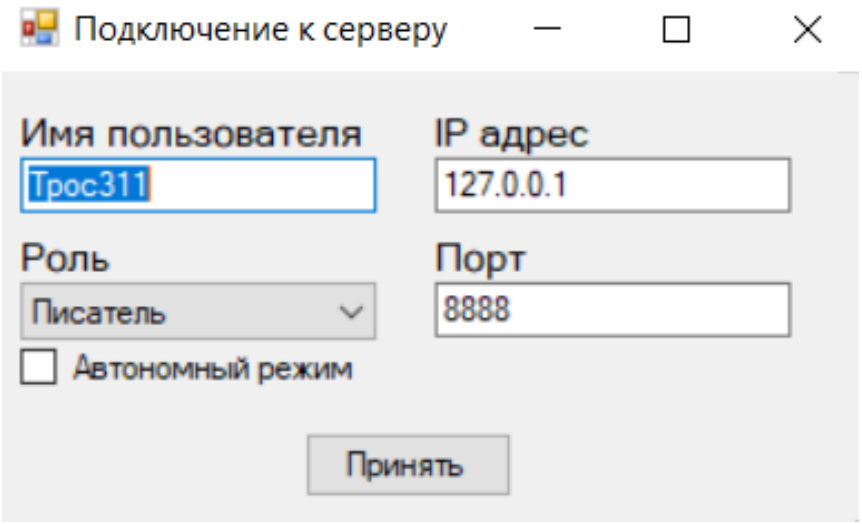


Рисунок20 – Форма подключения к серверу

Данная форма позволяет выбрать режим работы и ввести данные, необходимые для подключения к серверу.

В форму необходимо ввести имя пользователя, под которым будет проходить работа с сервером, роль, IPадрес сервера и порт. Режим, в котором происходит работа с сервером, зависит от выбранной роли.

При выборе роли «Писатель» будет активирован режим писателя. При выборе роли «Рецензент» будет активирован режим рецензента, соответственно. При активации поля «Автономный режим», поля «Роль», «IPадрес», «Порт» и «Имя пользователя» становятся неактивными, поскольку для работы в данном режиме соединение с сервером не нужно.

Таким образом, приложение имеет режимы «Писатель» и «Рецензент», которые являются основными, а также, дополнительный автономный режим.

Затем необходимо нажать кнопку «Принять», после чего запустится приложение в одном из трёх режимов, описанных выше.

3.2 Режим «Писатель»

При выборе данного режима происходит подключение к серверу. Затем появляется окно выбора файла на сервере (см. рис. 21):

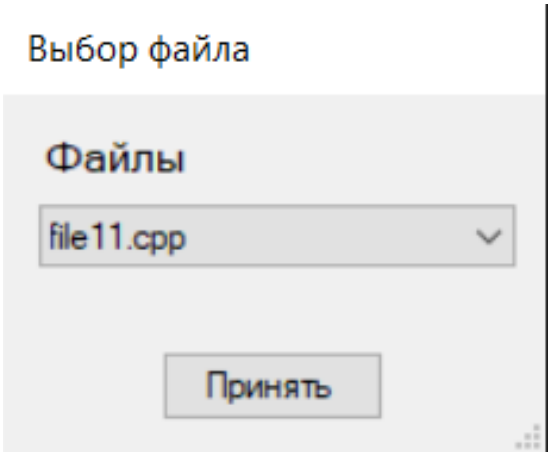


Рисунок21 – Окно выбора файла на сервере

В данном окне необходимо выбрать файл из выпадающего списка и нажать кнопку «Принять».

Далее откроется окно редактирования для выбранного файла. В нижней части находится поле «Комментарии», в котором отображаются комментарии от рецензентов, наблюдающих за работой данного писателя (см. рис. 22):

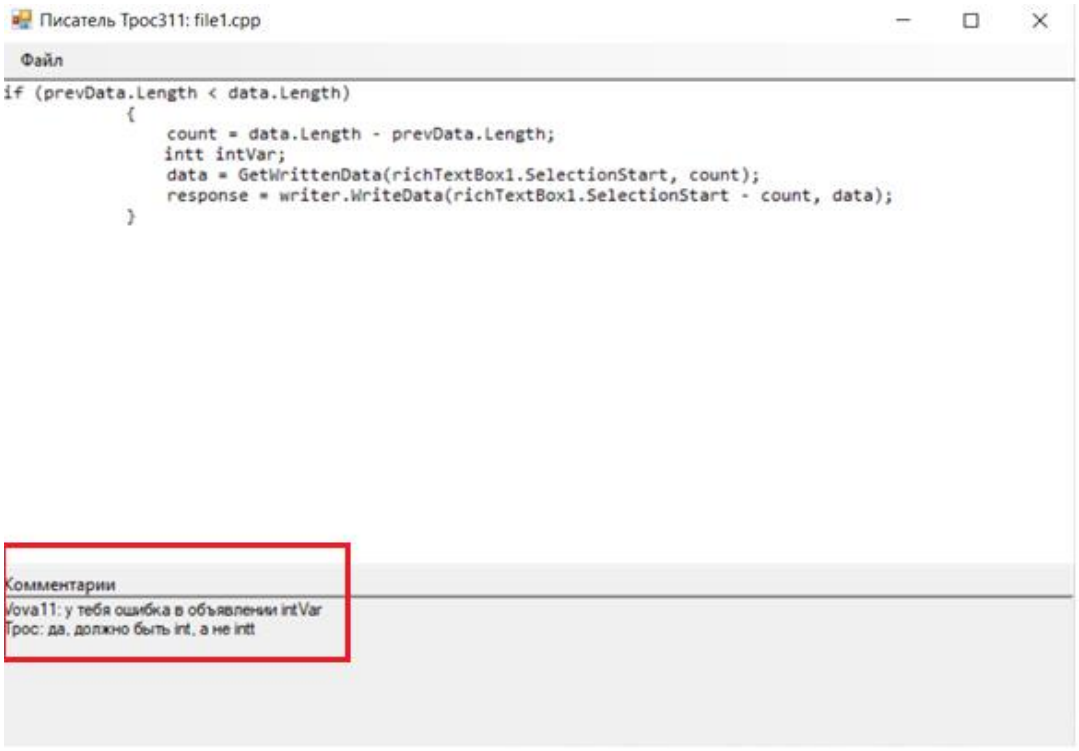


Рисунок22 – Работа режима «Писатель»

В случае, если на сервере ещё нет ни одного файла для данного пользователя, будет выведено соответствующее сообщение (см. рис. 23):

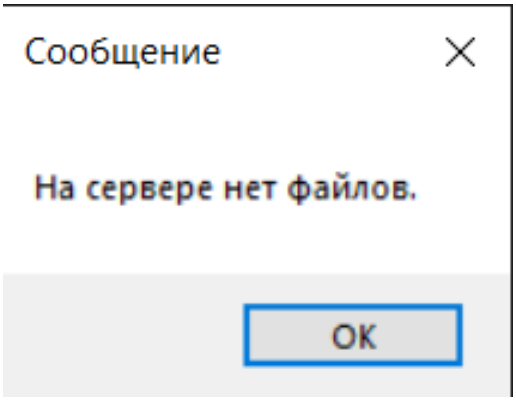


Рисунок23 – Сообщение об отсутствии файлов на сервере

После чего будет открыта пустая форма, где поле редактирования заблокировано (см. рис. 24).

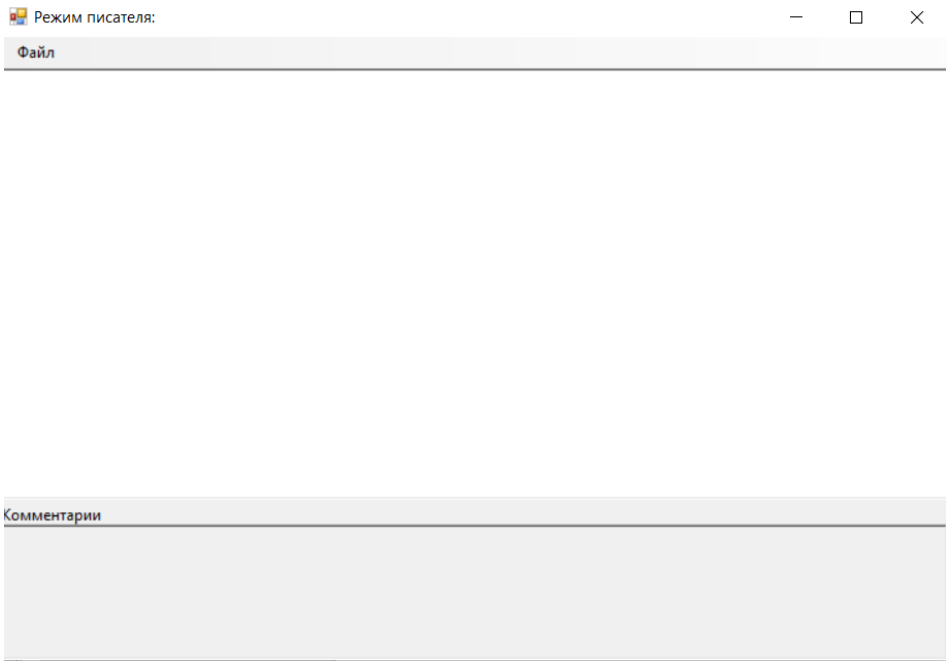


Рисунок24 – Пустая форма писателя

Поле редактирования будет разблокировано, когда будет создан новый файл.

В режиме писателя доступны следующие функции по отношению к файлам на сервере:

создать;

открыть;

сохранить;

удалить;

Данные функции доступны из выпадающего меню «Файл» в левом верхнем углу экрана (см. рис. 25):

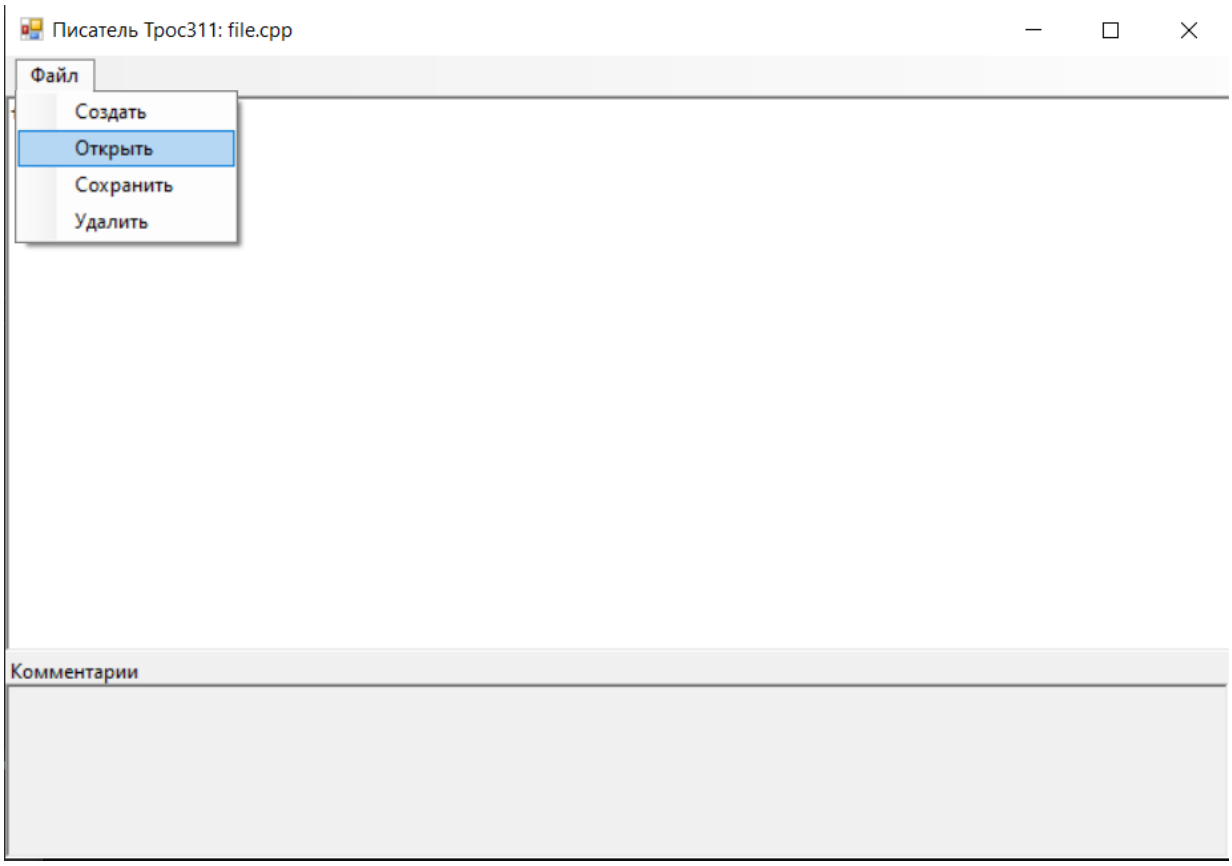


Рисунок25 – Функции для работы с файлом

Создание файла

После выбора функции «Создать» появляется окно ввода имени и расширения файла (см. рис. 26):

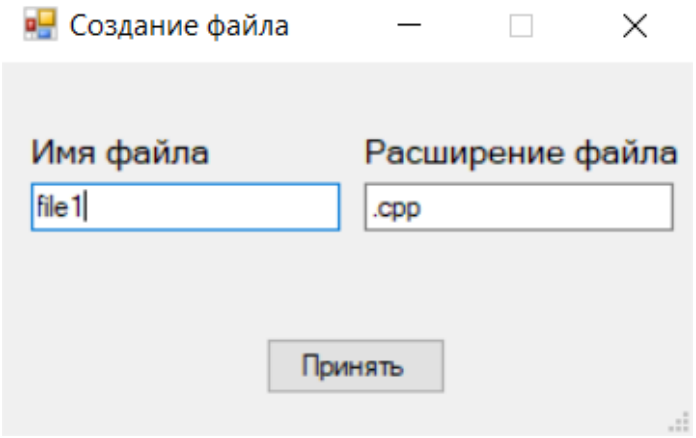


Рисунок26 – Форма создания файла

После нажатия кнопки «Принять» будет создан новый файл на сервере и открыт для редактирования на клиенте.

Открытие файла

Для открытия файла на сервере необходимо выбрать пункт «Открыть». После чего будет открыта форма выбора файла на сервере (см. рис. 21).

Сохранение файла

Для сохранения файла необходимо выбрать пункт «Сохранить» в меню «Файл», после чего будет выведено окно результата выполнения операции (см. рис. 27):

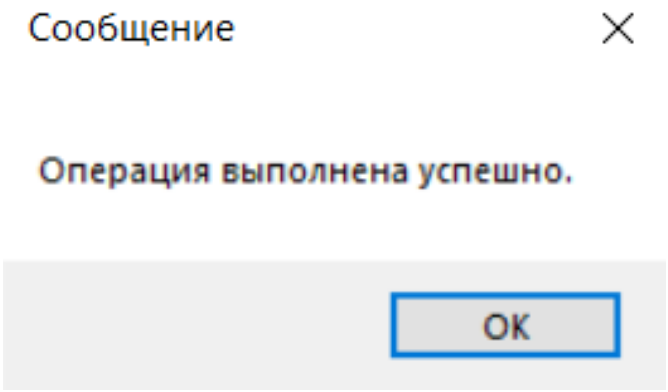


Рисунок27 – Сообщение об успешном выполнении операции

Удаление файла

Для удаления файла необходимо выбрать пункт «Удалить» в меню «Файл», затем выбрать файл из списка в открывшейся форме (см. рис. 21). Далее будет выведен результат выполнения операции, а выбранный файл удалён с сервера (см. рис. 27).

3.3 Режим «Рецензент»

При выборе режима «Рецензент» происходит подключение к серверу, а затем открывается окно выбора писателей, подключенных на данный момент к серверу (см. рис. 28):

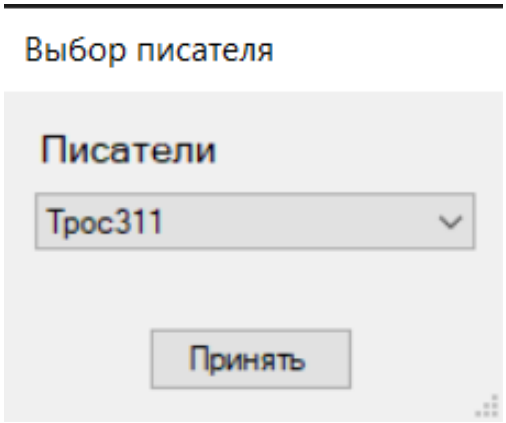


Рисунок28 – Форма выбора писателя

После нажатия кнопки «Принять» будет открыта форма рецензента (см. рис. 29):

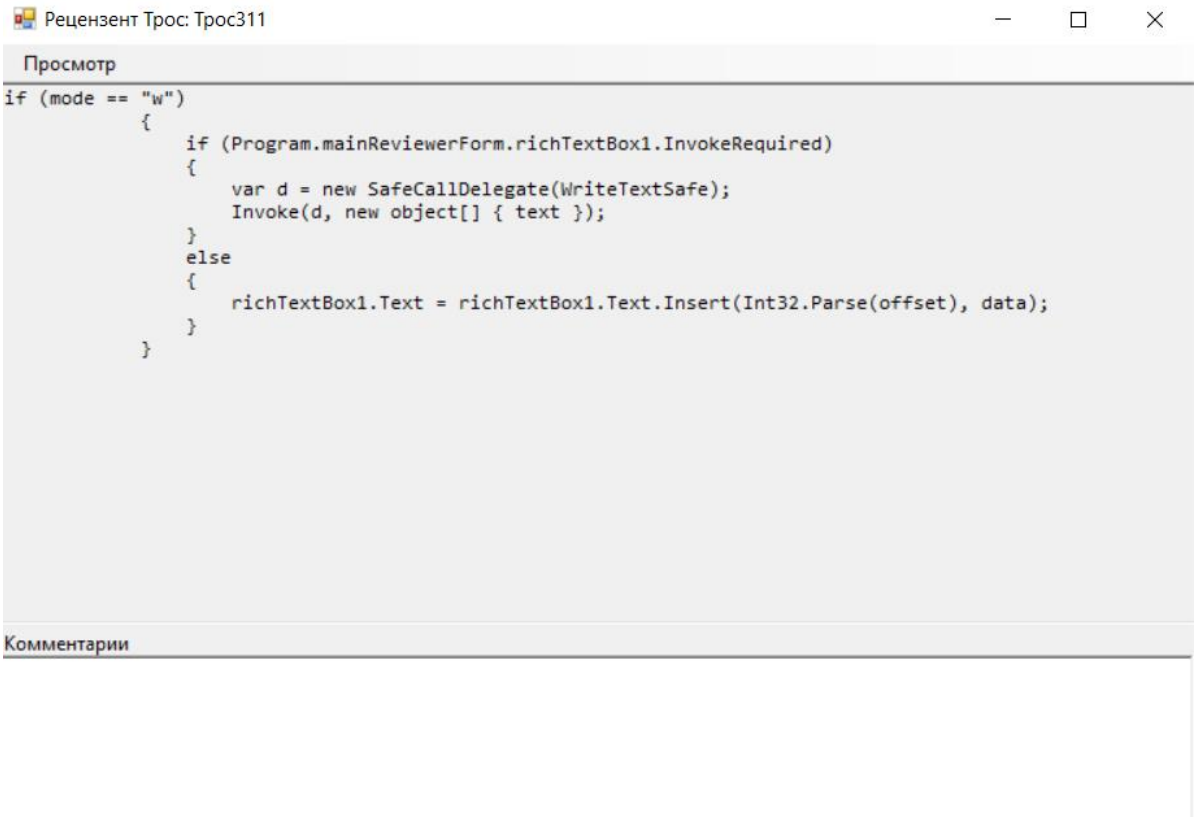


Рисунок29 – Форма рецензента

В центре формы находится поле просмотра. В данном поле видно процесс редактирования файла писателем.

Ниже, в поле «Комментарии», можно отправлять комментарии писателю. Для этого необходимо ввести комментарий в поле внизу формы и нажать клавишу «Enter» для отправки (см. рис. 30):

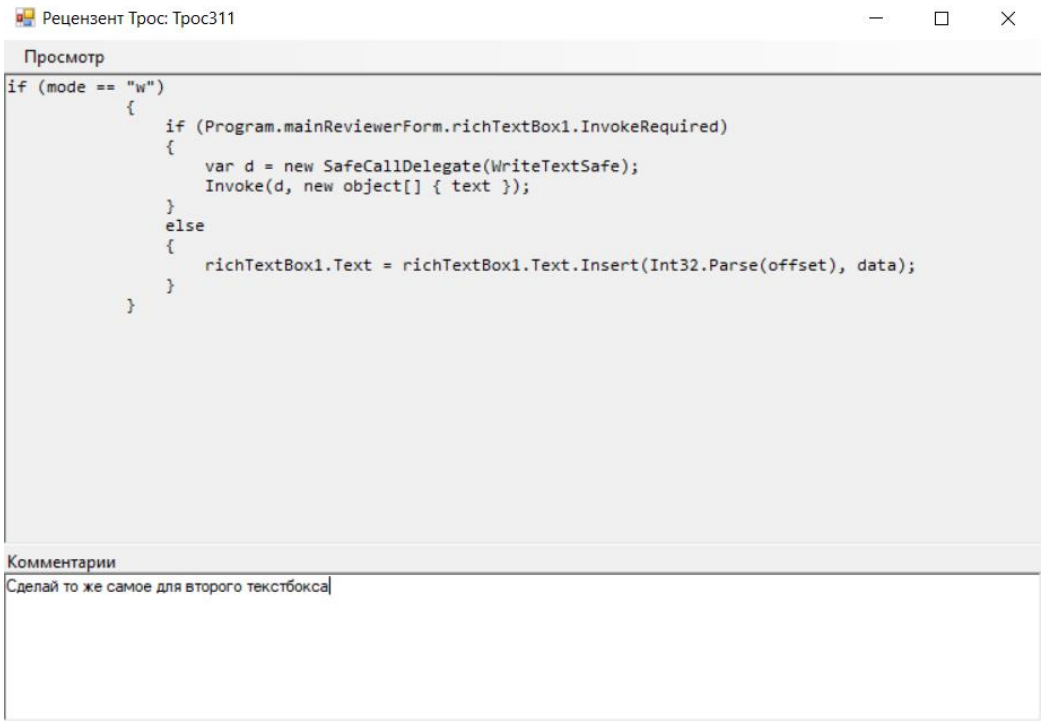


Рисунок30 – Ввод комментариев

Данный комментарий отобразится у писателя в том же поле (см. рис. 31):

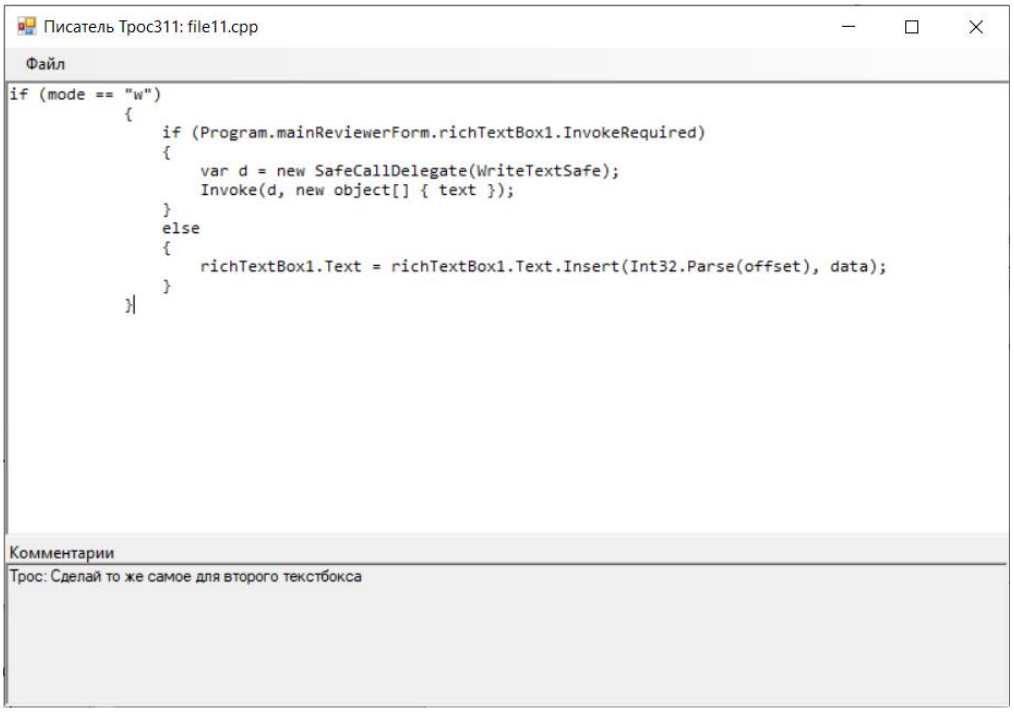


Рисунок31 – Отображение комментария у писателя

После отправки комментария писателю поле очистится для новых комментариев.

Режим «Рецензент» реализует следующие функции в меню «Просмотр» (см. рис. 32):

в текущем окне;

в новом окне.

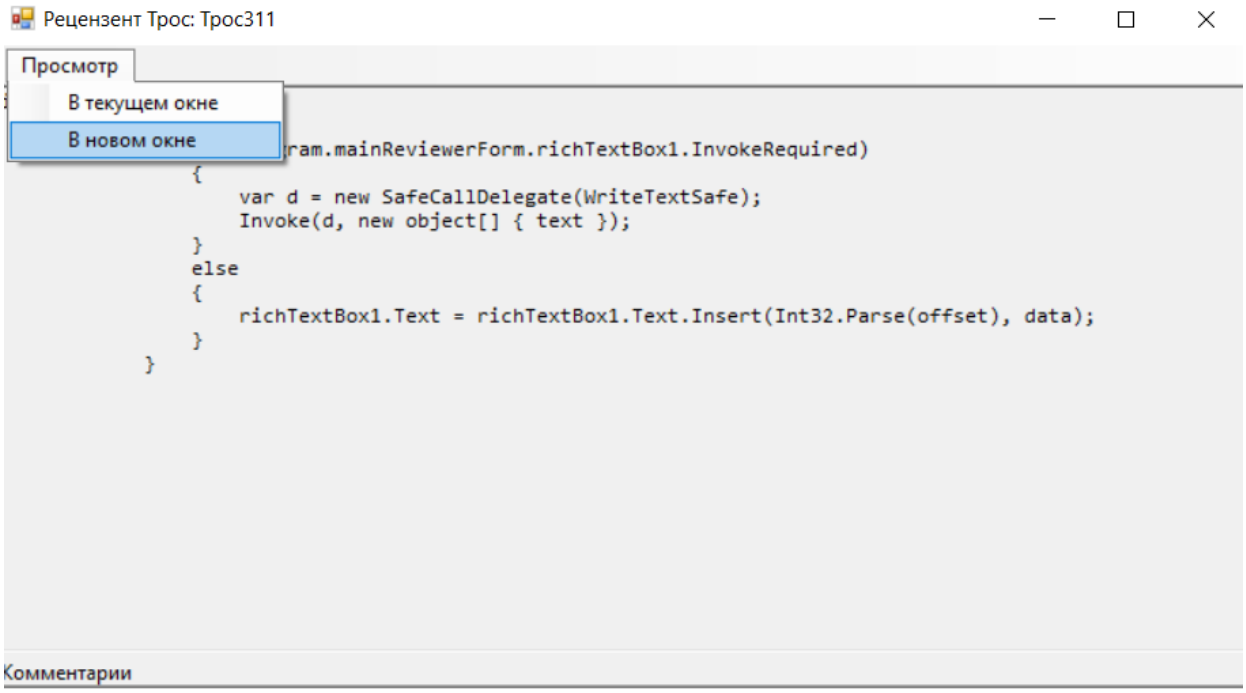


Рисунок32 – Функции режима «Рецензент»

При выборе пункта «В текущем окне» будет выведено окно выбора писателя (см. рис. 28).

После выбора другого писателя в данной форме будет начат процесс наблюдения за ним (см. рис. 33):

Рисунок33 – Просмотр другого писателя в текущем окне

При выборе пункта «В новом окне» будет открыта новая форма для наблюдения.

При этом наблюдение в предыдущей форме продолжается (см. рис. 34):

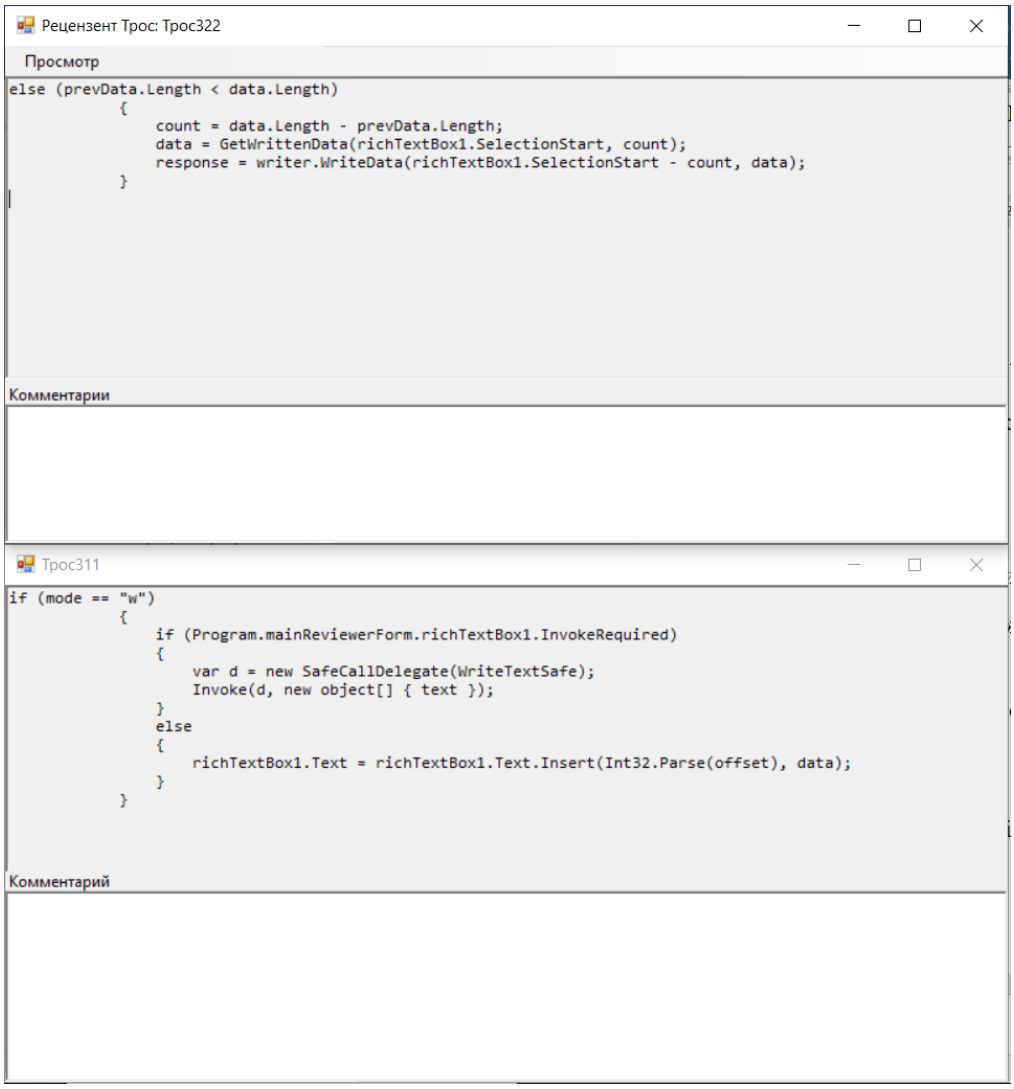


Рисунок34 – Результат выполнения операции «В новом окне»

На рисунке 35 представлен процесс ревизии кода двух писателей одним рецензентом:

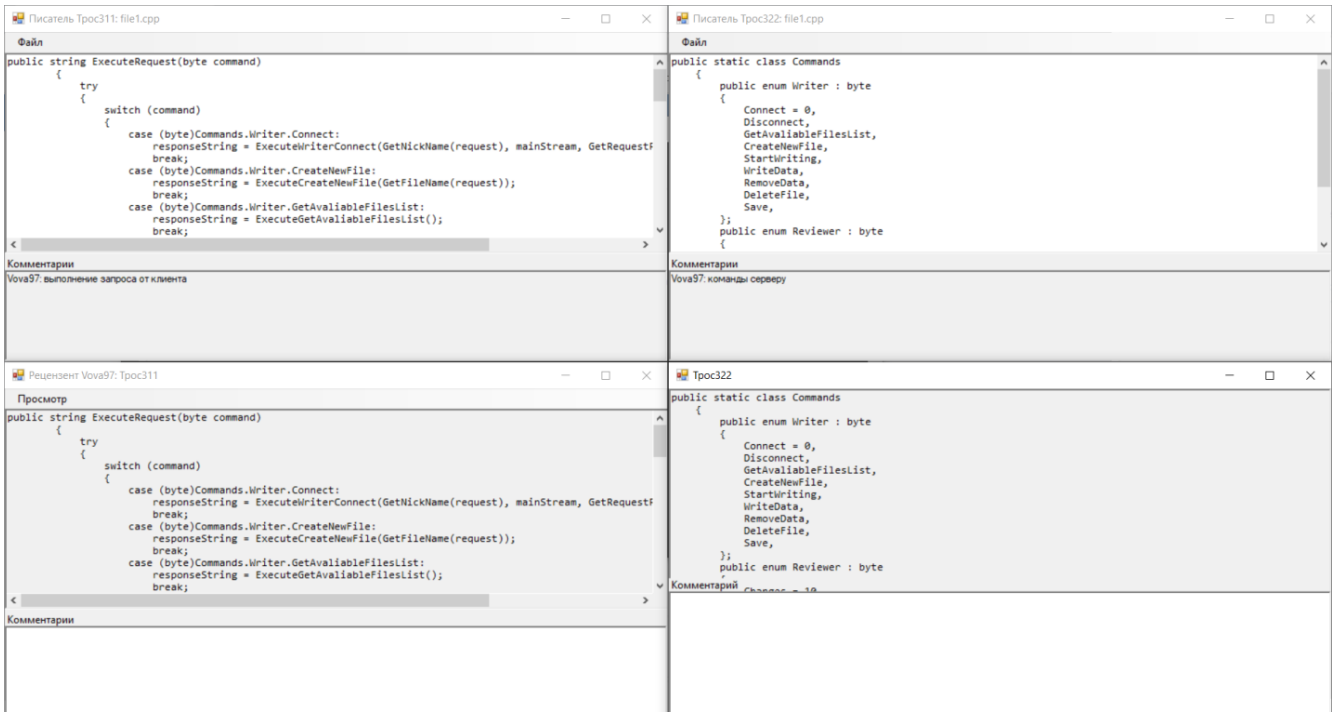


Рисунок35 – Рецензирование двух писателей одним рецензентом

3.4 Автономный режим

Автономный режим обеспечивает базовые функции текстового редактора, а именно, создания, открытия, редактирования и сохранения файла. Форма автономного режима после запуска приложения показана на рисунке36:

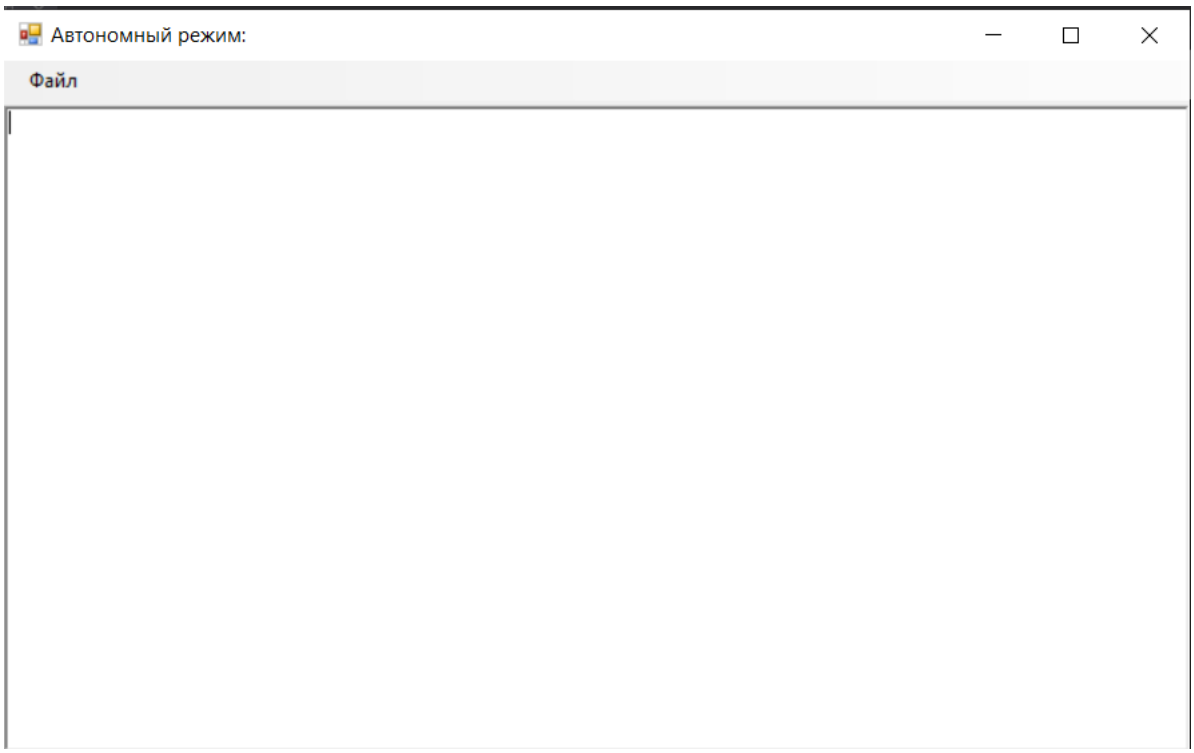


Рисунок36 – Форма автономного режима

В автономном режиме доступны следующие операции над файлом:

создать;

открыть;

сохранить как…

При выборе любого из данных вариантов открывается окно выбора файла (см. рис. 37):

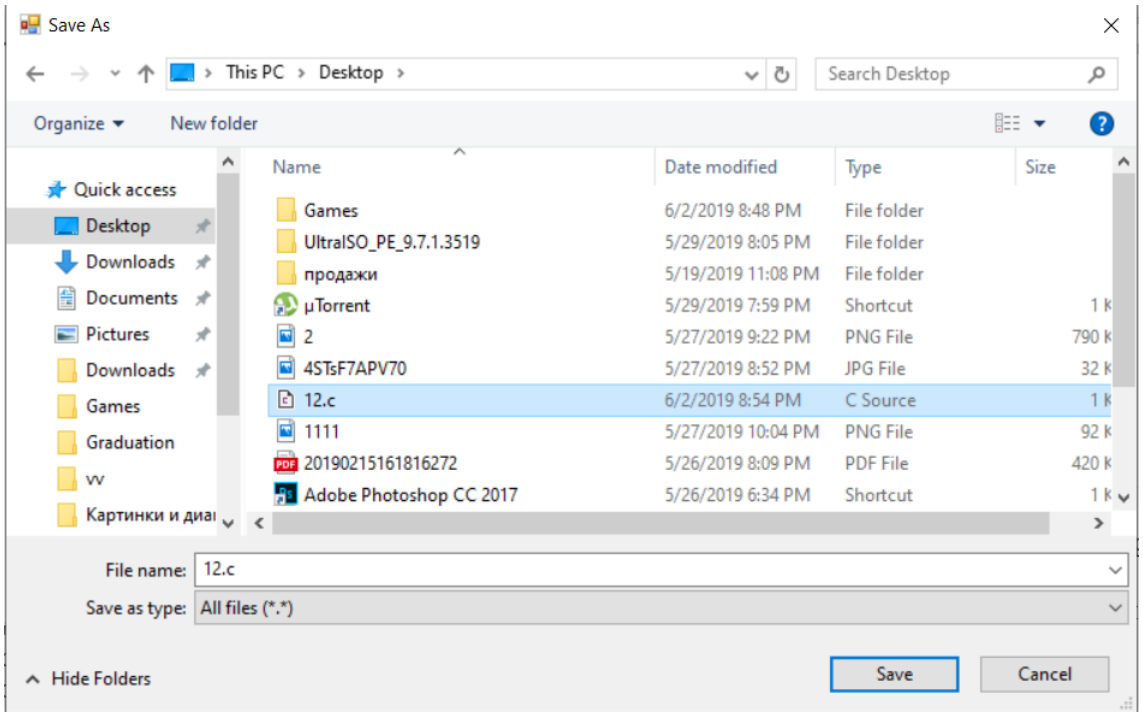


Рисунок37 – Диалоговое окно выбора файла

В зависимости от выбранного пункта в меню «Файл» произойдёт соответствующее действие для выбранного файла (или имени файла).

ЗАКЛЮЧЕНИЕ

В данной работе была реализована система дистанционной ревизии кода, позволяющая проводить рецензирование и редактирование файлов в режиме реального времени.

Поставленная цель быладостигнута. Все задачи были решены в полном объеме.

В данной работе:

1. изучены современные подходы к повышению качества программного кода;
2. проведён анализ предметной области на примере существующих систем и платформ, обеспечивающих рецензирование кода;
3. определена и рассмотрена архитектура приложения;
4. определены методы и средства, используемые при разработке компонентов системы дистанционной ревизии кода;
5. приведено описание хода разработки приложения, в частности клиента (режимы работы) и сервера;
6. на основе вышесказанного разработаны компоненты системы;
7. описан процесс эксплуатации системы;
8. рассмотрена работа основных компонентов системы, а также процессы функционирования этих компонентов.

Возможно дальнейшее развитие системы дистанционной ревизии кода, которое заключается в расширении ее функциональных возможностей, таких как:

1) улучшение механизма обработки подключенных клиентов и обмена данными между ними;
2) изменение алгоритма передачи данных по сети с целью повышения надёжности;

3) ведение истории изменений и комментариев;
4) введение в графический интерфейс нумерации строк и подсветки синтаксиса для различных языков программирования.

В процессе выполнения работы получен опыт программирования на языке C# и разработки приложений с графическим интерфейсом.Также, получены навыки по выбору архитектуры приложения и разработке приложений, имеющих сложную структуру модулей и состоящих из множества компонентов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Просмотр кода.[Электронный ресурс]. Дата обращения: 7.05.2019.URL:https://ru.wikipedia.org/wiki/Просмотр_кода/.
2. Сравнение методик обзора кода. [Электронный ресурс]. Дата обращения: 8.05.2019. URL: https://habr.com/ru/post/151450/.
3. 20 причин проводить обзоры кода. [Электронный ресурс]. Дата обращения: 8.05.2019. URL:https://habr.com/ru/post/88075/
4. С. Макконнелл. Совершенный код. Мастер-класс / Пер. с англ. – СПб.: БХВ, 2018. – 896 стр.
5. Парное программирование. [Электронный ресурс]. Дата обращения: 10.05.2019. URL: https://ru.wikipedia.org/wiki/Парное_программирование/.
6. Разработка через тестирование. [Электронный ресурс]. Дата обращения: 10.05.2019. URL: https://ru.wikipedia.org/wiki/Разработка_через_тестирование
7. Разработка через тестирование. [Электронный ресурс]. Дата обращения: 10.05.2019. URL:https://ru.wikipedia.org/wiki/Клиент_—_сервер
8. Протокол передачи данных. [Электронный ресурс]. Дата обращения: 25.05.2019. URL: https://ru.wikipedia.org/wiki/Протокол_передачи_данных.
9. Сетевая модель OSI. [Электронный ресурс]. Дата обращения: 26.05.2019. URL: https://ru.wikipedia.org/wiki/Сетевая_модель_OSI.
10. Сетевая модель OSI. [Электронный ресурс]. Дата обращения: 26.05.2019. URL: https://ru.wikipedia.org/wiki/Текстовый_процессор.
11. Шилдт Г. «C# 4.0: полное руководство» 2011г.
12. Экстремальное программирование: Pair Programming. [Электронный ресурс]. Дата обращения: 6.06.2019. URL: https://habr.com/ru/post/151563/.

13.5 способов повысить концентрацию. [Электронный ресурс]. Дата обращения: 6.06.2019. URL: https://newochem.ru/zdorove/5-sposobov-povysit-koncentraciyu/.

ПРИЛОЖЕНИЕ А: ЛИСТИНГ ПРОГРАММЫ КЛИЕНТА

Приложение А.1 – Файл «Writer.cs»

```
using System;
using System.Collections.Generic;
using System.Net.Sockets;
using TextEditorClient.Scripts;

namespace TextEditorClient.Clients
{
    class Writer
    {
        private bool isConnected = false;
        private WriterRequest request = null;
        private TcpClient tcpWriter = null;
        private string response = null;
        private string nickName = null;
        private string role = null;
        private string currentFile = null;
        private string data = null;
        private int offset = 0;
        private int count = 0;

        public Writer()
        {
        }
        public NetworkStream GetStream()
        {
            return request.GetStream();
        }
        public string Connect()
        {
            try
            {
                tcpWriter = new TcpClient(Program.ip, Program.port);
                request = new WriterRequest(tcpWriter);
                request.SendRequest((byte)Commands.Writer.Connect, this);
                response = request.AcceptResponse();
                if (response != OperationResult.success)
                {
                    return response;
                }
                isConnected = true;
                return response;
            }
            catch(Exception e)
            {
                return e.Message;
            }
        }
        public List<string> GetAvaliableFilesList()
        {
            request.SendRequest((byte)Commands.Writer.GetAvaliableFilesList, this);
            response = request.AcceptResponse();
            List<string> filesList = new List<string>();
            string[] files = response.Split('>');
            for (int i = 0; i < files.Length; i++) {
                if (files[i].Contains("buffer")) continue;
                filesList.Add(files[i].TrimStart('<'));
            }
```

```
            if(filesList[0] == OperationResult.noFiles) return null;
            return filesList;
        }
        public string StartWriting(string file)
        {
            response = null;
            request.SendRequest((byte)Commands.Writer.StartWriting, this);
            response = request.AcceptResponse();
            if (response == OperationResult.success) response = "";
            return response;
        }
        public string CreateFile(string fName) {
            currentFile = fName;
            request.SendRequest((byte)Commands.Writer.CreateNewFile, this);
            response = request.AcceptResponse();
            return response;
        }
        public string WriteData(int offS, string changes)
        {
            data = changes;
            offset = offS;
            request.SendRequest((byte)Commands.Writer.WriteData, this);
            response = request.AcceptResponse();
            return response;
        }
        public string RemoveData(int offS, int c)
        {
            offset = offS;
            count = c;
            request.SendRequest((byte)Commands.Writer.RemoveData, this);
            response = request.AcceptResponse();
            return response;
        }
        public string Save()
        {
            if (currentFile != null)
            {
                request.SendRequest((byte)Commands.Writer.Save, this);
                response = request.AcceptResponse();
                return response;
}
            else
            {
                return OperationResult.fileNameError;
            }

        }
        public string DeleteFile(string file)
{
            string savef = currentFile;
            currentFile = file;
            request.SendRequest((byte)Commands.Writer.DeleteFile, this);
currentFile = savef;
            response = request.AcceptResponse();
            return response;
}
        public int DisconnectFromServer()
        {
            if (isConnected == false) return 0;
            request.SendRequest((byte)Commands.Writer.Disconnect, this);
            response = request.AcceptResponse();
            request.CloseStream();
tcpWriter.Close();
            if (response == "ok") return 0;
```

```
            else return 1;
        }
        public string GetFileName()
        {
            return currentFile;
        }
        public long GetOffset() {
            return offset;
        }
        public long GetCount()
        {
            return count;
        }
        public int SetCurrentFile(string fName)
        {
            currentFile = fName;
            return 0;
        }
        public bool IsConnected() {
            return isConnected;
        }
        public void SetClientInfo(string nName, string r)
        {
            nickName = nName;
            role = r;
        }
        public string GetRole()
        {
            return role;
        }
        public string GetNickName()
        {
            return nickName;
        }
        public string GetData()
        {
            return data;
        }
        public string GetCurrentFile()
        {
            return currentFile;
        }
    }
}
```

Приложение А.2 – Файл «BuildWriterRequest.cs»

```
using TextEditorClient.Clients;
using System.Net.Sockets;
//<command><requestRange><nickName><role><filename><offset><count><filler><data>
namespace TextEditorClient.Scripts.Requests
{
    class BuildRequest
    {
        public NetworkStream stream = null;
        public BuildRequest(TcpClient c) {
            stream = c.GetStream();
        }
        public BuildRequest()
        {
        }
        public NetworkStream GetStream()
        {
```

```
    return stream;
}
public string BuildConnect(byte command, Writer data) {
    if (Program.requestRange == 0) return null;

    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + Program.requestRange.ToString() + ">";
    request += "<" + data.GetNickName() + ">";
    request += "<" + data.GetRole() + ">";
    request += "<" + ">";
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
    }
    request += ">";
    return request;
}
public string BuildCreateNewFile(byte command, Writer data)
{
    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + ">";   //requestRange
    request += "<" + ">";   //NickName
    request += "<" + ">";   //Role
    request += "<" + data.GetCurrentFile() + ">";   //fileName
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
    }
    request += ">";
    request += "<" + ">";
    return request;
}
public string BuildGetAvaliableFilesList(byte command, Writer data)
{
    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + ">";   //requestRange
    request += "<" + ">";   //NickName
    request += "<" + ">";   //Role
    request += "<" + ">";   //FileName
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
    }
    request += ">";

    return request;
}
public string BuildStartWriting(byte command, Writer data) {
    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + ">";   //requestRange
    request += "<" + data.GetNickName() + ">";   //NickName
    request += "<" + ">";   //Role
    request += "<" + data.GetFileName() + ">";
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
```

```
    }
    request += ">";

    return request;
}
public string BuildWrite(byte command, Writer data)
{
    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + ">";   //requestRange
    request += "<" + data.GetNickName() + ">";   //NickName
    request += "<" + ">";   //Role
    request += "<" + data.GetFileName() + ">";
    request += "<" + data.GetOffset() + ">";
    request += "<" + ">";
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
    }
    request += ">";
    request += "<" + data.GetData() + ">";
    return request;
}
public string BuildRemoveData(byte command, Writer data)
{
    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + ">";   //requestRange
    request += "<" + data.GetNickName() + ">";   //NickName
    request += "<" + ">";   //Role
    request += "<" + data.GetFileName() + ">";
    request += "<" + data.GetOffset() + ">";
    request += "<" + data.GetCount() + ">";
    request += "<" + ">";
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
    }
    request += ">";
    request += "<" + ">";
    return request;
}
public string BuildSave(byte command, Writer data)
{
    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + ">";   //requestRange
    request += "<" + data.GetNickName() + ">";   //NickName
    request += "<" + ">";   //Role
    request += "<" + data.GetFileName() + ">";
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
    }
    request += ">";
    request += "<" + ">";
    return request;
}
public string BuildDeleteFile(byte command, Writer data)
{
    string request = "";
```

```
            request += "<" + command.ToString() + ">";
            request += "<" + ">";   //requestRange
            request += "<" + data.GetNickName() + ">";   //NickName
            request += "<" + ">";   //Role
            request += "<" + data.GetFileName() + ">";
            request += '<';
            for (int i = request.Length; i < Program.requestRange - 1; i++)
            {
                request += "0";
            }
            request += ">";
            request += "<" + ">";
            return request;
        }
        public string BuildDisconnect(byte command, Writer data)
        {
            string request = "";
            request += "<" + command.ToString() + ">";
            request += "<" + ">";   //requestRange
            request += "<" + data.GetNickName() + ">";
            request += "<" + ">";   //Role
            request += "<" + ">";   //fileName
            request += '<';
            for (int i = request.Length; i < Program.requestRange - 1; i++)
            {
                request += "0";
            }
            request += ">";
            request += "<" + ">";
            return request;
}
    }
}
```

Приложение А.3 – Файл «WriterRequest.cs»

```
using System.Text;
using System.Net.Sockets;
using TextEditorClient.Scripts;
using TextEditorClient.Scripts.Requests;
using TextEditorClient.Clients;
using System;
using System.IO;

namespace TextEditorClient
{
    class WriterRequest : BuildRequest
    {
        private string request = null;
        private BinaryReader reader = null;
        private BinaryWriter writer = null;
        public WriterRequest(TcpClient c) : base(c) {
            stream = c.GetStream();
            writer = new BinaryWriter(stream, Encoding.Unicode, true);
            reader = new BinaryReader(stream, Encoding.Unicode, true);
            reader.BaseStream.ReadTimeout = 100;
        }
        public WriterRequest() : base()
        {
        }
        public void CatchChanges(object st)
        {
            while (!Program.acceptBlock)
```

```
            {
                try
                {
                    byte[] bytes = new byte[64];
                    int count = 0;
                    StringBuilder builder = new StringBuilder();
                    while (((NetworkStream)st).DataAvailable)
                    {
                        count = ((NetworkStream)st).Read(bytes, 0, bytes.Length);
                        builder.Append(Encoding.Unicode.GetString(bytes, 0, count));
                    }
                    string[] data = builder.ToString().Split();
                    if (builder.Length > 0)
                    {
                        string[] result = builder.ToString().Split('>');
                        string name = result[0].TrimStart('<');
                        Program.mainWriterForm.WriteTextSafe(builder.ToString());
                    }
                }
                catch(Exception e)
                {

                }
            }
        }
        public int SendRequest(byte command, Writer data)
        {
            switch (command) {
                case (byte)Commands.Writer.Connect:
                    request = BuildConnect(command, data);
                    Send(request);
                    break;
                case (byte)Commands.Writer.Disconnect:
                    request = BuildDisconnect(command, data);
                    Send(request);
                    break;
                case (byte)Commands.Writer.GetAvaliableFilesList:
                    request = BuildGetAvaliableFilesList(command, data);
                    Send(request);
                    break;
                case (byte)Commands.Writer.StartWriting:
                    request = BuildStartWriting(command, data);
                    Send(request);
                    break;
                case (byte)Commands.Writer.WriteData:
                    request = BuildWrite(command, data);
                    Send(request);
                    break;
                case (byte)Commands.Writer.RemoveData:
request = BuildRemoveData(command, data);
                    Send(request);
                    break;
                case (byte)Commands.Writer.Save:
                    request = BuildSave(command, data);
                    Send(request);
                    break;
                case (byte)Commands.Writer.DeleteFile:
                    request = BuildDeleteFile(command, data);
                    Send(request);
                    break;
                case (byte)Commands.Writer.CreateNewFile:
                    request = BuildCreateNewFile(command, data);
                    Send(request);
                    break;
```

```
                default:
                    request = OperationResult.badCase;
                    break;
            }
            return 0;
        }
        private int Send(string data) {
            byte[] bytes = Encoding.Unicode.GetBytes(data);
            stream.Write(bytes, 0, bytes.Length);
            return 0;
        }
        public string AcceptResponse()
        {
            try
            {
                string response = null;
                response = reader.ReadString();
                return response;
            }
            catch (Exception e) {
                return OperationResult.success;
            }
        }
        public int CloseStream() {
            stream.Close();
            return 0;
        }
    }
}
```

Приложение А.4 – Файл «Program.cs»

```
using System;
using System.Windows.Forms;
using TextEditorClient.Clients;
using System.Text;
using System.Collections.Generic;
using TextEditorClient.ReviewerScripts;

namespace TextEditorClient
{
    static class Program
    {
        public static string ip = "127.0.0.1";
        public static readonly string wrRole = "Писатель";
        public static readonly string revRole = "Рецензент";
        public static readonly int requestRange = 100;
        public static readonly int fileNameLength = 30;
        public static int port = 8888;
        public static bool acceptBlock = false;
        public static ReviewerForm mainReviewerForm = null;
        public static WriterForm mainWriterForm = null;
        public static Dictionary<string, WatchForm> watchForms = new Dictionary<string,
WatchForm>();
        [STAThread]
        static void Main()
       {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            InputUserInfo inputUserInfo = new InputUserInfo();
            if (inputUserInfo.ShowDialog() != DialogResult.OK) return;
            if (inputUserInfo.offlineMode)
            {
```

```
                Application.Run(new OfflineForm());
            }
            else if (inputUserInfo.role == "Писатель")
            {
                Writer reviewer = new Writer();
                mainWriterForm = new WriterForm(inputUserInfo.nickName, inputUserInfo.role);
                ip = inputUserInfo.ipAddress;
                port = inputUserInfo.port;
                Application.Run(mainWriterForm);
            }
            else if (inputUserInfo.role == "Рецензент")
            {
                Reviewer reviewer = new Reviewer(inputUserInfo.nickName, inputUserInfo.role);
                mainReviewerForm = new ReviewerForm(reviewer, inputUserInfo.nickName,
inputUserInfo.role);
                ip = inputUserInfo.ipAddress;
                port = inputUserInfo.port;
                Application.Run(mainReviewerForm);
            }
        }
    }
}
```

Приложение А.5 – Файл «Reviewer.cs»

```
using System.Collections.Generic;
using System.Net.Sockets;
using TextEditorClient.Scripts;

namespace TextEditorClient.ReviewerScripts
{
    class Reviewer
    {
        private bool isConnected = false;
        private TcpClient tcpReviewer = null;
        private ReviewerRequest request = null;
        private string response = null;
        private string nickName = null;
        private string writerNickName = null;
        private string role = null;
        private string data = null;
        //private List<string> viewingWriters = new List<string>();

        public Reviewer(string nName, string r)
        {
            nickName = nName;
            role = r;
        }

        public string Connect()
        {
            try
            {
                response = null;

                tcpReviewer = new TcpClient(Program.ip, Program.port);
                //tcpReviewer.ReceiveTimeout = 20;
                request = new ReviewerRequest(tcpReviewer);
                request.SendRequest((byte)Commands.Reviewer.Connect, this);
                response = request.AcceptResponse();
                if (response == OperationResult.success)
                {
                    isConnected = true;
```

```
                return response;
            }
            return response;
        }
        catch
        {
            return OperationResult.couldNotConnect;
        }
    }
    public string StartWatching(string wNN)
    {
        writerNickName = wNN;
        request.SendRequest((byte)Commands.Reviewer.StartWatching, this);
        response = request.AcceptResponse();
        return response;
    }
    public string EndWatching(string wNN)
    {
        request.SendRequest((byte)Commands.Reviewer.EndWatching, this);
        response = request.AcceptResponse();

        //viewingWriters.Remove(wNN);
        return response;
    }
    public List<string> GetConnectedWritersList()
    {
        request.SendRequest((byte)Commands.Reviewer.GetConnectedWritersList, this);
        response = request.AcceptResponse();
        List<string> writersList = new List<string>();
        string[] files = response.Split('>');
        for (int i = 0; i < files.Length; i++)
        {
            if (files[i].Contains("buffer")) continue;
            writersList.Add(files[i].TrimStart('<'));
        }
        if (writersList[0] == OperationResult.noFiles) return null;
        return writersList;
    }
    public string WriteComment(string rnn, string wnn, string d)
    {
        data = d;
        nickName = rnn;
        writerNickName = wnn;
        request.SendRequest((byte)Commands.Reviewer.WriteComment, this);
response = request.AcceptResponse();
        return response;
    }
    //public string RemoveComment(int offS, int c)
    //{
    //    request.SendRequest((byte)Commands.Reviewer.RemoveComment, this);
    //    response = request.AcceptResponse();
    //    return response;
    //}
    public NetworkStream GetStream() {
        return request.GetStream();
    }
    public string DisconnectFromServer() {
        if (isConnected != false)
        {
            request.SendRequest((byte)Commands.Reviewer.Disconnect, this);
            response = request.AcceptResponse();
            request.CloseStream();
            tcpReviewer.Close();
            return response;
```

```
            }
            else
            {
                return OperationResult.clientIsAlreadyDisconnected;
            }
        }
        public bool IsConnected()
        {
            return isConnected;
        }
        public string GetRole()
        {
            return role;
        }
        public string GetNickName()
        {
            return nickName;
        }
        public string GetWriterNickName() {
            return writerNickName;
        }
        public string GetData()
        {
            return data;
        }
    }
}
```

Приложение А.6 – Файл «BuildReviewerRequest.cs»

```
using System.Net.Sockets;
//<command><requestRange><nickName><role><WriterName><><><filler><data>

namespace TextEditorClient.ReviewerScripts
{
    class BuildRequest
    {
        public NetworkStream stream = null;

        public BuildRequest(TcpClient c) {
            stream = c.GetStream();
        }
        public BuildRequest()
        {
        }

        public string BuildConnect(byte command, Reviewer data) {
            if (Program.requestRange == 0) return null;
            string request = "";
            request += "<" + command.ToString() + ">";
            request += "<" + Program.requestRange.ToString() + ">";
            request += "<" + data.GetNickName() + ">";
            request += "<" + data.GetRole() + ">";

            request += "<" + ">";
            request += "<" + ">";
            request += "<" + ">";
            request += '<';
            for (int i = request.Length; i < Program.requestRange - 1; i++)
            {
                request += "0";
            }
            request += ">";
```

```
    return request;
}
public string BuildGetConnectedWritersList(byte command, Reviewer data)
{
    if (Program.requestRange == 0) return null;
    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + ">";//<requestRange>
    request += "<" + data.GetNickName() + ">";
    request += "<" + ">";//<role>

    request += "<" + ">";//<WriterName>
    request += "<" + ">";
    request += "<" + ">";
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
    }
    request += ">";

    return request;
}
public string BuildStartWatching(byte command, Reviewer data)
{
    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + ">";//<requestRange>
    request += "<" + data.GetNickName() + ">";
    request += "<" + ">";//<role>

    request += "<" + data.GetWriterNickName() + ">";
    request += "<" + ">";
    request += "<" + ">";
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
    }
    request += ">";

    return request;
}
public string BuildWriteComment(byte command, Reviewer data)
{
    string request = "";
    request += "<" + command.ToString() + ">";
    request += "<" + ">";//<requestRange>
    request += "<" + data.GetNickName() + ">";   //NickName
    request += "<" + ">";//<role>

    request += "<" + data.GetWriterNickName() + ">";
    request += "<" + ">";
    request += "<" + ">";
    request += '<';
    for (int i = request.Length; i < Program.requestRange - 1; i++)
    {
        request += "0";
    }
    request += ">";
    request += "<" + data.GetData() + ">";
    return request;
}
```

```
        public string BuildDisconnect(byte command, Reviewer data) {
            string request = "";
            request += "<" + command.ToString() + ">";
            request += "<" + ">";//<requestRange>
            request += "<" + data.GetNickName() + ">";
            request += "<" + ">";//<role>

            request += "<" + data.GetWriterNickName() + ">";
            request += "<" + ">";
            request += "<" + ">";
            request += '<';
            for (int i = request.Length; i < Program.requestRange - 1; i++)
            {
                request += "0";
            }
            request += ">";

            return request;
        }
    }
}
```

Приложение А.7 – Файл «ReviewerRequest.cs»

```
using System.Text;
using System.Net.Sockets;
using TextEditorClient.Scripts;
using TextEditorClient.Clients;
using System.Windows.Forms;
using System.Threading;
using System;
using System.IO;
using System.Threading.Tasks;

namespace TextEditorClient.ReviewerScripts
{
    class ReviewerRequest : BuildRequest
    {
        private string request = null;
        BinaryReader reader = null;
        BinaryWriter writer = null;
        public ReviewerRequest(TcpClient c) : base(c) {
            stream = c.GetStream();
            writer = new BinaryWriter(stream, Encoding.Unicode, true);
            reader = new BinaryReader(stream, Encoding.Unicode, true);
        }
        public ReviewerRequest() : base()
        {
        }
        public NetworkStream GetStream()
        {
            return stream;
        }
        public int SendRequest(byte command, Reviewer data)
        {
            switch (command) {
                case (byte)Commands.Reviewer.Connect:
                    request = BuildConnect(command, data);
                    Send(request);
                    break;
                case (byte)Commands.Reviewer.GetConnectedWritersList:
                    request = BuildGetConnectedWritersList(command, data);
                    Send(request);
```

```
            break;
        case (byte)Commands.Reviewer.StartWatching:
            request = BuildStartWatching(command, data);
            Send(request);
            break;
        case (byte)Commands.Reviewer.WriteComment:
            request = BuildWriteComment(command, data);
            Send(request);
            break;
        //case (byte)Commands.Reviewer.RemoveComment:
        //    request = BuildRemoveComment(command, data);
        //    Send(request);
        //    break;
        //case (byte)Commands.Reviewer.EndWatching:
        //    request = BuildEndWatching(command, data);
        //    Send(request);
        //    break;
        case (byte)Commands.Reviewer.Disconnect:
            request = BuildDisconnect(command, data);
            Send(request);
            break;
        default:
            request = OperationResult.badCase;
            break;
    }
    return 0;
}

private string Send(string data) {
    try
    {
        byte[] bytes = Encoding.Unicode.GetBytes(data);
        stream.Write(bytes, 0, bytes.Length);
        return OperationResult.success;
    }catch(Exception e)
    {
        stream.Flush();
        return e.Message;
    }
}
public string AcceptResponse()
{
    try
    {
        //byte[] bytes = new byte[64]; // буфер для получаемых данных
        //StringBuilder builder = new StringBuilder();
        //int count = 0;
        //do
        //{
        //    count = stream.Read(bytes, 0, bytes.Length);
        //    builder.Append(Encoding.Unicode.GetString(bytes, 0, count));
        //} while (stream.DataAvailable);
        string response = null;

        //int count = reader.ReadInt32();
        response = reader.ReadString();
        //string response = builder.ToString();
        return response;
    }
    catch (Exception e) {
        return e.Message;
    }
}
//public string BuildWrite(byte command, Reviewer data)
```

```
//{
//    string request = "";
//    request += "<" + command.ToString() + ">";
//    request += "<" + ">";   //requestRange
//    request += "<" + data.GetNickName() + ">";   //NickName
//    request += "<" + ">";   //Role
//    request += "<" + ">";
//    request += "<" + data.GetOffset() + ">";
//    request += "<" + ">";
//    request += '<';
//    for (int i = request.Length; i < Program.requestRange - 1; i++)
//    {
//        request += "0";
//    }
//    request += ">";
//    request += "<" + data.GetData() + ">";
//    return request;
//}
//public string BuildRemoveData(byte command, Reviewer data)
//{
//    string request = "";
//    request += "<" + command.ToString() + ">";
//    request += "<" + ">";   //requestRange
//    request += "<" + data.GetNickName() + ">";   //NickName
//    request += "<" + ">";   //Role
//    request += "<" + ">";
//    request += "<" + data.GetOffset() + ">";
//    request += "<" + data.GetCount() + ">";
//    request += "<" + ">";
//    request += '<';
//    for (int i = request.Length; i < Program.requestRange - 1; i++)
//    {
//        request += "0";
//    }
//    request += ">";
//    request += "<" + ">";
//    return request;
//}
public void CatchChanges()
{
    while (!Program.acceptBlock)
    {
        try
        {
            byte[] bytes = new byte[64]; // буфер для получаемых данных
            //StringBuilder builder = new StringBuilder();
            int count = 0;
            StringBuilder builder = new StringBuilder();
            while (stream.DataAvailable)
            {
                count = stream.Read(bytes, 0, bytes.Length);
                builder.Append(Encoding.Unicode.GetString(bytes, 0, count));
            }
            string[] data = builder.ToString().Split();
            if (builder.Length > 0)
            {
                string[] result = builder.ToString().Split('>');
                string name = result[0].TrimStart('<');
                if (Program.watchForms.ContainsKey(name))
                {
                    Program.watchForms[name].WriteTextSafe(builder.ToString());
                }
                else
                {
```

```
                            Program.mainReviewerForm.WriteTextSafe(builder.ToString());
                        }
                    }
                }
                catch(Exception e)
                {
                    Console.WriteLine(e.Message);
                }
            }
        }
        public int CloseStream() {
            stream.Close();
            return 0;
        }
    }
}
```

Приложение А.8 – Файл «OperationResult.cs»

```
namespace TextEditorClient.Scripts
{
publicstaticclassOperationResult
    {
publicstaticstring success = "Операция выполнена успешно.";
publicstaticstring commonError = "Неизвестная ошибка.";
publicstaticstring disconnect = "Соединение закрыто успешно.";
publicstaticstring noFiles = "На сервере нет файлов.";
publicstaticstring badCase = "Необработанный case.";
publicstaticstringnoConnectedWriters = "Нет подключенных к серверу писателей.";
publicstaticstring exist = "Файл уже существует.";
publicstaticstringnameIsBusy = "Данное имя пользователя уже занято. Выберете другое.";
publicstaticstringcouldNotConnect = "Невозможно подключится к серверу.";
publicstaticstringcreateFileError = "Ошибка создания файла.";
publicstaticstring StartWritingError = "Ошибка start writing";
publicstaticstring saveError = "Ошибка сохранения файла";
publicstaticstring fileNameError = "Ошибка имени файла";
publicstaticstring choiseFileError = "Файл не выбран.";
publicstaticstring clientIsAlreadyDisconnected = "Клиент уже был отключен.";
publicstaticstringalreadyWatching = "Этот писатель уже просматривается.";
}
}
```

Приложение А.9 – Файл «Commands.cs»

```
namespace TextEditorClient.Scripts
{
    //<command><requestRange><nickName><role><filename><offset><count><writerName><filler><data>
    public static class Commands
    {
        public enum Writer : byte
        {
            Connect = 0,
            Disconnect,
            GetAvaliableFilesList,
            CreateNewFile,
            StartWriting,
            WriteData,
            RemoveData,
            DeleteFile,
            Save,
        };
        public enum Reviewer : byte
```

```
        {
            Changes = 10,
            Connect,
            WriteComment,
            RemoveComment,
            GetConnectedWritersList,
            StartWatching,  //<reviewerName><writerName>
            EndWatching,
            Disconnect,
};
    }
}
```

ПРИЛОЖЕНИЕ Б: ЛИСТИНГ ПРОГРАММЫ СЕРВЕРА

Приложение Б.1 – Файл «Server.cs»

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Collections.Generic;

namespace TextEditorServer
{
    class Server
    {
        public static readonly string wrRole = "Писатель";
        public static readonly string revRole = "Рецензент";
        public static int requestRange = 100;
        public static Dictionary<string, string> clientsList = new Dictionary<string, string>();
//имяроль

        public struct ReviewerInfo
        {
            public ReviewerInfo(string n, NetworkStream s) {
                name = n;
                stream = s;
            }
            public string name;
            public NetworkStream stream;
        }
        public static Dictionary<string, List<ReviewerInfo>> writerToReviewers =
            new Dictionary<string, List<ReviewerInfo>>();//имяписокРецензентов
        public static Dictionary<string, NetworkStream> writerStream = new Dictionary<string,
NetworkStream>(); //имяссылкаНаПоток
        public static Dictionary<string, string> writersData = new Dictionary<string, string>();
//имяданныеФайла

        private static readonly string ip = "127.0.0.1";
        private static readonly int port = 8888;
        private static TcpListener server = null;
        public static List<Thread> clientsThreads = new List<Thread>();

        private static void Main()
        {
            try
            {
                IPAddress localAddr = IPAddress.Parse(ip);
                server = new TcpListener(localAddr, port);
                server.Start();
                Console.WriteLine("Server has started\n");
while (true)
                {
                    TcpClient mainConnection = server.AcceptTcpClient();
                    ConnectNewClient(mainConnection, new FileProcessing());
                    RemoveDisconnectedClientsThreads();
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
```

```
            {
                if (server != null)
                    server.Stop();
            }
        }
        private static void ConnectNewClient(TcpClient connection, FileProcessing fileProcessing)
        {
            TheClient newClient = new TheClient(connection, fileProcessing);
            clientsThreads.Add(new Thread(newClient.Process));
            clientsThreads[clientsThreads.Count - 1].Start();
        }
        private static void RemoveDisconnectedClientsThreads()
        {
            for (int i = 0; i < clientsThreads.Count; i++)
            {
                if (!(clientsThreads[i].IsAlive)) clientsThreads.RemoveAt(i--);
}
        }
    }
}
```

Приложение Б.2 – Файл «TheClient.cs»

```
using System;
using System.Net.Sockets;

namespace TextEditorServer{
    public class TheClient{
        public TcpClient clientConnection = null;
        private ClientRequest clientRequest = null;
        private NetworkStream clientStream = null;
        private FileProcessing fileProcessing = null;
        public string nickName = null;

        public TheClient(TcpClient tcpConnection, FileProcessing f) {
            clientConnection = tcpConnection;
            fileProcessing = f;
        }
        public void Process()
        {
            do
            {
                clientStream = clientConnection.GetStream();
                clientRequest = new ClientRequest(this, fileProcessing);
                byte command = clientRequest.RecieveRequest(clientConnection.Connected);
                string response = clientRequest.ExecuteRequest(command);
                if (response != OperationResult.disconnect && response !=
OperationResult.nameIsBusy)
                {
                    clientRequest.SendResponse(response);
                }
                else if (response == OperationResult.nameIsBusy)
                {
                    clientRequest.SendResponse(response);
                    clientStream.Close();
                    clientConnection.Close();
                }
                else if(response == null)
                {
                    clientRequest.SendResponse("null response");
                }
                else
                {
```

```
                    clientRequest.SendResponse(response);
                    DisconnectClient(clientStream, GetNickName());
                }
            } while (clientConnection.Connected);
        }
        public void DisconnectClient(NetworkStream stream, string clientName)
        {
            stream.Close();
            clientConnection.Close();
            Server.clientsList.Remove(clientName);
            Console.WriteLine(clientName + " has disconnected");
            Console.WriteLine("Connected clients: " + Server.clientsList.Count + '\n');
        }
        public string GetNickName()
        {
            return nickName;
        }
        public void SetNickName(string nn)
        {
            nickName = nn;
        }
    }
}
```

Приложение Б.3 – Файл «ClientRequest.cs»

```
using System;
using System.Text;
using System.Net.Sockets;
using System.Linq;
using System.IO;

namespace TextEditorServer
{
    public class ClientRequest
    {
        //private TheClient theClient = null;
        private NetworkStream mainStream = null;
        private TheClient client = null;
        private string role = null;
        private string request = null;
        string responseString = null;
        private FileProcessing fileProcessing = null;
        BinaryReader reader = null;
        BinaryWriter writer = null;
        public ClientRequest(TheClient cl, FileProcessing f)
        {
            client = cl;
            mainStream = cl.clientConnection.GetStream();
            fileProcessing = f;
            reader = new BinaryReader(mainStream, Encoding.Unicode, true);
            writer = new BinaryWriter(mainStream, Encoding.Unicode, true);
        }

        public byte RecieveRequest(bool isConnected)
        {
            if (isConnected)
            {
                try
                {
                    StringBuilder builder = new StringBuilder();
byte[] data = new byte[64]; // буфер для получаемых данных
int bytes = 0;
```

```
                    do
                    {
                        bytes = mainStream.Read(data, 0, data.Length);
                        builder.Append(Encoding.Unicode.GetString(data, 0, bytes));
                    }
                    while (mainStream.DataAvailable);
                    request = builder.ToString();
                    return GetCommand(request);
                }
                catch (Exception e)
                {
                    Console.WriteLine(e.Message);
                    return 0;
                }
            }
            else return 127;
        }
        public void SendResponse(string response)
        {
            try
            {
                //writer.Write(response.Length);
                writer.Write(response);
                //byte[] data = Encoding.Unicode.GetBytes(response);
                //mainStream.Write(data, 0, data.Length);
}
            catch { return; }
            //тут проблема с записью. нужно ждать завершения записи или ждать этого на клиенте
}
        public string ExecuteRequest(byte command)
        {
            try
            {
                switch (command)
                {
                    case (byte)Commands.Writer.Connect:
                        responseString = ExecuteWriterConnect(GetNickName(request), mainStream,
GetRequestRange(request), GetRole(request));
                        break;
                    case (byte)Commands.Writer.CreateNewFile:
                        responseString = ExecuteCreateNewFile(GetFileName(request));
                        break;
                    case (byte)Commands.Writer.GetAvaliableFilesList:
                        responseString = ExecuteGetAvaliableFilesList();
                        break;
                    case (byte)Commands.Writer.StartWriting:
                        responseString = ExecuteStartWriting(GetNickName(request),
                            GetFileName(request));
                        if (responseString == "")
                        {
                            responseString = OperationResult.success;
                        }
                        int count = responseString.Length;
                        break;
                    case (byte)Commands.Writer.WriteData:
                        responseString = ExecuteWriteData(GetNickName(request),
                        Convert.ToInt32(GetOffset(request)),
                        GetData(request));
                        break;
                    case (byte)Commands.Writer.RemoveData:
                        responseString = ExecuteRemoveData(GetNickName(request),
                            Convert.ToInt32(GetOffset(request)),
                            Convert.ToInt32(GetCount(request)));
                        break;
```

```
                    case (byte)Commands.Writer.Save:
                        responseString = ExecuteSave(GetNickName(request),
                            GetFileName(request));
                        break;
                    case (byte)Commands.Writer.DeleteFile:
                        responseString = ExecuteDeleteFile(GetNickName(request),
                            GetFileName(request));
                        break;
                    case (byte)Commands.Writer.Disconnect:
                        responseString = ExecuteWriterDisconnect(GetNickName(request),
mainStream);
                        break;
                    case (byte)Commands.Reviewer.Connect:
                        responseString = ExecuteReviewerConnect(GetNickName(request));
                        break;
                    case (byte)Commands.Reviewer.GetConnectedWritersList:
                        responseString = ExecuteGetConnectedWritersList();
                        break;
                    case (byte)Commands.Reviewer.StartWatching:
                        responseString = ExecuteStartWatching(GetNickName(request),
                            GetWriterName(request), mainStream);
                        break;
                    case (byte)Commands.Reviewer.WriteComment:
                        responseString = ExecuteWriteComment(GetWriterName(request),
                            GetNickName(request),
                            GetData(request));
                        break;
                    case (byte)Commands.Reviewer.Disconnect:
                        responseString = ExecuteReviewerDisconnect(GetNickName(request),
                            GetWriterName(request), mainStream);
                        break;
                    default:
                        responseString = OperationResult.badCase;
                        break;
                }
            }
            catch(Exception e) { Console.WriteLine(e.Message); }
            return responseString;
        }
        private string ExecuteWriterConnect(string cName, NetworkStream stream, int rr, string r)
{
            if (!(Server.clientsList.ContainsKey(cName)))
            {
                Server.requestRange = rr;
                client.SetNickName(cName);
                role = r;
                Server.clientsList.Add(cName, role);
                Server.writerStream.Add(cName, stream);
                Console.WriteLine(cName + " has connected");
                return fileProcessing.ConnectWriter(cName);
            }
            else { return OperationResult.nameIsBusy; }
        }
        private string ExecuteCreateNewFile(string fName)
        {
            return fileProcessing.CreateFile(fName);
        }
        private string ExecuteGetAvaliableFilesList()
        {
            return fileProcessing.GetAllFiles();
        }
        private string ExecuteStartWriting(string cName , string fName)
        {
            return fileProcessing.StartWriting(cName, fName);
```

```
        }
        private string ExecuteWriteData(string cName, int offSet, string data)
        {
            string dataToReviewer = fileProcessing.WriteData(cName, offSet, data);
            if (Server.writerToReviewers.ContainsKey(cName))
            {
                dataToReviewer = "<" + cName + ">" + dataToReviewer;
                for (int i = 0; i < Server.writerToReviewers[cName].Count; i++)
                {
                    byte[] bytes = Encoding.Unicode.GetBytes(dataToReviewer);
                    Server.writerToReviewers[cName].ElementAt(i).stream.Write(bytes, 0,
bytes.Length);
                }
            }
            return OperationResult.success;
        }
        private string ExecuteRemoveData(string cName, int offSet, int count)
        {
            string dataToReviewer = fileProcessing.RemoveData(cName, offSet, count);
            if (Server.writerToReviewers.ContainsKey(cName))
            {
                for (int i = 0; i < Server.writerToReviewers[cName].Count; i++)
                {
                    dataToReviewer = "<" + cName + ">" + dataToReviewer;
                    byte[] bytes = Encoding.Unicode.GetBytes(dataToReviewer);
                    Server.writerToReviewers[cName].ElementAt(i).stream.Write(bytes, 0,
bytes.Length);
                }
            }
            return OperationResult.success;
        }
        private string ExecuteDeleteFile(string cName, string fName)
        {
            return fileProcessing.DeleteFile(cName, fName);
        }
        private string ExecuteSave(string cName, string fName)
        {
            return fileProcessing.Save(cName, fName);
        }
        private string ExecuteWriteComment(string wrName, string cName, string data)
        {
            string dataToWriter = "<" + cName + ">" + "<" + data + ">";
            if (Server.writerToReviewers.ContainsKey(wrName) &
                Server.writerToReviewers[wrName].Contains(new Server.ReviewerInfo(cName,
mainStream)))
            {
                byte[] bytes = Encoding.Unicode.GetBytes(dataToWriter);
                Server.writerStream[wrName].Write(bytes, 0, bytes.Length);
            }
            return OperationResult.success;
        }
        private string ExecuteWriterDisconnect(string cName, NetworkStream stream)
        {
            Server.writerStream.Remove(cName);
            fileProcessing.DeleteFromWatch(cName, stream);
            return fileProcessing.DisconnectWriter(cName);
        }
        private string ExecuteReviewerConnect(string cName)
        {
            Server.requestRange = GetRequestRange(request);
            client.SetNickName(cName);
            role = GetRole(request);

            if (!(Server.clientsList.ContainsKey(cName)))
```

```
            {
                Server.clientsList.Add(cName, role);
                Console.WriteLine(cName + " has connected");
                return OperationResult.success;
            }
            else { return OperationResult.nameIsBusy; }
        }
        private string ExecuteStartWatching(string cName, string wrName, NetworkStream stream)
        {
            return fileProcessing.AddToWatch(wrName, cName, stream);
        }
        private string ExecuteGetConnectedWritersList()
        {
            string writers = "";

            for (int i = 0; i < Server.clientsList.Count; i++)
            {
                if (Server.clientsList.ElementAt(i).Value == Server.wrRole)
                    writers += Server.clientsList.ElementAt(i).Key + ">";
            }
            if (writers == "") return OperationResult.noConnectedWriters;
            return writers.TrimEnd('>');
        }
        private string ExecuteReviewerDisconnect(string cName, string wrName, NetworkStream
stream) {
            fileProcessing.DeleteFromWatch(wrName, cName, stream);
            return OperationResult.disconnect;
        }
        public byte GetCommand(string req)
        {
            string[] info = req.Split('>');
            return (byte)Convert.ToSByte(info[0].TrimStart('<'));
        }
        public int GetRequestRange(string req)
        {
            string[] info = req.Split('>');
            return (byte)Convert.ToInt32(info[1].TrimStart('<'));
        }
        public string GetNickName(string req)
        {
            string[] info = req.Split('>');
            return info[2].TrimStart('<');
        }
        public string GetWriterName(string req)
        {
            string[] info = req.Split('>');
            return info[4].TrimStart('<');
}
        //public string GetWriterNameReviewer(string req)
        //{
        //    string[] info = req.Split('>');
        //    return info[4].TrimStart('<');
        //}
        public string GetRole(string req)
        {
            string[] info = req.Split('>');
            return info[3].TrimStart('<');
        }
        public string GetFileName(string req)
        {
            string[] info = req.Split('>');
            string str = info[4].TrimStart('<');
            return str;
        }
```

```
        public string GetOffset(string req)
        {
            string[] info = req.Split('>');
            return info[5].TrimStart('<');
        }
        public string GetCount(string req)
        {
            string[] info = req.Split('>');
            return info[6].TrimStart('<');
        }
        public string GetData(string req)
        {
            string[] info = req.Split(new char[] { '>' });
            info[8] = info[8].TrimEnd('>');
            return info[8].TrimStart('<');
        }
        private string ExecuteEndWatching(string cName, string wrName, NetworkStream stream)
        {
            return fileProcessing.DeleteFromWatch(wrName, cName, stream);
        }
        public int GetChange(string ch)
        {
            if (ch == "")
            {
                return 0;
            }
            else return 1;
}
    }
}
```

Приложение Б.4 – Файл «FileProcessing.cs»

```
using System.IO;
using System;
using System.Linq;
using System.Net.Sockets;
using System.Collections.Generic;

namespace TextEditorServer
{
    public class FileProcessing
    {
        private string filePath = Directory.GetCurrentDirectory() + @"\files";
        private DirectoryInfo fileDirectory;
        private string fileName = null;

        public string ConnectWriter(string cName) {
            try
            {
                Server.writersData.Add(cName, null);
                Server.writersData[cName] = "";
                filePath += @"\" + cName;
                fileDirectory = new DirectoryInfo(filePath);
                if (!fileDirectory.Exists) fileDirectory.Create();
                return OperationResult.success;
            }
            catch (Exception e)
            {
                return e.Message;
            }
        }
        public string AddToWatch(string wrName, string revName, NetworkStream stream)
```

```
        {
            try
            {
                if (!Server.writerToReviewers.ContainsKey(wrName))
                {
                    Server.writerToReviewers.Add(wrName, new List<Server.ReviewerInfo>());
                    Server.writerToReviewers[wrName].Add(new Server.ReviewerInfo(revName,
stream));
                    return Server.writersData[wrName];
                }
                else if (!Server.writerToReviewers[wrName].Contains(new
Server.ReviewerInfo(revName, stream)))
                {
                    Server.writerToReviewers[wrName].Add(new Server.ReviewerInfo(revName,
stream));
                    return Server.writersData[wrName];
                }
                else if(Server.writerToReviewers[wrName].Contains(new
Server.ReviewerInfo(revName, stream)))
                {
                    return Server.writersData[wrName];
                }
                return OperationResult.commonError;
            }
            catch (Exception e)
            {
                return e.Message;
            }
        }
        public string DeleteFromWatch(string wrName, string revName, NetworkStream stream)
        {
            try
            {
                if (Server.writerToReviewers.ContainsKey(wrName))
                {
                    Server.writerToReviewers[wrName].Remove(new Server.ReviewerInfo(revName,
stream));
                    Server.writerToReviewers.Remove(wrName);
                    return OperationResult.success;
                }
                return OperationResult.commonError;
            }
            catch (Exception e)
            {
                return e.Message;
            }
        }
        public string DeleteFromWatch(string wrName, NetworkStream stream)
        {
            try
            {
                if (Server.writerToReviewers.ContainsKey(wrName))
                {
                    Server.writerToReviewers.Remove(wrName);
                    return OperationResult.success;
                }
                return OperationResult.commonError;
            }
            catch (Exception e)
            {
                return e.Message;
            }
        }
        public string GetAllFiles()
```

```
{
    try
    {
        string files = "";
        DirectoryInfo currentDirectory = new DirectoryInfo(filePath);
        foreach (var item in currentDirectory.GetFiles())
        {
            files += item.Name + ">";
        }
        if (files == "")
        {
            return OperationResult.noFiles;
        }
        else
        {
            return files.TrimEnd('>');
        }
    }
    catch (Exception e)
    {
        return e.Message;
    }
}
public string CreateFile(string fName)
{
    try
    {
        if (File.Exists(filePath + @"\" + fName))
        {
            return OperationResult.exist;
        }
        else
        {
            using (File.Create(filePath + @"\" + fName)) { } ;
        }
        return fName;
    }
    catch (Exception e)
    {
        return e.Message;
    }
}
public string StartWriting(string cName, string fName)
{
    try
    {
        string data = "";
        data = File.ReadAllText(filePath + @"\" + fName);
        Server.writersData[cName] = data;
        fileName = fName;
        if (Server.writersData[cName] != data) return OperationResult.StartWritingError;
        return data;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        return OperationResult.StartWritingError;
    }
}
public string WriteData(string cName, int offset, string data)
{
    try
    {
        Server.writersData[cName] = Server.writersData[cName].Insert(offset, data);
```

```
                return "<w>" + "<" + offset + ">" + "<" + data + ">";
            }
            catch (Exception e)
            {
                return e.Message;
            }
        }
        public string RemoveData(string cName, int offset, int count)
        {
            try
            {
                Server.writersData[cName] = Server.writersData[cName].Remove(offset, count);
                return "<r>" + "<" + offset + ">" + "<" + count + ">";
            }
            catch (Exception e)
            {
                return e.Message;
            }
        }
        public string Save(string cName, string fName)
        {
            try
            {
                File.WriteAllText(filePath + @"\" + fName, Server.writersData[cName]);
                return OperationResult.success;
            }
            catch (Exception e)
            {
                return e.Message;
            }
        }
        public string DeleteFile(string cName, string fName)
        {
            try
            {
                File.Delete(filePath + @"\" + fName);
                return OperationResult.success;
            }
            catch (Exception e)
            {
                return e.Message;
            }
        }
        public string DisconnectWriter(string cName)
        {
            try
            {
                Server.writersData.Remove(cName);
                return OperationResult.disconnect;
            }
            catch(Exception e)
            {
                return e.Message;
            }
        }
    }
}
```

Приложение Б.5 – Файл «OperationResult.cs»

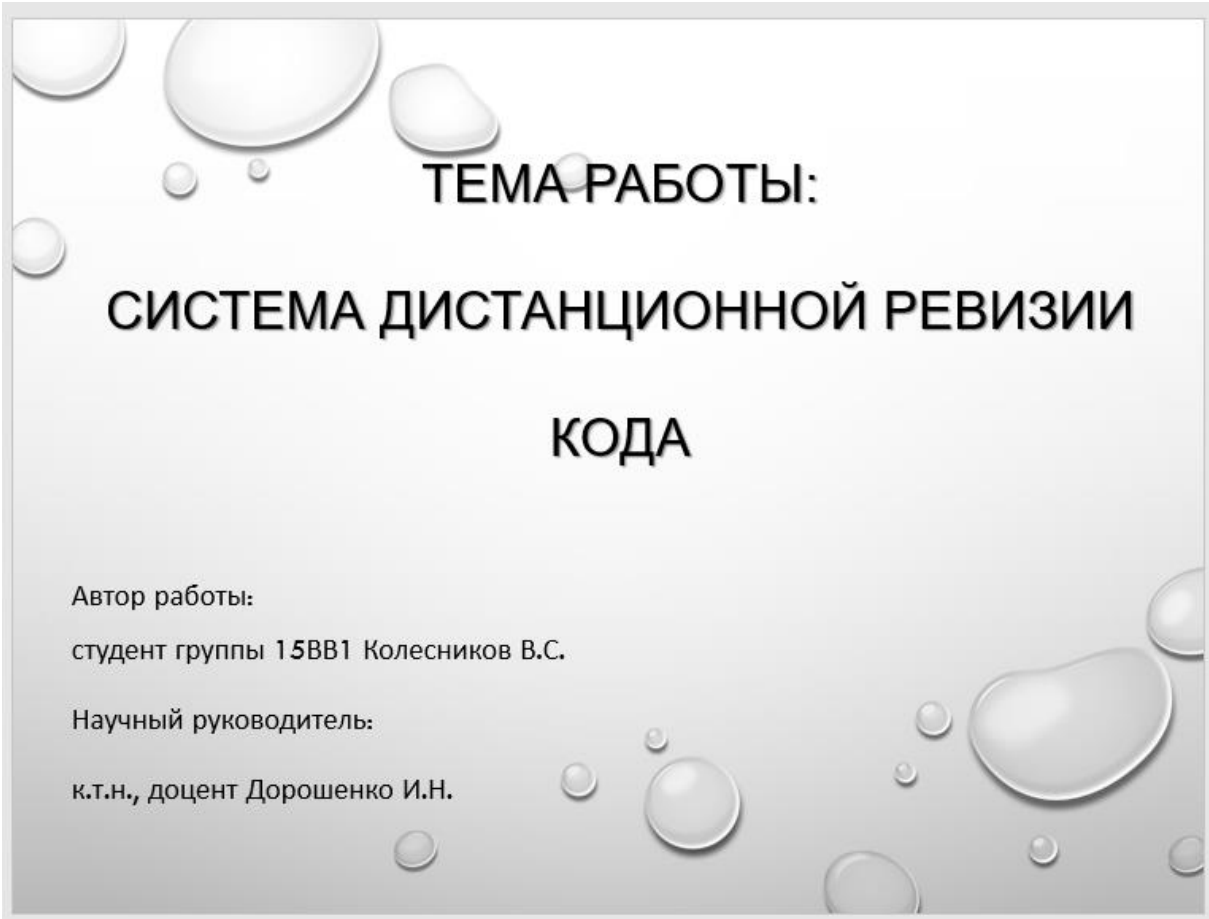
```
namespace TextEditorServer
{
    public static class OperationResult
```

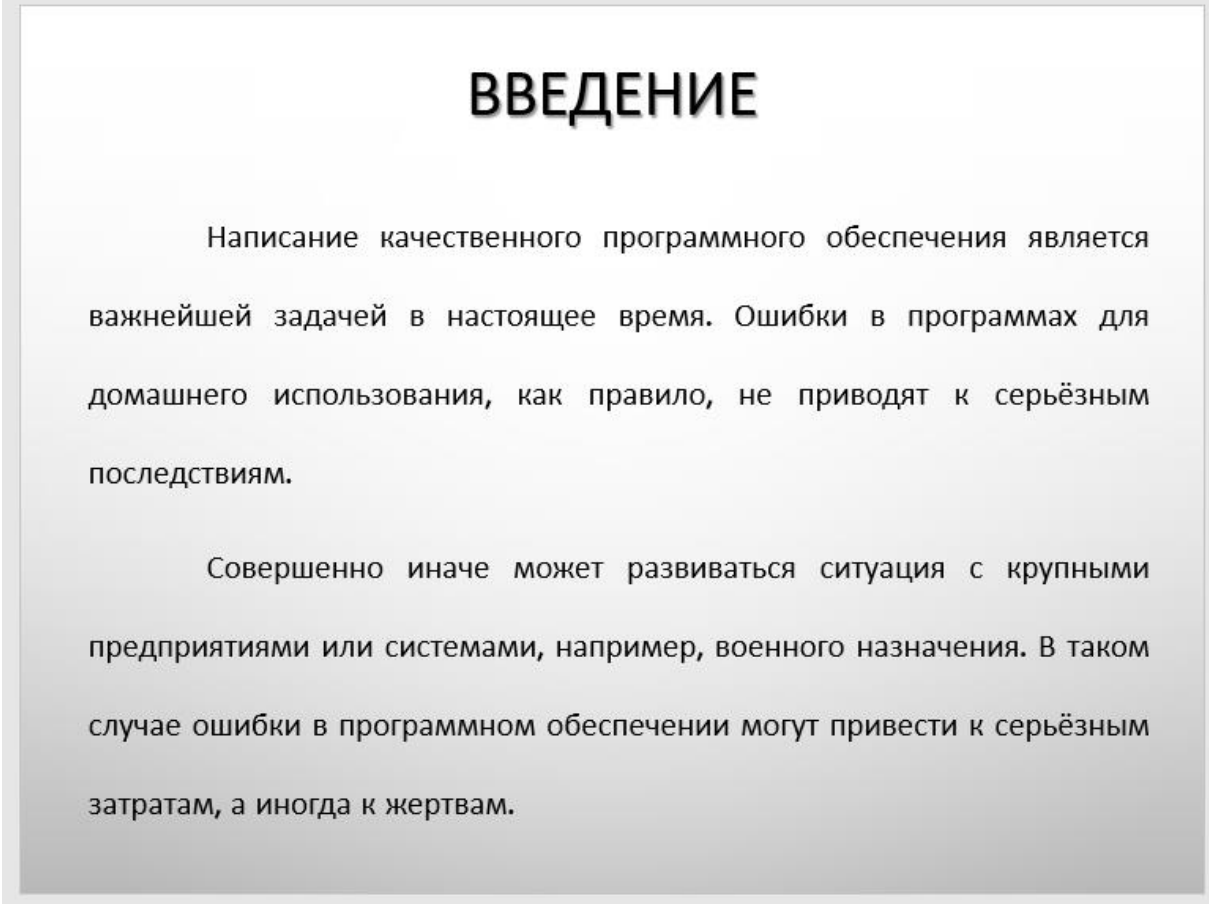
```
    {
        public static string success = "Операция выполнена успешно.";
        public static string commonError = "Неизвестная ошибка.";
        public static string disconnect = "Соединение закрыто успешно.";
        public static string noFiles = "На сервере нет файлов.";
        public static string badCase = "Необработанный case.";
        public static string noConnectedWriters = "Нет подключенных к серверу писателей.";
        public static string exist = "Файл уже существует.";
        publicstaticstringnameIsBusy = "Данное имя пользователя уже занято. Выберете другое.";
publicstaticstringcouldNotConnect = "Невозможно подключится к серверу.";
publicstaticstringcreateFileError = "Ошибка создания файла.";
public static string StartWritingError = "Ошибка start writing";
        public static string saveError = "Ошибка сохранения файла";
        public static string fileNameError = "Ошибка имени файла";
        public static string choiseFileError = "Файл не выбран.";
        public static string clientIsAlreadyDisconnected = "Клиент уже был отключен.";
        public static string alreadyWatching = "Этот писатель уже просматривается.";
    }
}
```

Приложение Б.6 – Файл «Commands.cs»

```
namespace TextEditorServer
{
    public static class Commands
    {
        public enum Writer : byte
        {
            Connect = 0,
            Disconnect,
            GetAvaliableFilesList,
            CreateNewFile,
            StartWriting,
            WriteData,
            RemoveData,
            DeleteFile,
            Save,
        };
        public enum Reviewer : byte
        {
            Changes = 10,
            Connect,
            WriteComment,
            RemoveComment,
            GetConnectedWritersList,
            StartWatching,  //<reviewerName><writerName>
            EndWatching,
            Disconnect,
};
    }
}
```

ПРИЛОЖЕНИЕ В: ПРЕЗЕНТАЦИЯ





ЦЕЛЬ И АКТУАЛЬНОСТЬ

Целью данной работы является разработка системы, позволяющей проводить рецензирование программного кода, а также, обеспечивать возможность редактирования файлов.

Данная разработка призвана упростить процесс ревизии программного кода, который может быть применен в различных областях, в том числе и в образовательных целях. С помощью такой разработки станет удобно работать в удалённом режиме, набрав небольшую группу учеников.

ЗАДАЧИ

- изучить современные подходы к повышению качества программного кода;
- провести анализ предметной области на примере существующих систем и платформ, обеспечивающих рецензирование кода;
- определить методы и средства, используемые при разработке компонентов приложения;
- на основе вышесказанного разработать компоненты системы.

СОВРЕМЕННЫЕ МЕТОДЫ ПОВЫШЕНИЯ КАЧЕСТВА ПРОГРАММНОГО КОДА

- программные методы;
- непрограммные методы.

НЕПРОГРАММНЫЕ МЕТОДЫ

- просмотр программного кода (ревизия);
- парное программирование.

ОБЩАЯ СХЕМА ПРИЛОЖЕНИЯ

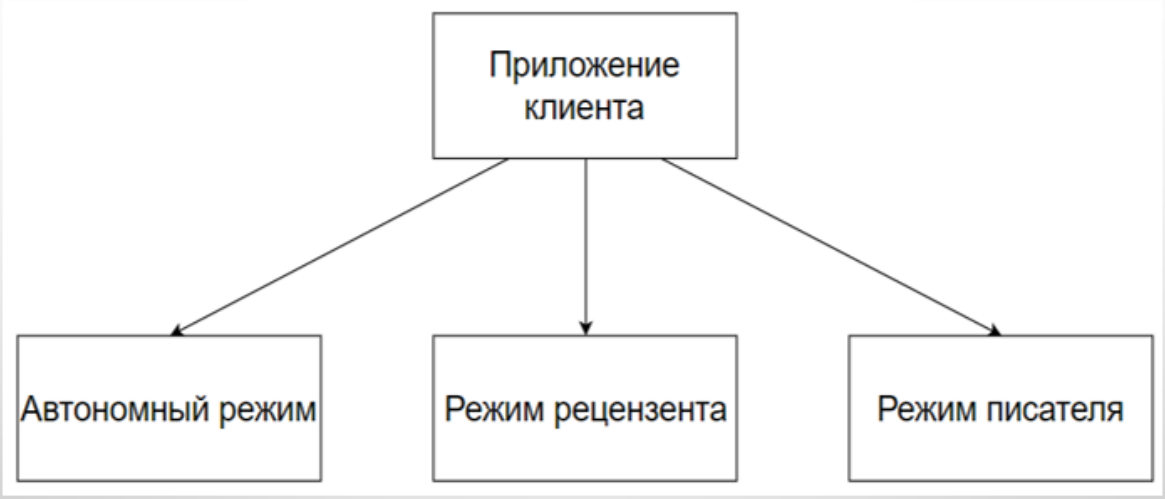
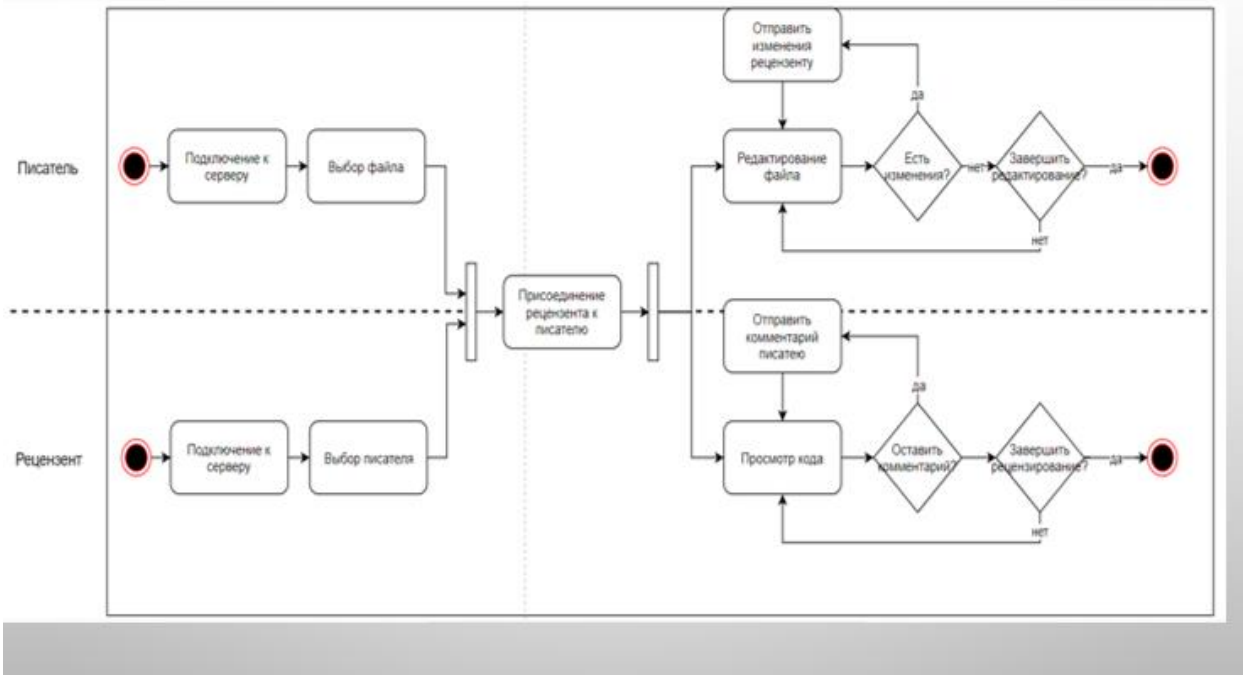


ДИАГРАММА ДЕЯТЕЛЬНОСТИ



ВНЕШНИЙ ВИД ОКНА ПИСАТЕЛЯ

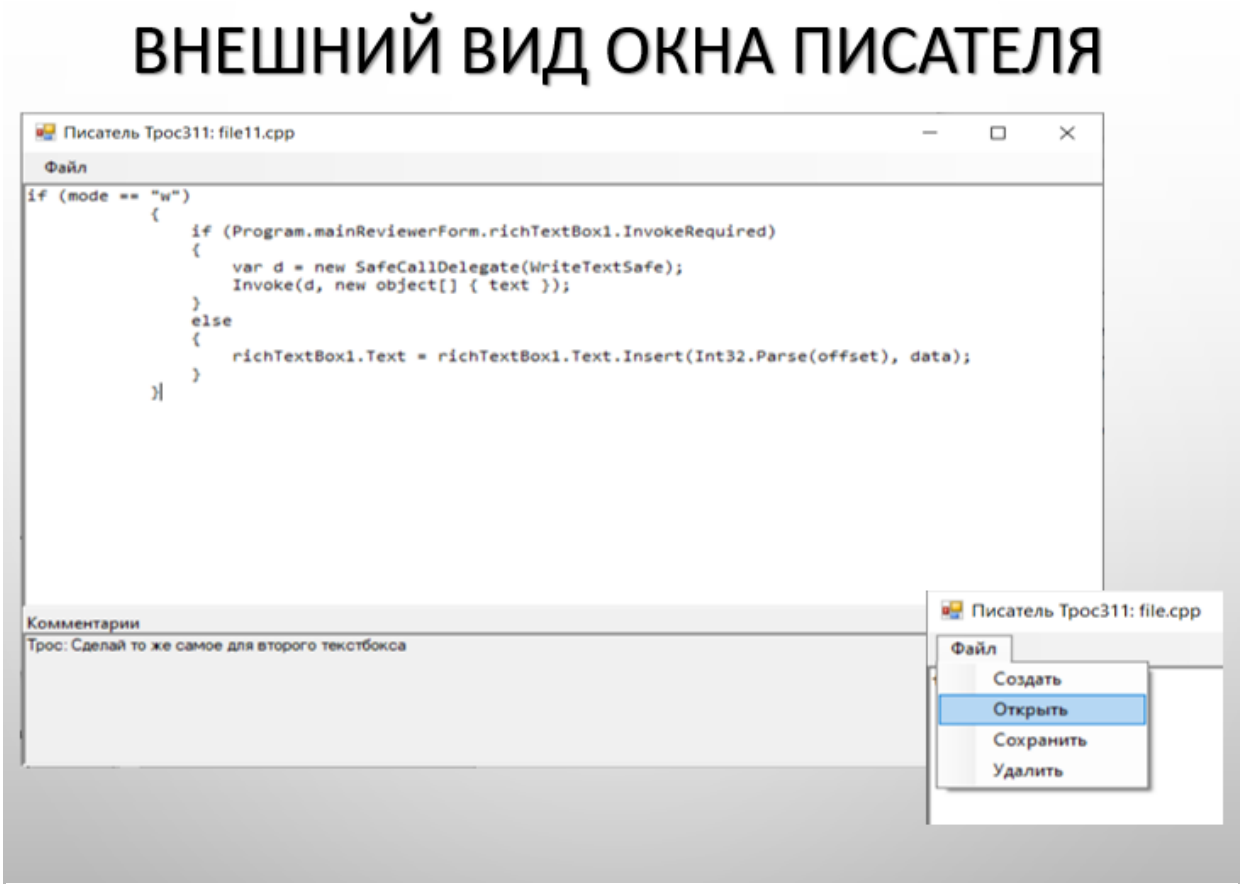
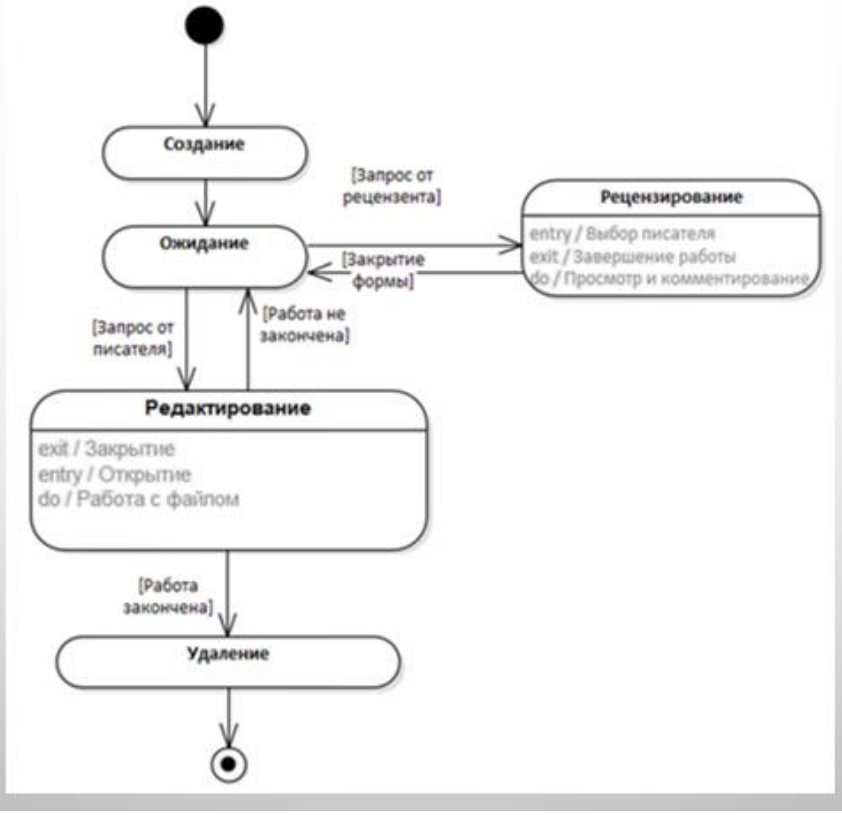
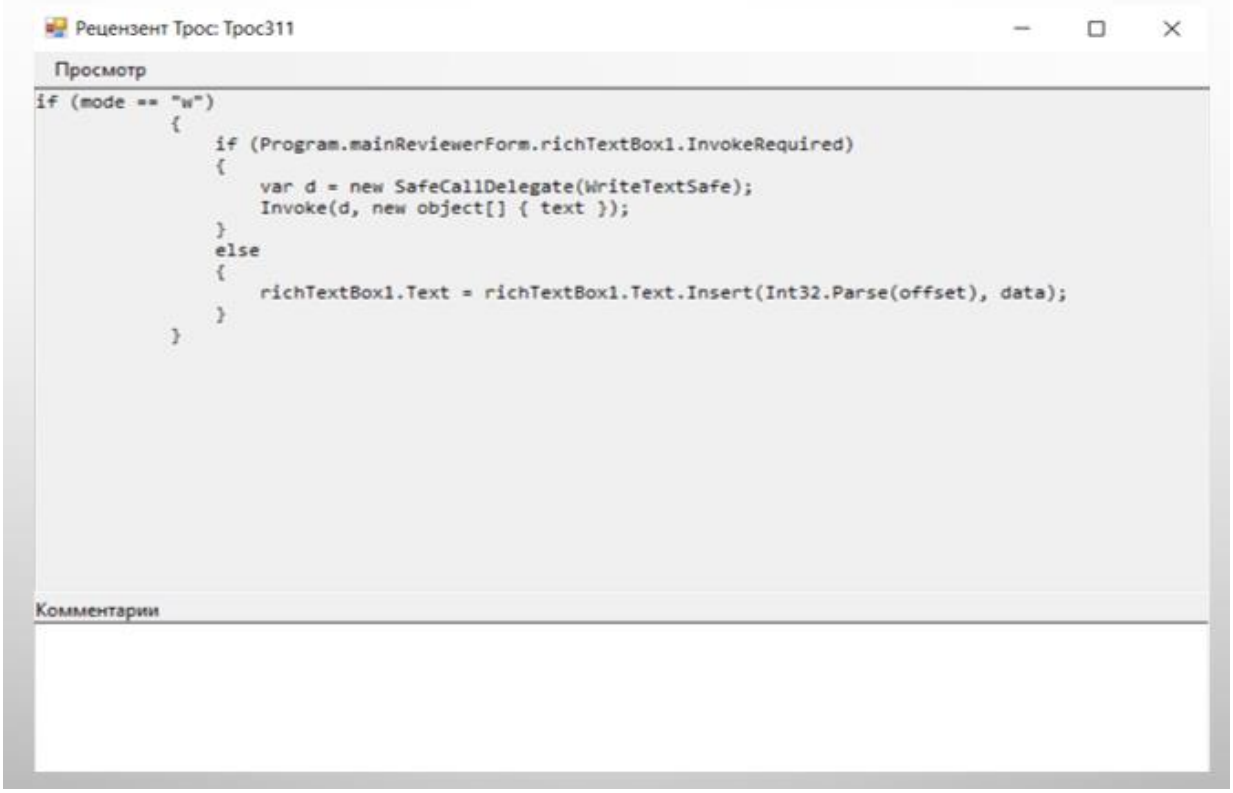


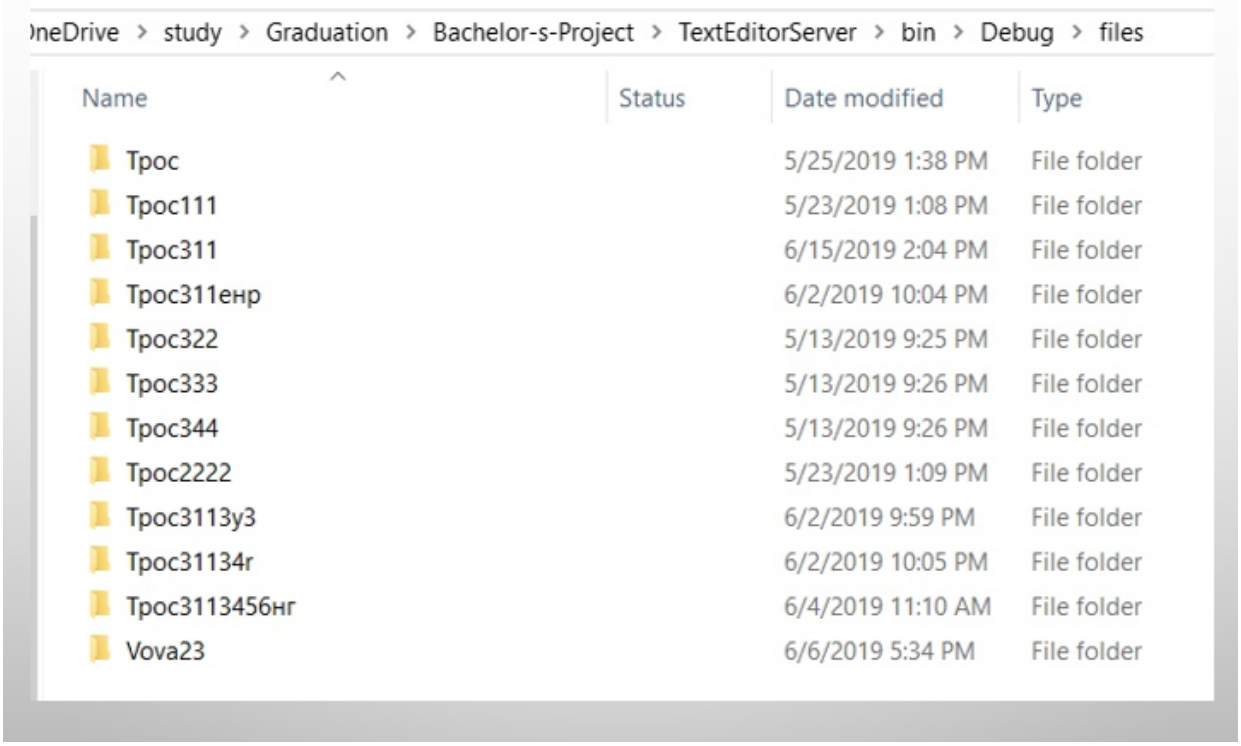
ДИАГРАММА СОСТОЯНИЙ ФАЙЛА



ВНЕШНИЙ ВИД ОКНА РЕЦЕНЗЕНТА



РОЛЬ СЕРВЕРА



РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ РАБОТЫ

- изучены современные подходы к повышению качества программного кода;
- определены методы и средства, используемые при разработке компонентов приложения;
- на основе вышесказанного разработаны компоненты системы;
- описан процесс эксплуатации системы;

ЗАКЛЮЧЕНИЕ

В данной работе была реализована система дистанционной ревизии кода, позволяющая проводить рецензирование и редактирование файлов в режиме реального времени.

Поставленная цель была достигнута . Все задачи были решены в полном объеме.

СПАСИБО
ЗА ВНИМАНИЕ!