

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

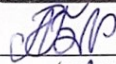
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

МЕССЕНДЖЕР С ШИФРОВАНИЕМ

Студент

 30.05.25
(подпись, дата)

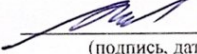
Брянцев Артём Гердович
(ФИО полностью)

Руководитель

 30.05.25
(подпись, дата)

Акифьев И.В.
(фамилия, инициалы)

Нормоконтролёр

 30.05.25
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

16 июня 2025 г. № 31)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2025

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 06 » 05 20 25 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Брянцев Артём Гердович гр. 21ВВП1 факультета вычислительной техники

2. Тема работы Мессенджер с шифрованием

Тема утверждена приказом ПГУ № 601/о от 06.05.2025 г.

3. Руководитель бакалаврской работы доцент кафедры ВТ ПГУ Акифьев И.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать клиент-серверное приложение. Язык программирования — Java, графический интерфейс — Java Swing, среда разработки IntelliJ IDEA. Серверная часть должна обеспечивать надёжное взаимодействие с клиентским интерфейсом и реализовывать следующие функциональные возможности:

1. Авторизацию пользователей;

2. Отправка сообщений на сервер в зашифрованном виде;

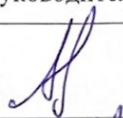
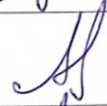
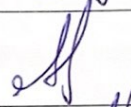
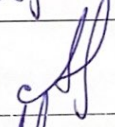
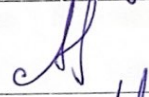
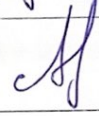
3. Дешифрование сообщений, полученных с сервера;

4. Хранение и обработку данных в файле;

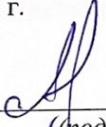
5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметной области: изучить существующие аналоги цифровых торговых платформ, проанализировать архитектурные решения и подходы к построению архитектуры. На основе анализа выбрать язык программирования и среду разработки. Описать архитектуру взаимодействия компонентов системы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	5%	06.05.2025 – 09.05.2025	
2	Анализ требований	10%	10.05.2025 – 13.05.2025	
3	Выбор технологий, применяемых при проектировании и разработке	10%	13.05.2025 – 15.05.2025	
4	Разработка приложения	35%	16.05.2025 – 21.05.2025	
5	Создание графического интерфейса	25%	22.05.2025 – 24.05.2025	
6	Оформление пояснительной записки	15%	25.05.2025 – 30.05.2025	

Дата выдачи задания "06" 05 2025 г.

Руководитель бакалаврской работы  06.05.2025 Акифьев И.В.
(подпись, дата)

Задание к исполнению принял студент  06.05.25 Брянцев А.Г.
(подпись, дата)

Работу к защите допустить
Декан факультета  11.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 96 листов, 20 рисунков, 1 таблицу, 14 источников, 3 приложения

КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, МЕССЕНДЖЕР, ТЕСТИРОВАНИЕ, JAVA, SWING

Цель работы – разработать клиент-серверное приложение для обмена зашифрованными сообщениями.

Разрабатываемая система представляет собой защищённый мессенджер, предназначенный для безопасного обмена сообщениями между пользователями. Клиентская часть приложения позволяет пользователям устанавливать соединение с сервером, отправлять и получать сообщения, шифруя их с использованием современных криптографических алгоритмов. В системе реализована поддержка как симметричного шифрования (AES), так и асимметричного (RSA), что обеспечивает надёжную защиту передаваемой информации.

В результате выполнения выпускной квалификационной работы была разработана клиент-серверная система на языке Java. Серверная часть реализована с использованием многопоточной архитектуры: для обработки каждого подключённого клиента создаётся отдельный поток. Клиентская часть построена по архитектурному шаблону MVC. Таким образом, разработанная система позволяет обеспечить базовую защищённую коммуникацию между пользователями и может служить основой для дальнейшего расширения, включая реализацию истории переписки, аутентификации пользователей и интеграции более устойчивых криптографических протоколов.

					ПГУ1.090301.05.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Мессенджер с шифрованием Пояснительная записка	Лит.	Лист	Листов
Разраб.		Брянцев А.Г.		30.06.21				
Провер.		Акифьев И.В.		30.06.21			2	96
Н. Контр		Бычков А.С.		30.06.21		ФВТ, каф. ВТ гр. 21ВВП1		
Утверд.		Митрохин М.А.		30.06.21				

Содержание

Перечень сокращений и обозначений.....	5
Введение	6
1 Мессенджер с шифрованием	7
1.1 Анализ предметной области	7
1.2 Анализ аналогов	9
1.2.1 Signal.....	9
1.2.2 Telegram (секретные чаты)	11
1.2.3 Threema.....	12
1.2.4 WhatsApp.....	13
1.2.5 Matrix	14
1.2.6 Retroshare.....	15
1.2.7 Вывод.....	16
1.3 Постановка задачи	17
1.4 Требования к аппаратуре и программному обеспечению	19
1.4.1 Клиентская часть	19
1.4.2 Серверная часть	19
2 Архитектура приложения	20
3 Выбор инструментов разработки	27
3.1 Выбор языка программирования	27
3.2 Выбор среды разработки.....	28
4 Реализация приложения	30
4.1 Описание функционала.....	30
4.2 Описание работы приложения	33
4.3 Описание интерфейса приложения.....	35
4.4 Тестирование приложения.....	38

5 Безопасность жизнедеятельности	40
5.1 Правила и техника безопасности при работе с ПК	40
5.2 Правила и техника безопасности при работе со смартфоном.....	41
Заключение.....	43
Список использованных источников	45
Приложение А.....	47
Приложение Б	53
Приложение В	91

Перечень сокращений и обозначений

В данной работе применяются следующие сокращения и обозначения

AES – симметричный алгоритм шифрования, применяемый для защиты данных

GUI – графический пользовательский интерфейс

IDE – интегрированная среда разработки программного обеспечения

IP – протокол, обеспечивающий адресацию и маршрутизацию пакетов данных в сети

JDK – комплект средств разработки программ на языке Java

JRE – среда выполнения Java-программ

MVC – шаблон проектирования, разделяющий приложение на модель, представление и контроллер

PKI – инфраструктура открытых ключей, обеспечивающая управление цифровыми сертификатами и ключами

RSA – алгоритм асимметричного шифрования, используемый для безопасной передачи ключей

SKZI – средства криптографической защиты информации

Swing – библиотека Java для создания графического интерфейса

TCP/IP – стек сетевых протоколов, используемый для передачи данных

Введение

В современном цифровом обществе, где значительная часть общения и обмена информацией осуществляется через интернет, защита пользовательских данных становится критически важной задачей. Особенно остро стоит вопрос безопасности личной переписки, которая нередко может содержать конфиденциальную информацию. Мессенджеры, как один из самых распространённых инструментов общения, требуют особого внимания в плане обеспечения приватности и сохранности передаваемых сообщений.

Данный дипломный проект посвящён разработке защищённого мессенджера, реализованного на языке программирования Java с использованием архитектуры клиент-сервер. Основной акцент сделан на обеспечении безопасности обмена сообщениями между пользователями за счёт использования современных криптографических методов. В ходе реализации применяются алгоритмы асимметричного и симметричного шифрования, в том числе организация передачи сообщений с помощью сессионных ключей. Клиентская часть приложения обеспечивает удобный пользовательский интерфейс, а серверная обрабатывает входящие сообщения и маршрутизирует их между участниками общения.

Разработка данного мессенджера позволяет продемонстрировать на практике принципы построения защищённых коммуникационных систем, изучить основы криптографии и сетевого программирования, а также реализовать взаимодействие между различными компонентами приложения. Проект может быть использован как учебная разработка, демонстрирующая базовые механизмы шифрования и передачи данных, а также как основа для последующего расширения функциональности или интеграции в корпоративные среды с повышенными требованиями к защите информации.

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Мессенджер с шифрованием

1.1 Анализ предметной области

В современном цифровом обществе безопасность и конфиденциальность личной переписки становятся ключевыми требованиями к любому средству коммуникации. Особенно остро этот вопрос стоит в условиях роста числа кибератак, фишинга, перехвата сообщений и несанкционированного доступа к данным пользователей. Именно поэтому шифрованные мессенджеры становятся не просто удобным средством общения, но и необходимым инструментом обеспечения цифровой безопасности.

Мессенджер с шифрованием необходим различным категориям пользователей. Частные лица, ценящие конфиденциальность, стремятся обезопасить личную переписку от несанкционированного доступа. Журналисты и правозащитники, особенно работающие в условиях ограниченной свободы слова, нуждаются в надёжных каналах связи с источниками информации. В деловой сфере компании регулярно обмениваются конфиденциальными данными – коммерческими предложениями, клиентской информацией, внутренними стратегиями – и утечка таких сведений может повлечь серьёзные финансовые и репутационные убытки. Также потребность в защищённой коммуникации испытывают государственные структуры и органы управления, которым важно сохранить безопасность служебной переписки. Студенты и ИТ-специалисты используют подобные проекты для практического освоения криптографии, построения сетевого взаимодействия и разработки архитектуры защищённых систем.

Использование мессенджера с шифрованием позволяет обеспечить конфиденциальность передаваемых данных, предотвращая доступ к содержимому сообщений со стороны третьих лиц. Кроме того, реализуются механизмы, обеспечивающие целостность сообщений, исключая возможность их искажения в процессе передачи. В образовательной сфере такой проект служит

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

отличной возможностью изучения основ сетевого программирования, применения криптографических методов, создания клиент-серверной архитектуры и разработки пользовательских интерфейсов.

Разрабатываемый мессенджер с функцией шифрования не нарушает действующее законодательство Российской Федерации. Напротив, он реализует право граждан на защиту личной информации и конфиденциальность переписки. В соответствии со статьёй 23 Конституции РФ, каждый имеет право на неприкосновенность частной жизни, личную и семейную тайну, а также тайну переписки, телефонных переговоров и иных сообщений. Это право может быть ограничено только на основании судебного решения. Также важным нормативным актом является Федеральный закон № 149-ФЗ «Об информации, информационных технологиях и о защите информации», который допускает использование шифровальных (криптографических) средств, если они не направлены на нарушение закона или сокрытие противоправной деятельности.

Часто в подобных проектах упоминается так называемый «пакет Яровой» (Федеральные законы № 374-ФЗ и № 375-ФЗ от 6 июля 2016 года), который устанавливает обязанности для организаторов распространения информации (ОРИ), включая хранение данных пользователей и предоставление ключей шифрования по требованию уполномоченных органов. Однако в рамках данного проекта не создаётся сервис массового распространения информации и не производится её хранение на серверах – мессенджер функционирует в формате учебного клиент-серверного приложения, не подлежащего обязательной регистрации в Роскомнадзоре как ОРИ. Программа не нарушает требований к техническим средствам шифрования, так как не использует сертифицированные СКЗИ (средства криптографической защиты информации), обязательные для некоторых сфер (например, при передаче государственной тайны).

Таким образом, реализация защищённой передачи сообщений в рамках данного проекта полностью соответствует российскому законодательству, не нарушает конституционные нормы и не требует дополнительных лицензий или

разрешений при использовании в учебных и исследовательских целях. Для отображения вариантов использования на рисунке 1 представлена диаграмма прецедентов.

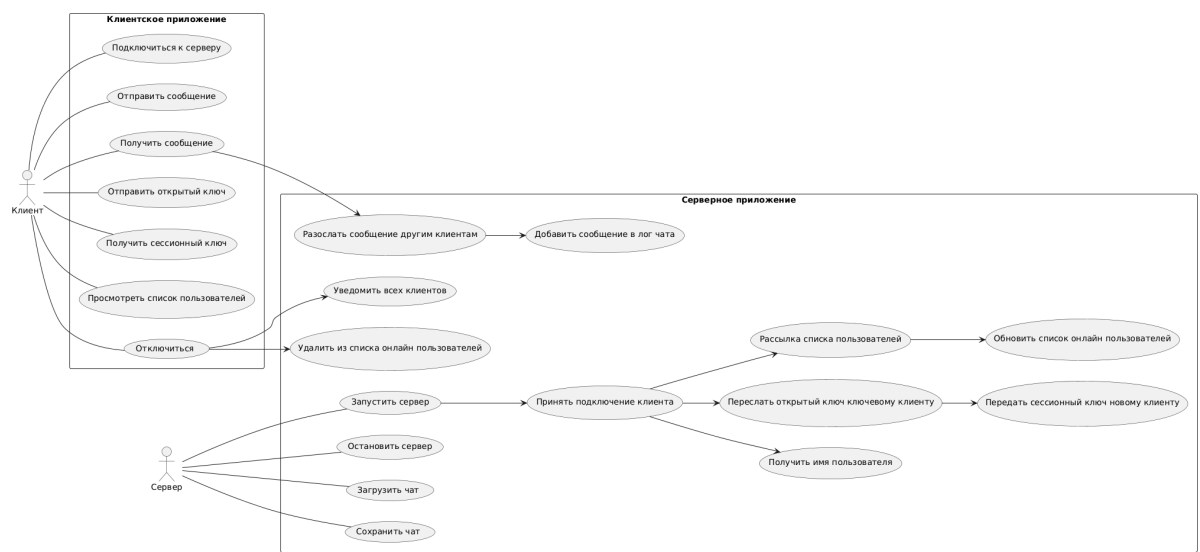


Рисунок 1 – UML-диаграмма прецедентов

1.2 Анализ аналогов

На рынке существует множество мессенджеров, использующих различную степень шифрования. Ниже представлены как прямые аналоги (выполняют те же задачи), так и косвенные (содержат элементы, сходные по логике или технологии).

1.2.1 Signal

Мессенджер с полной поддержкой зашифрованных голосовых и текстовых сообщений, видео- и аудиозвонков. Поддерживает групповые чаты, вложения, таймер самоуничтожения сообщений. На рисунке 2 показано окно чата Signal.

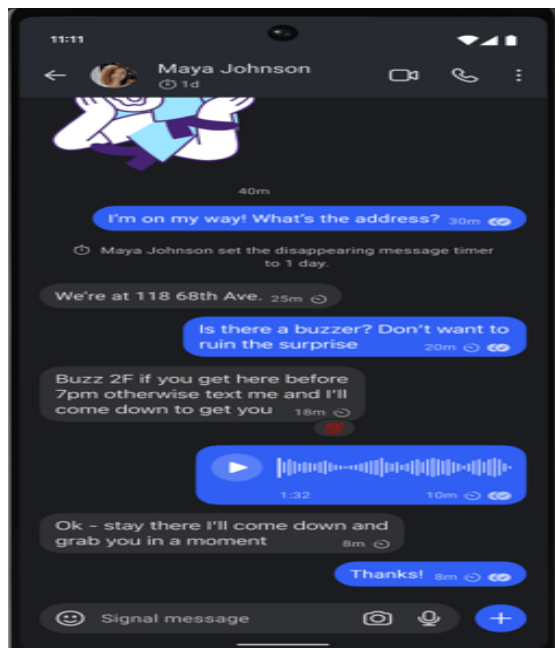


Рисунок 1 – Мессенджер Signal

Оптимален для пользователей, для которых конфиденциальность – приоритет. Подходит для индивидуального и профессионального использования. Используемые технологии и алгоритмы: Протокол Signal (включает Double Ratchet, X3DH, и Prekeys); Криптография (включает Curve25519, AES-GCM, HMAC-SHA256).

Достоинства:

- открытый исходный код – проект активно поддерживается сообществом;
- полноценная реализация сквозного шифрования по умолчанию;
- минимальный сбор метаданных;
- поддержка современных стандартов безопасности.

Недостатки:

- привязан к номеру телефона нет возможности узнать ассортимент заранее;
- возможны проблемы с доступом в странах с ограничениями;
- нет облачного хранения чатов (для повышения приватности).

1.2.2 Telegram (секретные чаты)

Telegram дает возможность использовать стандартные и секретные чаты. Последние используют сквозное шифрование, а также функции самоуничтожения сообщений. На рисунке 2 отображен процесс создания секретного чата в Telegram.

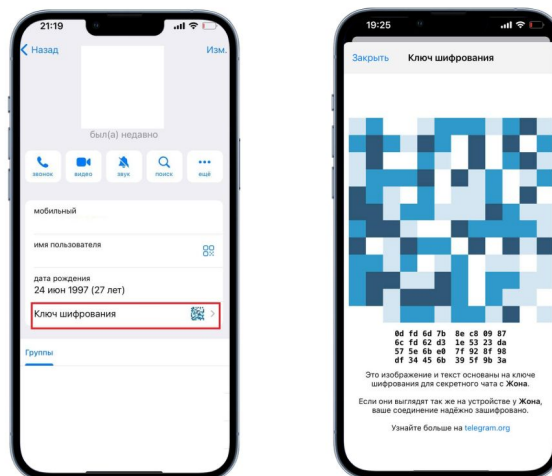


Рисунок 2 – секретный чат в Telegram

Универсальный инструмент с возможностями повышенной безопасности. Используется как в личных, так и в рабочих целях. Используемые технологии и алгоритмы: MTProto (собственный протокол Telegram); секретные чаты используют клиент-клиентное шифрование.

Достоинства:

- высокая скорость работы, хорошая оптимизация;
- широкая платформа – от смартфонов до десктопов и веба;
- гибкость интерфейса и API.

Недостатки:

- сквозное шифрование только в режиме "секретного чата", не используется по умолчанию;
- исходный код клиентской части открыт, серверной – закрыт;
- возможность хранения переписки в облаке снижает приватность.

1.2.3 Threema

Threema обеспечивает шифрование всех видов связи — текстовых сообщений, звонков, файлов. Не требует номера телефона или e-mail. Используемые технологии и алгоритмы: E2E-шифрование с использованием NaCl-библиотеки; протокол собственного дизайна на основе проверенных криптографических примитивов. На рисунке 3 показана главная страница мессенджера.

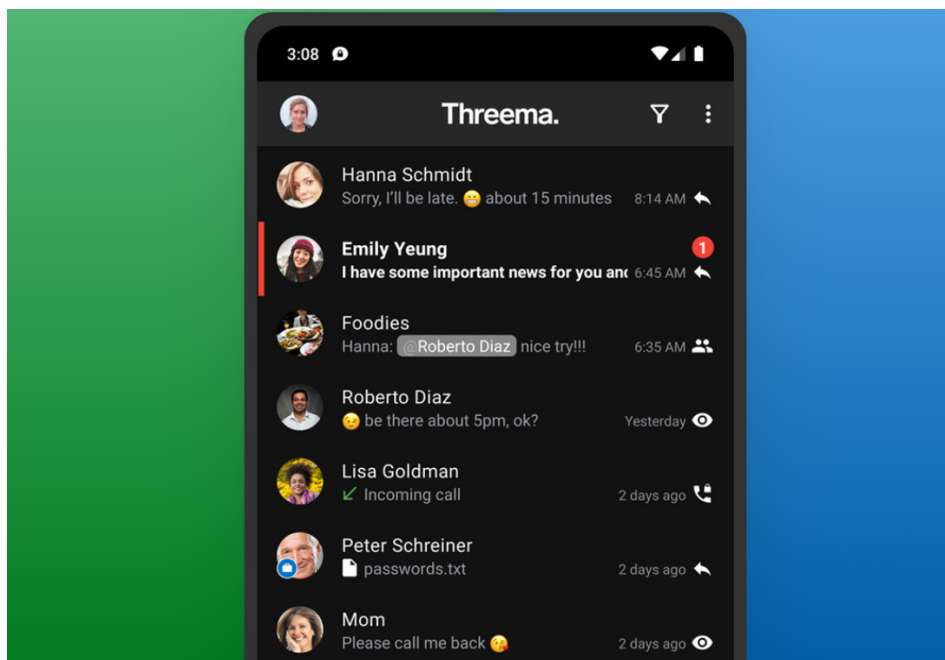


Рисунок 3 – Главная страница Threema

Мессенджер подходит для пользователей, желающих сохранить полную анонимность и не использовать номера телефонов.

Достоинства:

- полная анонимность пользователя (Threema ID вместо номера телефона);
- высокая безопасность и конфиденциальность;
- не требует привязки к облачным хранилищам.

Недостатки:

- платное приложение;
- относительно низкая популярность за пределами Европы;

– ограниченная открытость кода.

1.2.4 WhatsApp

WhatsApp - широко используемый мессенджер с поддержкой голосовой и видеосвязи, групп, пересылки медиа и документов. На рисунке 1 представлено окно чата WhatsApp.



Рисунок 4 – Главная страница WhatsApp

Мессенджер основан на протоколе Signal. Сквозное шифрование по умолчанию с 2016 года.

Достоинства:

- огромная база пользователей;
- простая авторизация по номеру телефона;
- хорошо работает на всех типах устройств.

Недостатки:

- закрытый исходный код;
- принадлежит компании Meta (есть опасения по поводу конфиденциальности);
- потенциальный сбор и анализ метаданных.

1.2.5 Matrix

Matrix использует открытый протокол децентрализованных коммуникаций. Клиент Element поддерживает чаты, видеозвонки, обмен файлами и интеграции с другими сервисами. На рисунке 5 показана главная страница мессенджера.

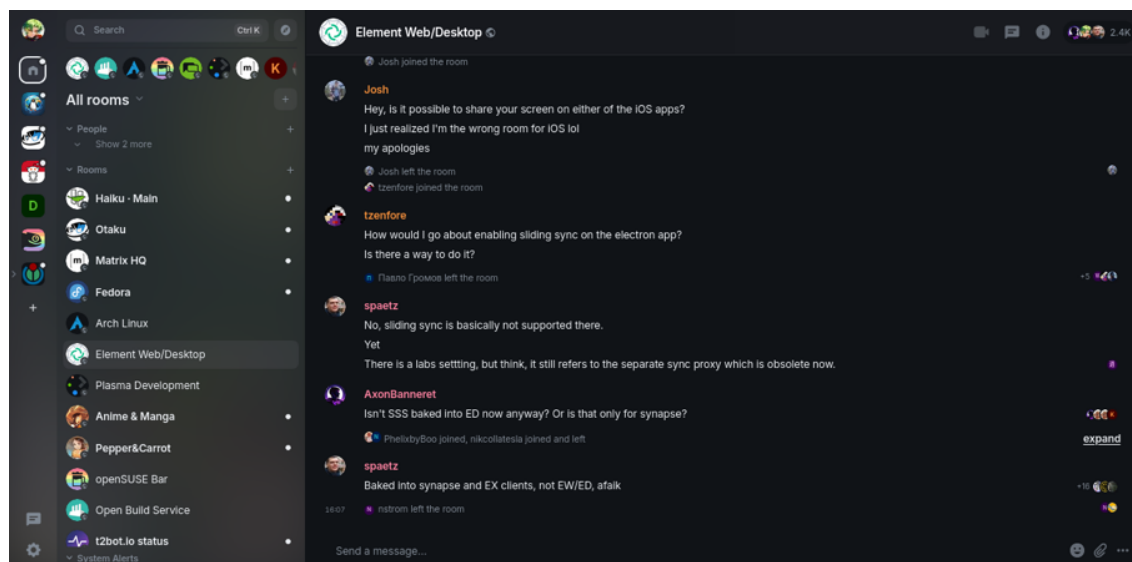


Рисунок 5 – Главная страница Matrix

Подходит для технически подкованных пользователей и организаций, желающих контролировать всю инфраструктуру связи. Использует технологии: протокол Matrix + модуль шифрования Olm/Megolm и E2E-шифрование для чатов.

Достоинства:

- возможность развёртывания собственного сервера;
- гибкость, масштабируемость;
- полностью открытый код.

Недостатки:

- сложность настройки для обычного пользователя;
- некоторые компоненты всё ещё нестабильны.

1.2.6 Retroshare

Retroshare - децентрализованная платформа для защищённой P2P-коммуникации. Предоставляет обмен сообщениями, форумами, файлами и каналами новостей. На рисунке 6 показана главная страница мессенджера.

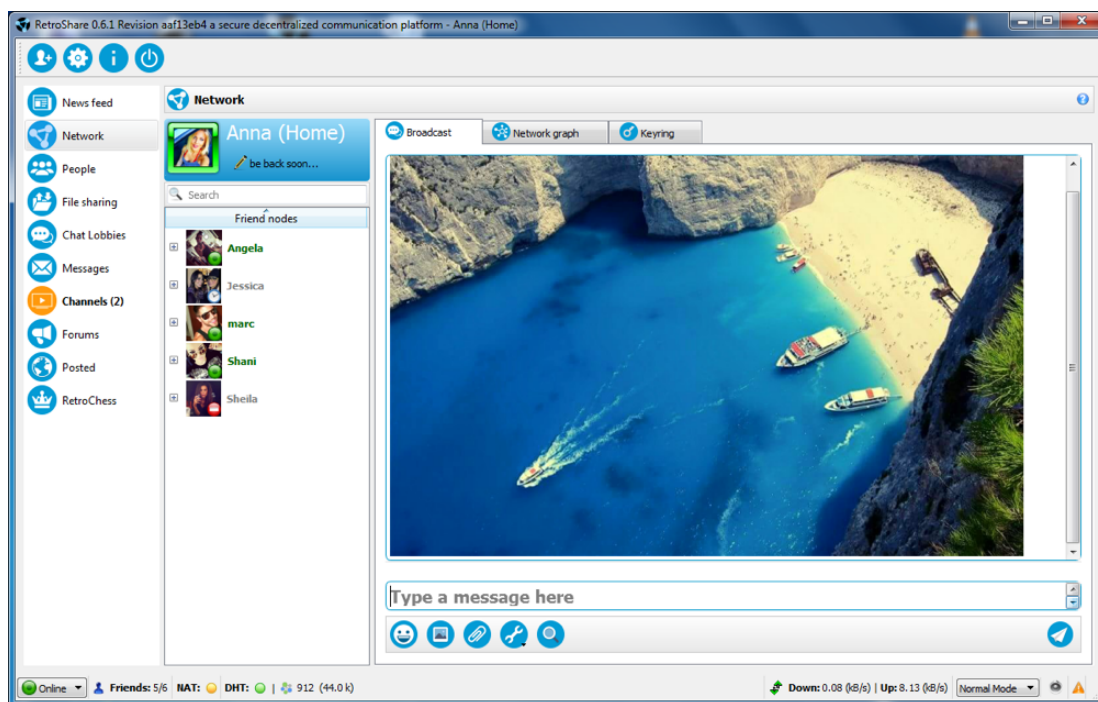


Рисунок 6 – Главная страница Retroshare

Платформа хороша для пользователей, которым критична анонимность и независимость от централизованных сервисов. Использует OpenSSL, асимметричное и симметричное шифрование, а все соединения шифруются end-to-end.

Достоинства:

- полная децентрализация;
- нет центральных серверов или единой точки отказа;
- высокий уровень анонимности.

Недостатки:

- сложность освоения;
- интерфейс не интуитивен;

– требуется техническая подготовка пользователя.

1.2.7 Вывод

Анализ существующих решений позволяет выделить важные характеристики для реализации собственного мессенджера с шифрованием: обязательная поддержка сквозного шифрования по умолчанию; открытость архитектуры; минимизация сбора метаданных; удобство пользовательского интерфейса и доступность на разных платформах; возможность использования без привязки к номеру телефона или централизованному облаку. На таблице 1 представлено сравнение характеристик аналогов.

Таблица 1 – Достоинства и недостатки аналогов

	Signal	Telegram	Threema	WhatsApp	Matrix	Retroshare
Сквозное шифрование	+	+	+	+	+	-
Открытый код	+	+	+	-	+	-

	Signal	Telegram	Threema	WhatsApp	Matrix	Retroshare
Привязка к номеру	+	+	-	+	-	+
Хранение данных	-	+	-	+	+	-
Платность	-	-	+	-	-	-

Собственный проект может занять нишу между удобством и полной приватностью, предложив упрощённую альтернативу Threema с открытым кодом и расширяемой архитектурой, особенно актуальной для обучения и интеграции в корпоративную инфраструктуру.

Таким образом, актуальной задачей является создание надёжного и безопасного мессенджера с поддержкой шифрования, обеспечивающего конфиденциальную передачу данных, удобный пользовательский интерфейс и простоту интеграции. Реализация такого решения позволяет удовлетворить потребность в защищённой коммуникации как для личного, так и для профессионального использования.

1.3 Постановка задачи

Целью данного проекта является разработка мессенджера с функцией шифрования сообщений, обеспечивающего безопасную передачу информации между пользователями через сервер. Особенностью системы является то, что

сервер выступает только как ретранслятор зашифрованных данных, не имея доступа к содержимому сообщений.

Пользователь должен иметь возможность подключаться к серверу, указывая имя. Клиент должен получать список всех подключённых пользователей, обмениваться сообщениями, которые шифруются перед отправкой. Первый подключившийся пользователь становится ключевым хранителем, отвечающим за генерацию сессионного ключа. Сервер принимает сообщения от клиентов и пересылает их остальным, не расшифровывая. Клиенты обмениваются публичными ключами через сервер для безопасной передачи сессионного ключа. Сервер ведёт лог сообщений и отображает его в графическом интерфейсе, отображает текущих онлайн-пользователей и обновляет список при подключении или отключении. Администратор сервера может сохранить или загрузить историю сообщений. Шифрование сообщений реализовано с использованием симметричных алгоритмов (например, AES), а обмен ключами осуществляется по открытому каналу с использованием асимметричного шифрования (например, RSA).

Программа должна быть кроссплатформенной и функционировать под Windows и Linux. Интерфейс сервера должен быть выполнен на Java Swing, быть простым и удобным. Все соединения клиентов с сервером обрабатываются в отдельных потоках, чтобы избежать блокировок. При потере соединения клиент должен быть корректно удалён из списка активных пользователей. Сервер не хранит сессионные ключи и открытые ключи после завершения сессии, а также сообщения в логах сервера не должны содержать расшифрованного текста (в целях безопасности). Программа должна быть устойчива к некорректному вводу и исключениям на сетевом уровне.

Таким образом, реализация мессенджера с шифрованием позволит обеспечить защищённую передачу сообщений между пользователями, повысить уровень конфиденциальности общения, а также улучшить пользовательский опыт за счёт простого и понятного интерфейса и стабильной работы системы.

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						18
Изм.	Лист	№ докум.	Подпись	Дата		

1.4 Требования к аппаратуре и программному обеспечению

1.4.1 Клиентская часть

Для полноценной работы клиентской части необходимы следующие минимальные требования к устройству:

- операционная система – Windows 7+, Linux, MacOS;
- Java Runtime Environment (JRE) 8 или выше;
- объём оперативной памяти от 512 МБ;
- стабильное подключение к интернету.

1.4.2 Серверная часть

Для размещения серверной части приложения, реализованной на Java с использованием Swing, предъявляются следующие минимальные требования к серверу:

- операционная система – Windows 7+, Linux, MacOS;
- Java Development Kit (JDK) 8 или выше;
- объём оперативной памяти от 1 ГБ (рекомендуется 4 ГБ и выше);
- наличие свободного порта для подключения;
- сеть – постоянное подключение к интернету с пропускной способностью от 10 Мбит/с.

2 Архитектура приложения

Разрабатываемое программное обеспечение представляет собой клиент-серверное приложение, реализующее функциональность обмена сообщениями с использованием асимметричного и симметричного шифрования. Архитектура разделена на три основные логические компонента: серверная часть, клиентская часть и общий модуль сообщений, что изображено на диаграмме компонентов на рисунке 6.

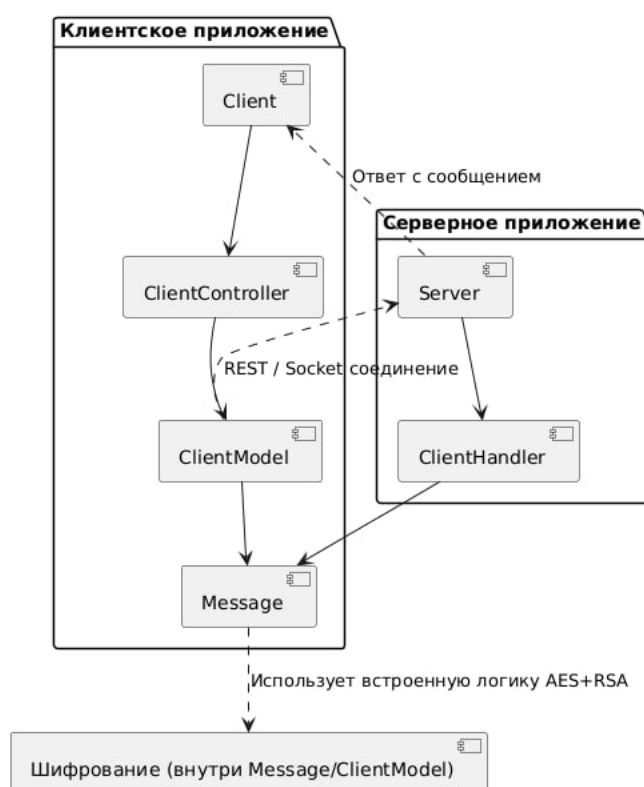


Рисунок 6 – UML-диаграмма компонентов

Серверная часть:

- класс Server отвечает за запуск сервера;
- ServerModel управляет хранилищем сообщений, списком подключённых пользователей, сохранением и загрузкой истории чата;
- ServerView реализует графический интерфейс с возможностью визуализации сообщений и управления сервером;

- ServerController соединяет модель и представление, а также обрабатывает действия пользователя (например, запуск сервера, сохранение чата);
- ClientHandler обрабатывает подключение каждого клиента в отдельном потоке, управляет пересылкой сообщений, списками пользователей и логикой распределения ключей шифрования.

Клиентская часть:

- Client управляет основным соединением с сервером и логикой запуска клиента;
- ClientModel содержит логику обработки сообщений, шифрования и дешифрования;
- ClientView отображает графический интерфейс пользователя;
- ClientController обрабатывает действия пользователя, отправку сообщений, подключение к серверу и взаимодействие с моделью.

Общий модуль сообщений:

- класс Message используется для передачи сообщений между клиентами и сервером;
- поддерживает флаги (например, является ли сообщение сеансовым ключом, обычным сообщением или системным уведомлением).

Передача данных между клиентом и сервером осуществляется через сериализованные Java-объекты. Для защиты конфиденциальности реализован механизм:

- генерации асимметричных ключей (RSA) на стороне клиентов;
- обмена сеансовыми симметричными ключами (AES) через открытый ключ;
- шифрования текстовых сообщений на стороне клиента перед отправкой.

Таким образом, сервер не имеет доступа к содержимому сообщений и служит лишь ретранслятором зашифрованных данных.

На рисунке 6 слева представлена клиентская часть, состоящая из следующих компонентов:

- Client – точка входа клиента, инициирует работу приложения;
- ClientController – связывает модель и представление, управляет логикой пользовательского взаимодействия;
- ClientModel – содержит бизнес-логику клиента, включая формирование и отправку сообщений;
- ClientView – отвечает за отображение интерфейса пользователю;

- Crypto (RSA/AES) – реализует шифрование и дешифрование сообщений. Модель клиента взаимодействует с этим модулем для обеспечения безопасности данных;
- Message – объект сообщения, используется для представления структуры передаваемых данных.

Клиент устанавливает TCP-соединение с сервером, передаёт зашифрованные сообщения, а также обменивается данными через ClientHandler на стороне сервера. Справа расположена серверная часть, включающая:

- Server – точка запуска серверного приложения;
- ServerController – организует работу между моделью и представлением сервера;
- ServerModel – обрабатывает логику работы с клиентами и сообщениями;
- ServerModel – обрабатывает логику работы с клиентами и сообщениями;
- ClientHandler – поток или модуль, выделенный на каждого подключённого клиента, обрабатывает индивидуальные соединения;
- Message – также используется на серверной стороне для приёма и обработки сообщений.

Между модулями ClientHandler и Client происходит двусторонний обмен данными, включая приём и отправку сообщений. ServerModel передаёт данные в ServerView для отображения текущего состояния.

В центре схемы выделен общий компонент Message, используемый как клиентской, так и серверной сторонами для обмена данными в унифицированном формате.

Эта архитектура обеспечивает модульность, масштабируемость и безопасность приложения за счёт чёткого разделения ответственности между компонентами и использования криптографических методов защиты.

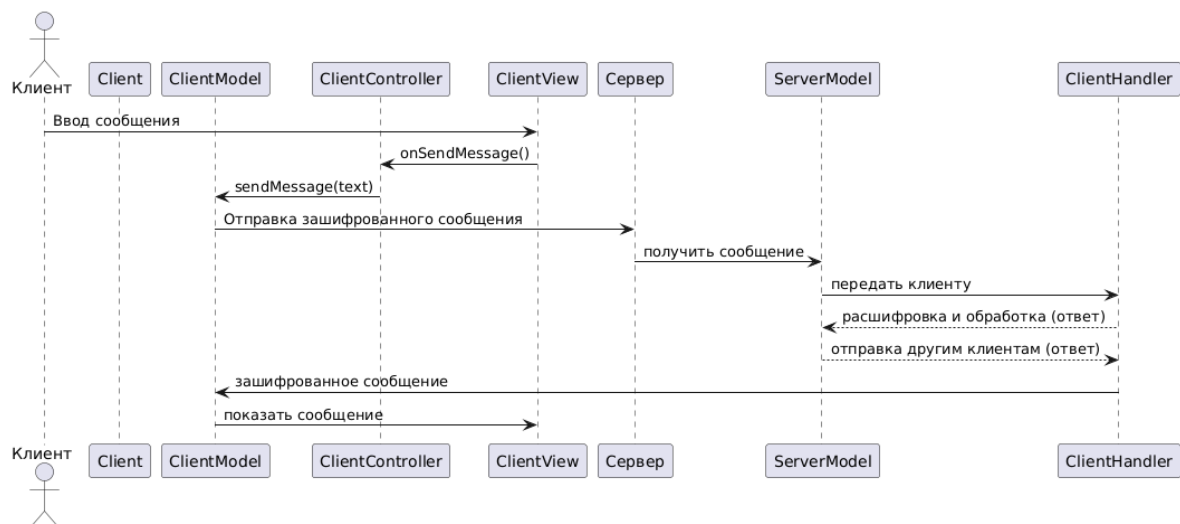


Рисунок 7 – UML-диаграмма последовательности

Клиентское приложение работает под управлением операционных систем Windows или Linux и реализовано на языке Java. Серверное приложение, также написанное на Java, размещено на сервере с операционной системой Linux и использует Java ServerSocket для обработки входящих подключений. На рисунке 7 показана диаграмма последовательности взаимодействия клиентов.

Взаимодействие между клиентом и сервером осуществляется по протоколу TCP/IP с передачей зашифрованных сообщений, что обеспечивает безопасность передаваемых данных. Для сохранения истории сообщений и событий системы используется запись в файл, что исключает необходимость применения СУБД и упрощает хранение и доступ к данным. Такой подход позволяет сохранять лог и обеспечивать его доступность для последующего анализа. На рисунке 8 представлена общая диаграмма развёртывания.

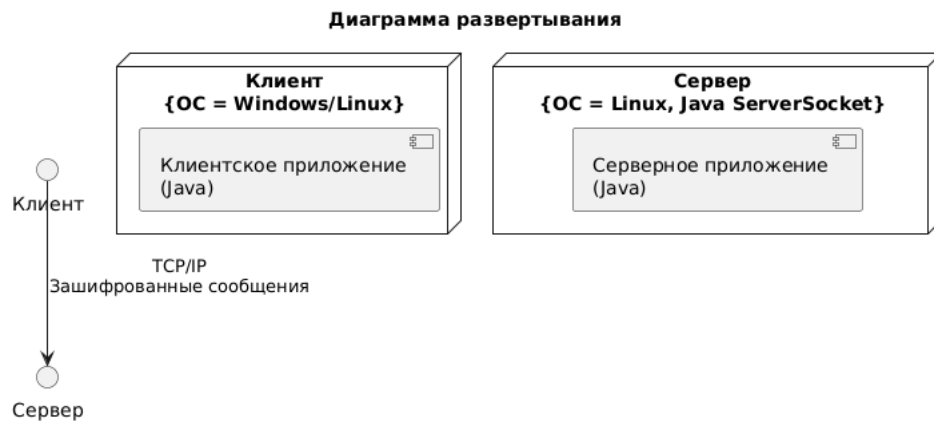


Рисунок 8 – UML-диаграмма развёртывания

Архитектура защищённого мессенджера, разделёна на три логических пакета: client (клиентская часть), server (серверная часть) и message (структура передаваемых сообщений). Она отражает структуру приложения, связи между основными модулями и их ключевые методы и поля.

В клиентской части приложения реализована модель MVC (Model-View-Controller):

- Client – основной класс клиента, устанавливающий соединение с сервером. Он содержит сокет, потоки ввода/вывода, имя пользователя и симметричный ключ сессии. Именно этот класс инициирует клиентскую часть приложения;
- ClientModel – модель, хранящая список сообщений и отвечающая за отправку и приём сообщений. Связана с потоком для передачи данных;
- ClientView – интерфейс пользователя. Содержит текстовое поле и область чата. Обновляет отображение сообщений;
- ClientController – контроллер, связывающий модель и представление. Инициализирует обработку событий и управляет взаимодействием между ними.

Взаимодействие связей между классами:

- Client создаёт и использует ClientModel, ClientView и ClientController;

- ClientController напрямую взаимодействует как с моделью (ClientModel), так и с представлением (ClientView).

Серверная часть также логически структурирована:

- Server — точка входа сервера. Запускает основной процесс;
- ServerController – координирует работу серверной модели и интерфейса. Отвечает за инициализацию;
- ServerModel – логика работы сервера: список онлайн-пользователей, приём/сохранение сообщений, работа с сокетами;
- ServerView – интерфейс администратора сервера, отображает список пользователей и историю сообщений;
- ClientHandler – поток, обрабатывающий отдельного клиента. Содержит потоки данных, ключ сессии и имя пользователя. Реализует логику чтения, расшифровки и пересылки сообщений.

Взаимодействие связей между классами:

- ServerController управляет ServerModel и ServerView;
- ServerModel взаимодействует с ClientHandler для обслуживания пользователей;
- ServerView получает данные от модели для отображения;
- ClientHandler тесно связан с ServerModel и объектами Message.

Message представляет структуру одного сообщения. Включает поля для текста, признаков шифрования и типа сообщения (например, ключ сессии или объявление). Используется как на стороне клиента (ClientModel, ClientHandler), так и на сервере (ServerModel, ClientHandler), что отражает универсальность его назначения. Диаграмма классов и связей представлена на рисунке 9.

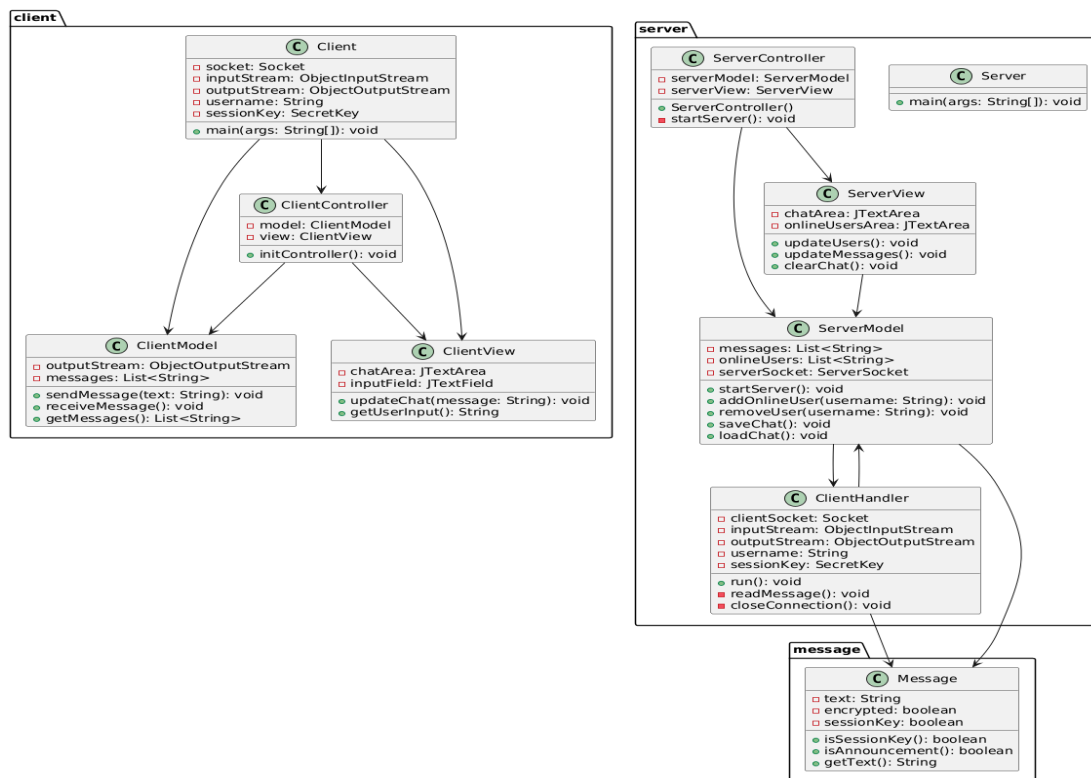


Рисунок 9 – UML-диаграмма классов

3 Выбор инструментов разработки

3.1 Выбор языка программирования

Для реализации мессенджера с шифрованием был выбран язык программирования Java. Этот выбор обусловлен рядом причин, среди которых можно выделить следующие:

- Java компилируется в байт-код, который исполняется на виртуальной машине Java (JVM). Это позволяет запускать приложение на любых операционных системах (Windows, Linux, macOS), что особенно важно для клиент-серверного взаимодействия;
- Java предоставляет богатый набор встроенных классов для работы с сетью (например, Socket, ServerSocket, ObjectInputStream, ObjectOutputStream), что упрощает реализацию надёжного TCP-соединения между клиентом и сервером;
- возможность создания многопоточных приложений средствами Java (Thread, Runnable, ExecutorService) делает этот язык отличным выбором для реализации параллельной обработки подключений на стороне сервера – в частности, через класс ClientHandler;
- проект построен с использованием объектно-ориентированных принципов: инкапсуляции, наследования и полиморфизма. Java полностью поддерживает ООП, что позволило разделить клиентскую и серверную логику по отдельным классам (например, ClientModel, ServerModel, Message и др.);
- встроенная криптографическая библиотека Java (javax.crypto, java.security) предоставляет все необходимые инструменты для реализации шифрования, включая генерацию ключей, шифрование и дешифрование, работу с сессионными ключами;

- Java имеет одно из крупнейших сообществ и большое количество документации, что значительно ускоряет разработку и поиск решений.

Таким образом, Java предоставляет мощные средства для работы с файлами и потоками ввода-вывода, что делает данный подход удобным и эффективным. Запись истории в файл обеспечивает сохранность данных между сессиями и помогает отслеживать коммуникацию пользователей, что важно для аудита и диагностики работы приложения. Такой подход также способствует снижению сложности архитектуры и упрощает сопровождение проекта.

3.2 Выбор среды разработки

В качестве основной среды разработки была выбрана IntelliJ IDEA, которая предоставляется компанией JetBrains. Выбор именно этой IDE обусловлен следующими факторами:

- Java имеет одно из крупнейших сообществ и большое количество документации, что значительно ускоряет разработку и поиск решений;
- IntelliJ IDEA изначально спроектирована для разработки на Java. Она обеспечивает: интеллектуальную подсветку синтаксиса и автодополнение, быстрый переход по классам и методам, расширенные функции рефакторинга, поддержку Maven и Gradle, а также других систем сборки;
- так как клиентская и серверная части проекта имеют графический интерфейс, построенный на Swing, важно было иметь удобный визуальный редактор форм (GUI Designer), который позволяет создавать и настраивать компоненты интерфейса в режиме реального времени. IntelliJ IDEA предоставляет встроенный редактор форм *.form, который значительно упрощает работу с визуальной частью;
- встроенные инструменты отладки позволяют пошагово отслеживать выполнение программы, устанавливать точки останова, просматривать

значения переменных, что особенно полезно при отладке клиент-серверных взаимодействий и шифрования;

- поддержка Git и других систем управления версиями позволяет организовать удобную командную работу, версионирование кода, а также создавать резервные копии проекта;
- IntelliJ IDEA зарекомендовала себя как стабильная и надёжная IDE, подходящая для проектов любой сложности;
- интеграция с плагинами для построения UML-диаграмм (например, PlantUML Integration, Diagram support) позволяет быстро визуализировать архитектуру приложения, варианты использования и схемы взаимодействий, что особенно важно при документировании дипломного проекта.

Выбор Java в качестве основного языка разработки и IntelliJ IDEA как среды разработки позволил реализовать кроссплатформенное, безопасное и масштабируемое приложение для обмена сообщениями. Эти инструменты обеспечили высокую производительность, удобство в разработке, поддержку работы с графическим интерфейсом и криптографическими модулями, а также способствовали созданию гибкой архитектуры.

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

4 Реализация приложения

4.1 Описание функционала

Разрабатываемое приложение представляет собой защищённый мессенджер, реализующий клиент-серверную архитектуру. Основной функционал направлен на безопасную передачу сообщений между пользователями. Все передаваемые данные шифруются с использованием гибридного подхода – симметричного и асимметричного шифрования.

Функциональные возможности системы включают в себя регистрацию и идентификацию пользователя по имени (username) при подключении к серверу.

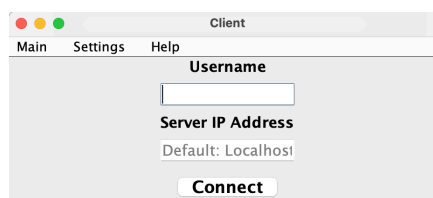


Рисунок 10 – Страница авторизации

Находясь на главной странице, изображенной на рисунке 10, пользователь может зарегистрироваться, ему необходимо указать:

- имя;
- IP адрес сервера.

После регистрации и подключения к нужному серверу клиенты могут обмениваться сообщениями, что показано на рисунке 11.

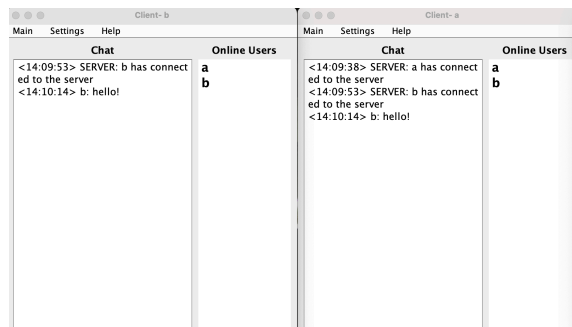


Рисунок 11 – Окна клиентов для обмена сообщениями

Шифрование сообщений происходит с использованием сессионных ключей и асимметричной криптографии. RSA используется для обмена ключами, а AES для шифрования текста. На рисунке 12 показано окно сервера и то, в каком виде на сервер поступают сообщения от клиентов.

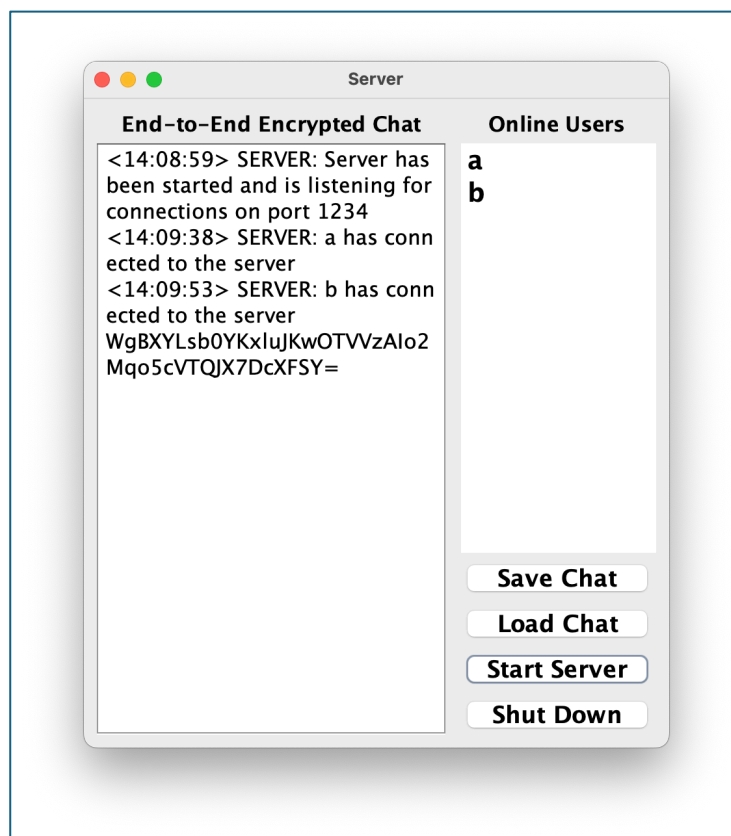


Рисунок 12 – Окно сервера и отправленное клиентом зашифрованное сообщение

Обмен сессионными ключами между клиентами происходит через сервер для установления защищённого канала связи. Чат отображается в графическом пользовательском интерфейсе (GUI), реализованном на Java Swing. На клиентской и серверной части есть отображение списка онлайн-пользователей на серверной стороне, показанное на рисунке 13.



Рисунок 13 – Окно сервера и отправленное клиентом зашифрованное сообщение

В проекте также имеется возможность сохранения истории чата. При помощи кнопок, представленных на рисунке 14 пользователь может загрузить старый чат или сохранить текущий.



Рисунок 14 – Окно сервера для сохранения текущего или загрузки старых чатов

4.2 Описание работы приложения

Разработанное приложение представляет собой мессенджер с функцией шифрования сообщений, реализованный по клиент-серверной архитектуре. Работа приложения строится на обмене данными между клиентами через центральный сервер, при этом обеспечивается безопасность передаваемой информации за счёт применения криптографических методов.

На клиентской стороне пользователь запускает приложение, указывает имя (username) и подключается к серверу, что показано на рисунке 15. После установления соединения клиент создаёт сессионный ключ (симметричный ключ шифрования AES), который используется для шифрования текстовых сообщений. Этот ключ передаётся серверу в зашифрованном виде с помощью алгоритма RSA (асимметричное шифрование), что позволяет избежать перехвата ключа третьими лицами.

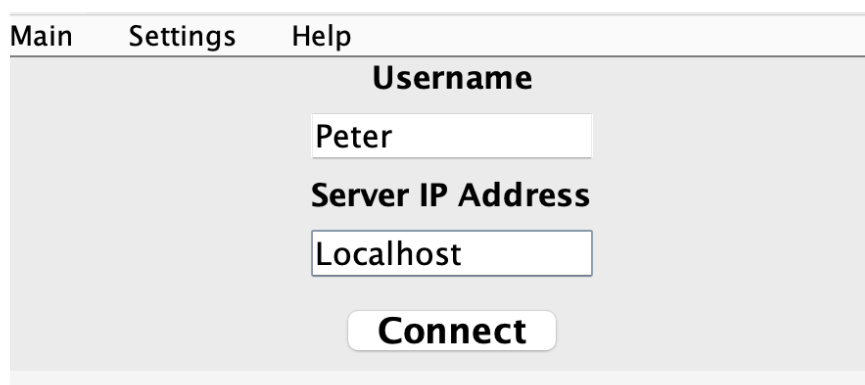


Рисунок 15 – Авторизация пользователя

Пользователь вводит сообщение в графическом интерфейсе. Сообщение шифруется сессионным ключом и отправляется через сокет-соединение на сервер. Отправка сообщения показана на рисунке 16.

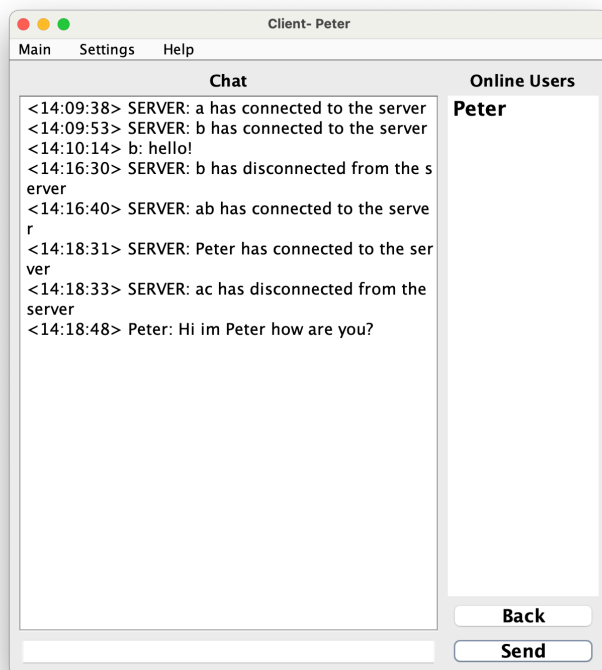


Рисунок 16 – Отправка сообщения пользователем

Шифрованный текст поступает на сервер, где обрабатывается серверной моделью. Затем сообщение доставляется остальным клиентам через поток ClientHandler. Зашифрованное сообщение показано на рисунке 17.

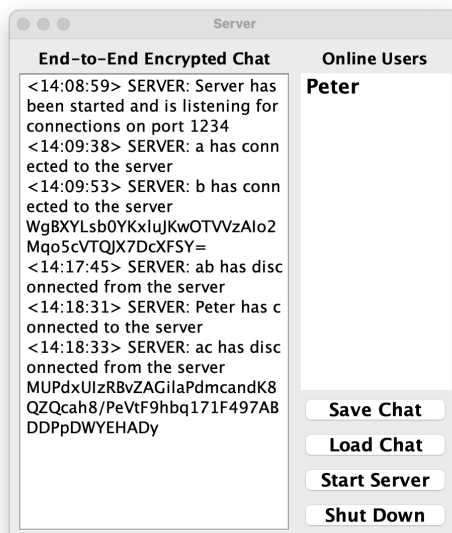


Рисунок 17 – Отображение зашифрованного сообщения на сервере

На серверной стороне приложение принимает подключения от клиентов, создаёт для каждого отдельный поток и обрабатывает передаваемые сообщения. Сервер ведёт учёт подключённых пользователей, отображает историю сообщений и сохраняет чат в файл по запросу. Полученные сообщения доставляются другим клиентам, где они расшифровываются с использованием локального сессионного ключа и отображаются в пользовательском интерфейсе.

Таким образом, приложение обеспечивает:

- безопасную передачу сообщений;
- управление подключениями клиентов;
- сохранение истории переписки;
- простое и понятное взаимодействие с пользователем через графический интерфейс.

Система реализована на языке Java, с использованием стандартных библиотек для работы с сетью (Socket), пользовательским интерфейсом (Swing), сериализацией объектов и криптографией (javax.crypto, java.security).

4.3 Описание интерфейса приложения

Пользовательский интерфейс разработанного мессенджера выполнен с использованием библиотеки Swing, входящей в стандартный набор Java. Интерфейс минималистичен, интуитивно понятен и предназначен для работы с основным функционалом приложения: отправкой и приёмом зашифрованных сообщений, отображением чата и подключённых пользователей. После запуска клиентского приложения пользователь видит главное окно чата, состоящее из следующих элементов:

- область отображения сообщений (chatArea) – основная текстовая панель, представленная на рисунке 18, на которой в хронологическом порядке отображаются входящие и исходящие сообщения. Каждое сообщение сопровождается именем отправителя;

```

<14:09:38> SERVER: a has connected to the server
<14:09:53> SERVER: b has connected to the server
<14:10:14> b: hello!
<14:16:30> SERVER: b has disconnected from the s
erver
<14:16:40> SERVER: ab has connected to the serve
r
<14:18:31> SERVER: Peter has connected to the ser
ver
<14:18:33> SERVER: ac has disconnected from the
server
<14:18:48> Peter: Hi im Peter how are you?

```

Рисунок 18 – Область отображения сообщений

– поле ввода сообщения на рисунке 19 – текстовое поле, куда пользователь вводит новое сообщение;

– кнопка отправки сообщения на рисунке 19 (в проекте отправка инициируется нажатием Enter или через соответствующий обработчик) – при активации сообщение шифруется и отправляется на сервер;

The image shows a horizontal UI element. On the left is a rectangular text input field with a light gray border. To its right is a rounded rectangular button with a blue border and the word "Send" in black text.

Рисунок 19 – Кнопка отправки сообщений и поле ввода

– область списка подключённых пользователей (onlineUsersArea) (на серверной стороне) – отображает список активных пользователей в реальном времени. Отображение списка подключенных пользователей показано на рисунке 20.

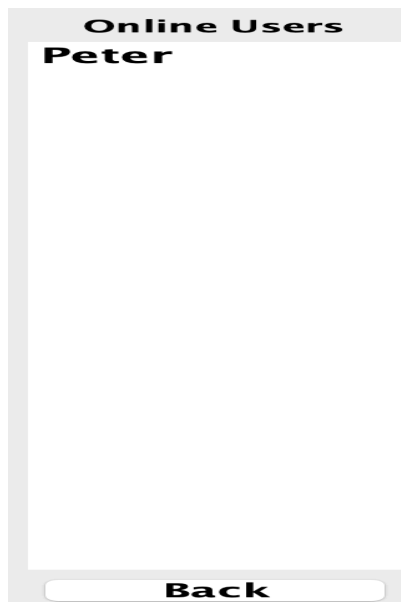


Рисунок 20 – область списка подключенных пользователей

Интерфейс реализован в классах ClientView на клиенте и ServerView на сервере. Обновление интерфейса происходит автоматически при получении нового сообщения или изменении состава участников. Компоненты интерфейса обновляются с использованием методов updateChat, updateMessages, updateUsers и других вспомогательных функций.

Главная цель интерфейса – обеспечить максимально простую и быструю коммуникацию между пользователями, при этом скрывая от них всю сложность криптографической обработки сообщений. Пользователь не взаимодействует с механизмами шифрования напрямую – они полностью автоматизированы и интегрированы в работу приложения.

Таким образом, интерфейс реализует базовые требования к мессенджеру:

- поле ввода сообщения (inputField) – текстовое поле, куда пользователь вводит новое сообщение;
- быстрый доступ к переписке;
- мгновенная отправка сообщений;
- визуальное отображение истории чата;
- отображение активных участников.

При необходимости интерфейс можно расширить дополнительными функциями, такими как: отправка файлов, история переписки, выбор алгоритма шифрования, смена темы оформления.

4.4 Тестирование приложения

После завершения этапа разработки было проведено функциональное и интеграционное тестирование клиентской и серверной частей приложения. Целью тестирования являлась проверка корректности реализации основных функций, стабильности передачи зашифрованных сообщений и надёжности соединения между клиентом и сервером.

Функциональное тестирование проводилось вручную и охватывало следующие сценарии:

- успешное подключение клиента к серверу;
- регистрация и аутентификация пользователя;
- передача текстовых сообщений между двумя и более клиентами;
- шифрование и расшифровка сообщений при передаче;
- обработка отключения клиента во время сессии;
- отображение истории чата и списка подключённых пользователей на стороне сервера.

Во всех тестируемых случаях приложение показало стабильную работу без критических ошибок.

Особое внимание было уделено взаимодействию модулей шифрования (RSA/AES) с остальной логикой приложения. Тестировались следующие случаи:

- генерация сессионного симметричного ключа AES на клиенте;
- успешная передача ключа на сервер в зашифрованном виде (RSA);
- использование сессионного ключа для симметричного шифрования сообщений;
- корректная расшифровка сообщений на принимающей стороне.

Все тесты были проведены с использованием заранее подготовленных тестовых данных, включая текстовые сообщения разной длины, специальные символы и символы Unicode. В рамках ограниченного тестирования производительности было запущено до 10 одновременных клиентов. Приложение успешно справилось с многопоточным обслуживанием без потери данных и ошибок, что подтверждает корректную работу ClientHandler на серверной стороне.

Проведённое тестирование показало, что приложение работает стабильно в рамках предполагаемой нагрузки и реализует заявленный функционал. Проблем с утечкой данных, нарушением целостности сообщений или падением соединения выявлено не было.

					ПГУ1.090301.05.001 ПЗ	Лист
						39
Изм.	Лист	№ докум.	Подпись	Дата		

5 Безопасность жизнедеятельности

Использование персонального компьютера и смартфона совсем не так безопасно, как можно подумать. Это убедительно подтверждается информацией, содержащейся в стандарте ГОСТ 12.0.003-2015, описывающем основные вредные и опасные факторы такого трудового процесса. Для обеспечения безопасной работы пользователей и разработчиков были разработаны специальные правила и нормативы. Несоблюдение данных правил может привести к негативным последствиям для физического состояния пользователя и его имущества.

5.1 Правила и техника безопасности при работе с ПК

Согласно требованиям стандарта ГОСТ 12.0.003-2015, при работе с персональным компьютером могут воздействовать следующие вредные факторы:

- высокая температура, исходящая от отдельных компонентов компьютерного оборудования и создающая повышенный температурный фон в помещении;
- высокая степень однообразия выполняемой работы;
- значительная зрительная нагрузка на работника;
- вероятность поражения статическим электричеством при случайном прикосновении к отдельным элементам техники;
- недостаточный уровень освещенности рабочего пространства;
- повышенные уровни электромагнитных и электрических полей, возникающих при работе оборудования;
- чрезмерная яркость и контрастность экрана, способные негативно влиять на зрение при длительном воздействии.

Для снижения вредного влияния необходимо соблюдать определённые меры и рекомендации.

Следует правильно организовать эргономику рабочего места:

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

– правильная посадка. Пользователь должен сидеть с прямой спиной, руки должны находиться на уровне клавиатуры. Стул должен регулироваться по высоте и иметь спинку, обеспечивающую возможность полностью выпрямить ноги и устойчиво поставить ступни на пол. Расстояние до экрана должно быть примерно на вытянутую руку, при этом экран желательно наклонить на 30 градусов для уменьшения нагрузки на спину, шею и руки;

– освещение. Рабочее место должно быть достаточно освещено, но без излишней яркости, чтобы избежать зрительного переутомления. Также важно исключить появление бликов на экране монитора.

В целях безопасности запрещается использовать компьютерную технику при наличии видимых повреждений. Следует регулярно проверять техническое состояние оборудования и не оставлять его включённым без присмотра на длительное время. Рабочие места рекомендуется размещать вдоль стен, чтобы задняя сторона монитора была обращена к стене — это способствует снижению уровня вредного электромагнитного излучения [5].

5.2 Правила и техника безопасности при работе со смартфоном

Так как разработанное приложение будет функционировать на телефоне, то следует также изучить возможное вредное влияние и все правила безопасности по обращению с ним.

Поскольку разработанное web-приложение может использоваться на мобильных устройствах, важно учитывать возможное вредное воздействие смартфонов и соблюдать основные правила безопасной эксплуатации

Не рекомендуется самостоятельно вскрывать, разбирать или изменять конструкцию устройства. Также следует избегать физических воздействий на смартфон — его нельзя ронять, сгибать, бросать, прокалывать, а также подвергать воздействию огня или влаги.

Использование мобильного телефона или зарядного устройства при наличии неисправностей запрещено. Чтобы избежать перегрева и короткого замыкания, следует применять только оригинальные зарядные устройства и кабели. Не оставляйте смартфон подключённым к зарядке под подушкой или в плохо проветриваемых местах — это может привести к перегреву устройства.

Мобильные телефоны постоянно излучают радиоволны, поэтому в целях безопасности не следует держать их близко к жизненно важным органам. Во время разговора рекомендуется периодически менять положение устройства относительно головы. Следует также учитывать, что на данный момент отсутствуют достоверные и всесторонние исследования, подтверждающие вред радиоизлучения [6].

Соблюдение этих рекомендаций способствует не только продлению срока службы мобильного устройства, но и сохранению здоровья пользователя, обеспечивая безопасное и комфортное использование смартфона в повседневной жизни.

					<i>ПГУ1.090301.05.001 ПЗ</i>	Лист
						42
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения выпускной квалификационной работы было разработано десктопное клиент-серверное приложение «Шифрованный мессенджер». Основной целью проекта являлось создание защищённого канала обмена сообщениями между пользователями с использованием механизмов шифрования, обеспечивающих конфиденциальность передаваемой информации.

Во время разработки были задействованы следующие технологии:

- Java – основной язык программирования, использованный для реализации как клиентской, так и серверной части приложения;
- Java Swing – библиотека для создания графического интерфейса пользователя (GUI) на стороне клиента;
- Socket API – для организации сетевого взаимодействия между клиентом и сервером;
- механизм сериализации Java-объектов – для передачи структурированных сообщений по сети;
- алгоритмы шифрования AES и RSA – используются для защиты сообщений и безопасного обмена ключами;
- модель клиент-серверной архитектуры – обеспечила разделение логики обработки данных и пользовательского интерфейса.

Разработанное приложение предоставляет пользователю возможность отправлять и получать зашифрованные сообщения, тем самым обеспечивая базовый уровень информационной безопасности. Шифрование сообщений реализовано на стороне клиента, что предотвращает возможность перехвата и расшифровки данных на сервере. Система поддерживает симметричное шифрование с использованием алгоритма AES, а безопасная передача ключей обеспечивается с помощью асимметричного алгоритма RSA.

Одним из ключевых преимуществ разработанного мессенджера является простота использования, а также базовая защита личной переписки,

реализованная без привлечения сторонних библиотек или сервисов. Программа имеет удобный интерфейс и может использоваться на компьютерах с операционной системой Windows/Linux.

В дальнейшем возможно расширение функциональности приложения: добавление групповых чатов, передачи файлов, истории сообщений, уведомлений, двухфакторной аутентификации, а также реализация более гибкой системы управления ключами и интеграция с облачными сервисами хранения. Эти усовершенствования сделают продукт более конкурентоспособным и востребованным для пользователей, заинтересованных в конфиденциальной коммуникации.

					ПГУ1.090301.05.001 ПЗ	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Nature.com: [Электронный ресурс]. URL: <https://www.nature.com/articles/s41598-024-71666-y> (дата обращения: 21.05.2025)
2. Itcollege.ee: [Электронный ресурс]. URL: https://enos.itcollege.ee/~edmund/materials/Telegram/A-practical-cryptanalysis-of-the-Telegram-messaging-protocol_master-thesis.pdf (дата обращения: 21.05.2025)
3. Mtpsym.github.io: [Электронный ресурс]. URL: <https://mtpsym.github.io> (дата обращения: 22.05.2025)
4. Usenix.org: [Электронный ресурс]. URL: <https://www.usenix.org/system/files/sec23fall-prepub-303-paterson.pdf> (дата обращения: 23.05.2025)
5. Ethz.ch: [Электронный ресурс]. URL: <https://ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/appliedcrypto/education/theses/threema.pdf> (дата обращения: 23.05.2025)
6. Whatsapp.com: [Электронный ресурс]. URL: <https://www.whatsapp.com/security> (дата обращения: 23.05.2025)
7. Arxiv.org: [Электронный ресурс]. URL: <https://arxiv.org/html/2408.12743v1> (дата обращения: 24.05.2025)
8. Nebuchadnezzar-megolm.github.io: [Электронный ресурс]. URL: <https://nebuchadnezzar-megolm.github.io/static/paper.pdf> (дата обращения: 24.05.2025)
9. Oracle.com: [Электронный ресурс]. URL: <https://docs.oracle.com/en/java/javase/17/> (дата обращения: 24.06.2025)
10. Baeldung.com: [Электронный ресурс]. URL: <https://www.baeldung.com/java-aes-encryption-decryption> (дата обращения: 26.05.2025)

11. Geeksforgeeks.org: [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/cryptography-and-its-types/> (дата обращения: 26.05.2025)
12. Wikipedia.org: [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Шифрование> (дата обращения: 08.06.2025)
13. Cryptobook.nakov.com: [Электронный ресурс]. URL: <https://cryptobook.nakov.com/> (дата обращения: 30.05.2025)
14. OWASP.org: [Электронный ресурс]. URL: <https://owasp.org/www-project-cryptographic-storage-cheat-sheet/> (дата обращения: 30.05.2025)

Приложение А

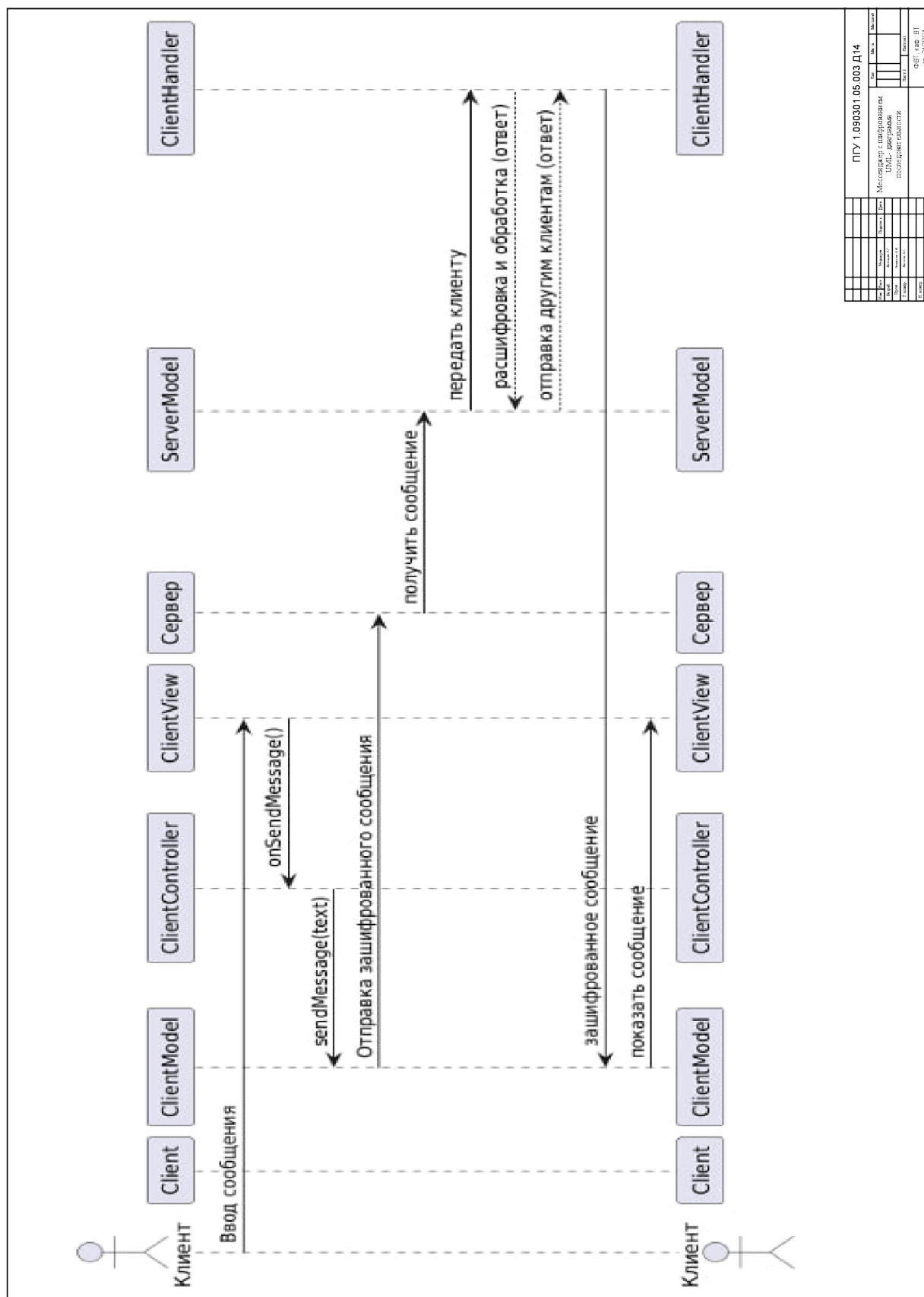
(обязательное)

Мессенджер с шифрованием

UML-диаграммы

					ПГУ1.090301.05.001 ПЗ	Лист
						47
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1-Диаграмма последовательности

[illegible]

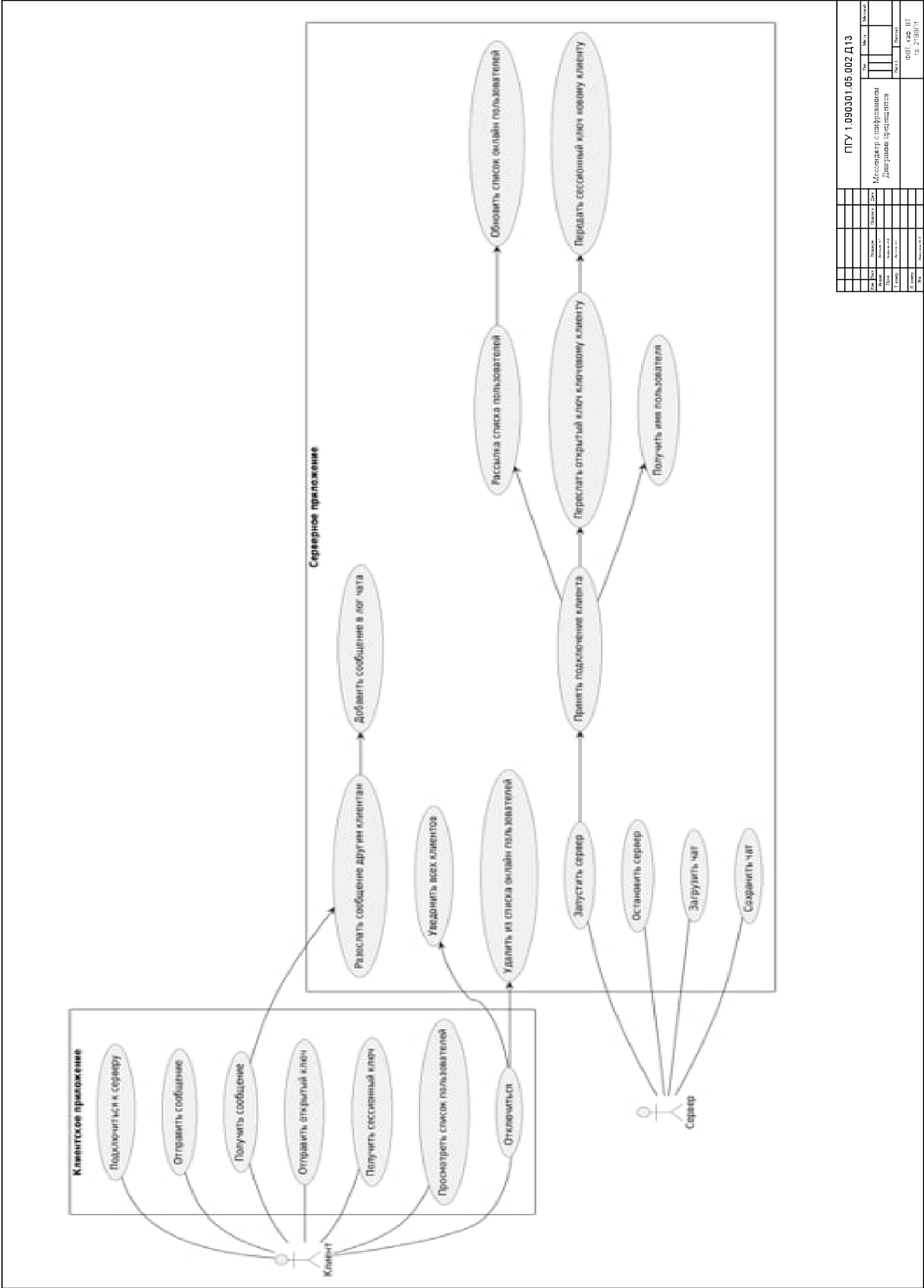
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.05.001 ПЗ

Лист

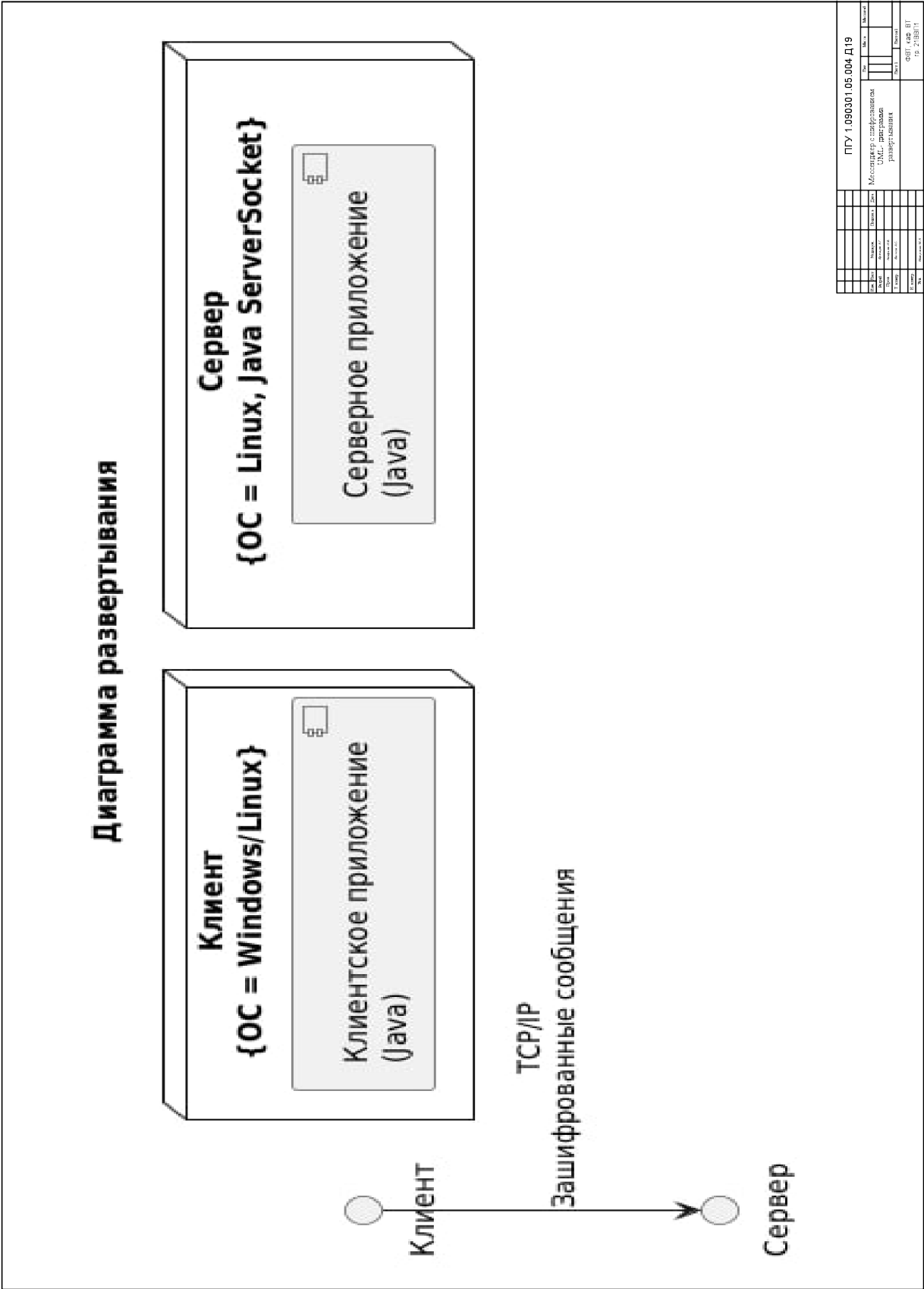
48

Приложение А.2-Диаграмма прецедентов

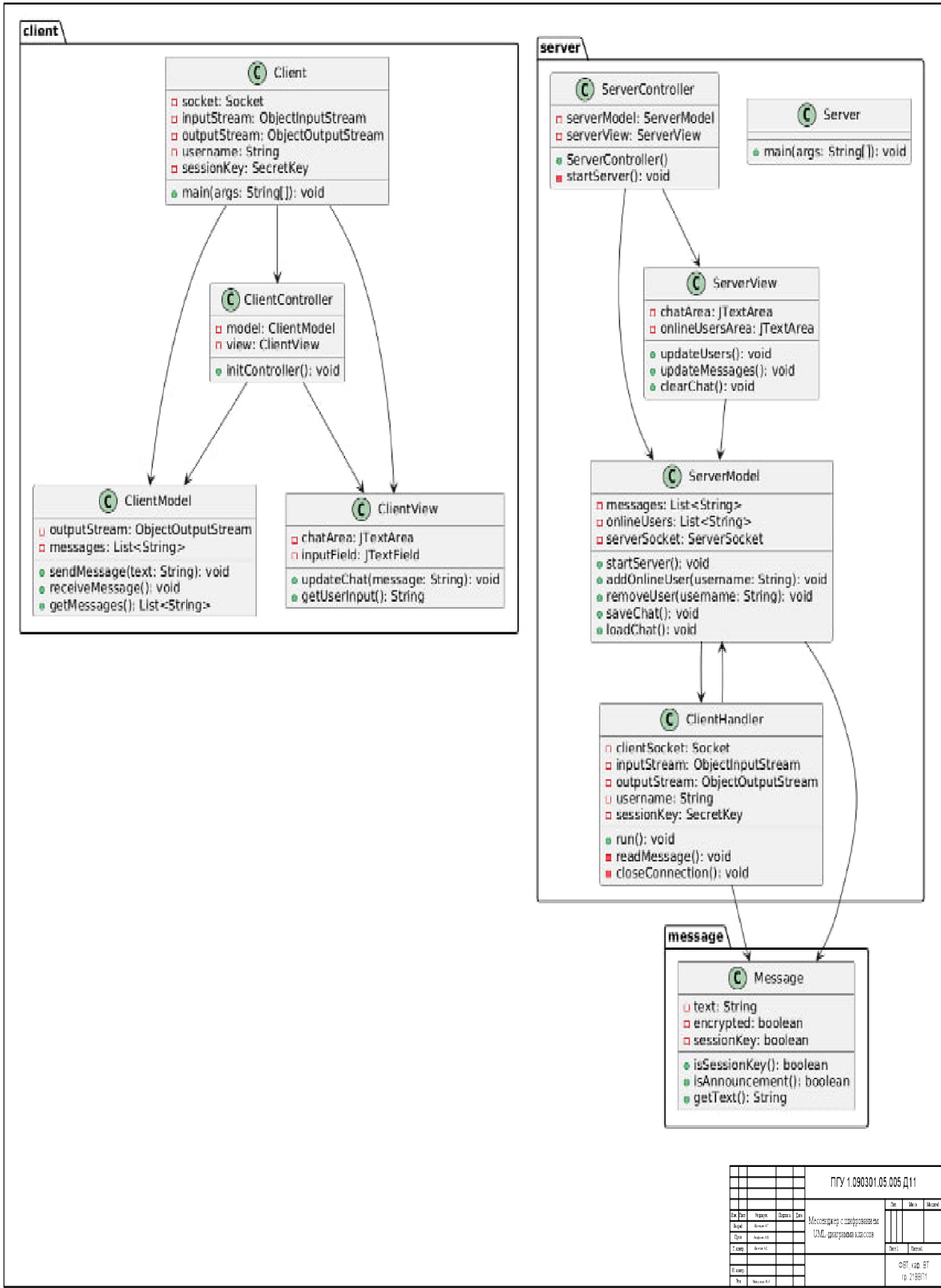


ПГУ 1.090301.05.002 Д13									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

Приложение А.3-Диаграмма развёртывания

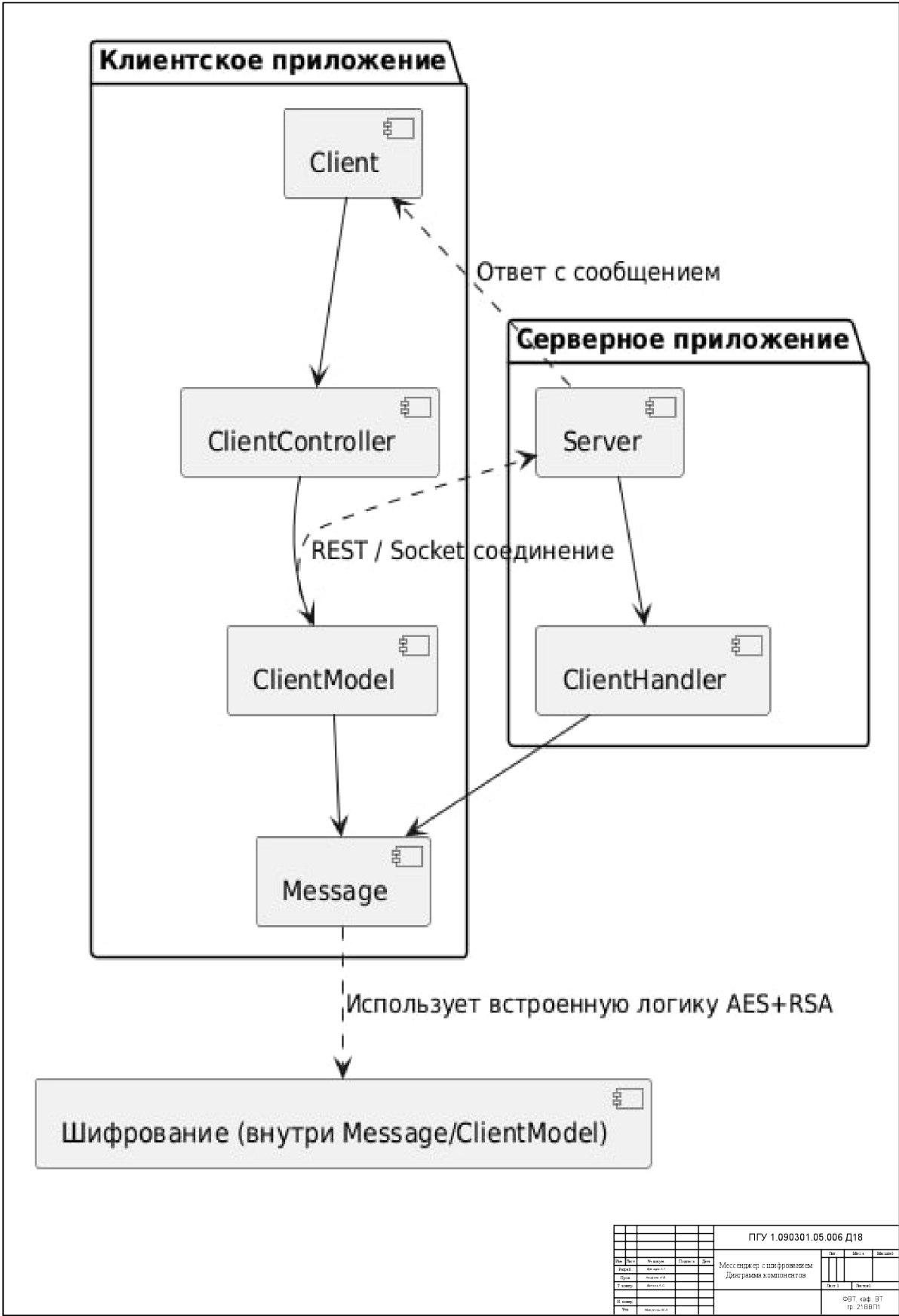


Приложение А.4-Диаграмма классов



ПГУ 1.090301.05.006 Д11									
№	Действ.	Инициал	Фамилия	Имя	Отчество	Дата	Подпись	Место	Должность
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									
26									
27									
28									
29									
30									
31									
32									
33									
34									
35									
36									
37									
38									
39									
40									
41									
42									
43									
44									
45									
46									
47									
48									
49									
50									
51									
52									
53									
54									
55									
56									
57									
58									
59									
60									
61									
62									
63									
64									
65									
66									
67									
68									
69									
70									
71									
72									
73									
74									
75									
76									
77									
78									
79									
80									
81									
82									
83									
84									
85									
86									
87									
88									
89									
90									
91									
92									
93									
94									
95									
96									
97									
98									
99									
100									

Приложение А.4-Диаграмма компонентов



ПГУ 1.090301.05.006 Д18									
№	Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.	Имя	Пол
Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.
Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.
Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.
Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.
Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.
Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.
Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.
Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.	Имя	Пол	Дл.

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(обязательное)

Мессенджер с шифрованием

Листинг программы

					ПГУ1.090301.05.001 ПЗ	Лист
						53
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «Client.java»

```
package client;

/**
 * The main class for running our client
 */
public class Client {
    public static void main(String[] args) {
        new ClientController();
    }
}
```

Файл «ClientController.java»

```
package client;

import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

/**
 * The client controller will connect the client view and the
 * client model.
 * When a user clicks on a button or interacts with the client
 * view, it will call methods in the client model
 * and update the client view.
 * Ex: When a user clicks the send button to send a message in
 * the client view, it will call the logic
 * method in the client model that actually sends the message to
 * the server.
 */
public class ClientController extends JFrame {

    private ClientModel clientModel;
    private ClientView clientView;

    public ClientController() {
        this.clientModel = new ClientModel();
        this.clientView = new ClientView(clientModel);

        //Add the client view as observer to the client model
        clientModel.addObserver(clientView);

        clientView.getConnectButton().addActionListener(new
        ActionListener() {
```

```

        @Override
        public void actionPerformed(ActionEvent e) {
            String username = clientView.getUserName();
            String serverIPAddress =
clientView.getServerIPAddress();
            if (username.isBlank()) {
                clientView.setErrorMessage("Please provide a
valid username");
            } else {
                try {
                    clientModel.connect(username,
serverIPAddress);

                    clientModel.listenForMessages();
                    //System.out.println("Change to
chatPanel");

                    clientView.showChat();
                    setTitle("Client- " + username);
                } catch (IOException exception) {
                    clientView.setErrorMessage("Failed to
connect to the server");
                    exception.printStackTrace();
                } catch (NoSuchPaddingException
noSuchPaddingException) {
                    noSuchPaddingException.printStackTrace();
                } catch (IllegalBlockSizeException
illegalBlockSizeException) {
                    illegalBlockSizeException.printStackTrace();
                } catch (NoSuchAlgorithmException
noSuchAlgorithmException) {
                    noSuchAlgorithmException.printStackTrace();
                } catch (BadPaddingException
badPaddingException) {
                    badPaddingException.printStackTrace();
                } catch (InvalidKeyException
invalidKeyException) {
                    invalidKeyException.printStackTrace();
                } catch (ClassNotFoundException
classNotFoundException) {
                    classNotFoundException.printStackTrace();
                }
            }
        }
    });

    clientView.getBackButton().addActionListener(new
ActionListener() {
        @Override

```

```

        public void actionPerformed(ActionEvent e) {
            //System.out.println("Change to connect panel");
            clientModel.disconnect();
            clientView.showHome();
        }
    });

    clientView.getHomeMenuItem().addActionListener(new
    ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            //System.out.println("Change to connect panel");
            clientModel.disconnect();
            clientView.showHome();
        }
    });

    clientView.getSendButton().addActionListener(new
    ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            String message = clientView.getMessage();
            try {
                clientModel.sendMessage(message);
                //clear the text in the message field after
a client has sent a message
                clientView.getMessageField().setText("");
            } catch (IOException ex) {
                ex.printStackTrace();
                clientModel.disconnect();
                clientView.showHome();
            } catch (IllegalBlockSizeException
illegalBlockSizeException) {
                illegalBlockSizeException.printStackTrace();
            } catch (NoSuchPaddingException
noSuchPaddingException) {
                noSuchPaddingException.printStackTrace();
            } catch (BadPaddingException
badPaddingException) {
                badPaddingException.printStackTrace();
            } catch (NoSuchAlgorithmException
noSuchAlgorithmException) {
                noSuchAlgorithmException.printStackTrace();
            } catch (InvalidKeyException
invalidKeyException) {
                invalidKeyException.printStackTrace();
            }
        }
    });

    //Also allow the client to send messages by pressing
enter on the keyboard

```

```

        clientView.getMessageField().addKeyListener(new
KeyAdapter() {
    @Override
    public void keyPressed(KeyEvent e) {
        String message = clientView.getMessage();
        if (e.getKeyChar() == KeyEvent.VK_ENTER) {
            try {
                clientModel.sendMessage(message);

clientView.getMessageField().setText("");
            } catch (IOException ex) {
                ex.printStackTrace();
            } catch (IllegalBlockSizeException
illegalBlockSizeException) {

illegalBlockSizeException.printStackTrace();
            } catch (NoSuchPaddingException
noSuchPaddingException) {

noSuchPaddingException.printStackTrace();
            } catch (BadPaddingException
badPaddingException) {
                badPaddingException.printStackTrace();
            } catch (NoSuchAlgorithmException
noSuchAlgorithmException) {

noSuchAlgorithmException.printStackTrace();
            } catch (InvalidKeyException
invalidKeyException) {
                invalidKeyException.printStackTrace();
            }
        }
    }
});

        clientView.getSaveMenuItem().addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            clientModel.saveChat();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
});

        clientView.getServerAddressField().addFocusListener(new
FocusListener() {
    @Override
    public void focusGained(FocusEvent e) {

```

```

        if
(clientView.getServerAddressField().getText().equals("Default:
Localhost")) {

clientView.getServerAddressField().setText("");

clientView.getServerAddressField().setForeground(Color.black);
    }
    }

@Override
public void focusLost(FocusEvent e) {
    if
(clientView.getServerAddressField().getText().isBlank()) {

clientView.getServerAddressField().setForeground(Color.gray);

clientView.getServerAddressField().setText("Default:
Localhost");
    }
    }
});

//JFRAME Methods
add(clientView.getClientPanel());
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setJMenuBar(clientView.getMenuBar());
setTitle("Client");
//Pack() makes sure the JFrame gets the same size as the
preferred size of the client view.
pack();
//Centers the JFrame at the center of the window
setLocationRelativeTo(null);
setVisible(true);

}

}

```

Файл «ClientModel.java»

```

package client;

import message.Message;

import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.io.*;
import java.net.Socket;
import java.security.*;
import java.util.ArrayList;
import java.util.Base64;

```

```

/**
 * Stores all the data of our application such as the chat log
 * between the client and the server as well
 * as a list of all the online users
 * Contains methods for communicating with the server: Methods
 * for the client to send messages to the server
 * and methods for the client to read messages from the server
 */
public class ClientModel {

    //messages can't be null because otherwise we will not be
    //able to add any elements to it
    //Initialize it to an empty arraylist
    private ArrayList<String> messages = new ArrayList<>();
    private ArrayList<String> onlineUsers = new ArrayList<>();
    private ObjectOutputStream objectOutputStream;
    private ObjectInputStream objectInputStream;
    private Socket clientSocket;
    private String username;

    //NEW FIELDS
    //The private and public RSA key will be used to securely
    //exchange the session key with the server
    private PublicKey publicKey;
    private PrivateKey privateKey;
    private SecretKey sessionKey;

    private static KeyPair generateKeyPair() {
        //Create a new keypair generator that generates an RSA
        //key pair
        KeyPairGenerator keyPairGenerator = null;
        try {
            keyPairGenerator =
                KeyPairGenerator.getInstance("RSA");
            //Set the key size to 2048 bits, this will be enough
            //for our 1-time use of the key
            keyPairGenerator.initialize(2048);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return keyPairGenerator.generateKeyPair();
    }

    private void generateSessionKey() {
        //Create a key generator that generates a symmetric AES
        //key
        KeyGenerator keyGenerator = null;
        try {
            keyGenerator = KeyGenerator.getInstance("AES");
            //Initialize the key size to 256 bits
            keyGenerator.init(256);
        }
    }
}

```

```

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        this.sessionKey = keyGenerator.generateKey();
    }

    public ClientModel() {
        KeyPair keyPair = generateKeyPair();
        this.privateKey = keyPair.getPrivate();
        this.publicKey = keyPair.getPublic();
    }

    //OBSERVER. ClientView observes the clientModel. The
    clientModel is OBSERVABLE
    private ClientView clientView;

    public void connect(String username, String serverAddress)
    throws IOException, ClassNotFoundException,
    NoSuchPaddingException, IllegalBlockSizeException,
    NoSuchAlgorithmException, BadPaddingException,
    InvalidKeyException {
        clientSocket = new Socket(serverAddress, 1234);
        this.username = username;
        objectOutputStream = new
    ObjectOutputStream(clientSocket.getOutputStream());
        objectInputStream = new
    ObjectInputStream(clientSocket.getInputStream());
        //The first thing the client writes to the server is
        their username
        //this is so that the server can add them to their
        online user list
        objectOutputStream.writeObject(username);

        this.onlineUsers = (ArrayList<String>)
    objectInputStream.readObject();
        clientView.updateUserList();
        System.out.println(onlineUsers.size());
        Message announcement = (Message)
    objectInputStream.readObject();
        messages.add(announcement.toString());
        clientView.updateMessages();

        if (onlineUsers.size() == 1) {
            //generate session key. Don't send it unless another
            client joins
            generateSessionKey();
        } else {
            //send public key to server, server relays it to key
            warden

            objectOutputStream.writeObject(this.publicKey);
            //key warden sends message to its client handler,

```

```

which relays it to our client handler
        Message encryptedSessionKey = (Message)
objectInputStream.readObject();
        //Decrypt the message to obtain a string
representation of the unencrypted session key
        String sessionKey =
encryptedSessionKey.decrypt(this.privateKey);
        this.sessionKey = new
SecretKeySpec(Base64.getDecoder().decode(sessionKey), "AES");
    }
}

    public void sendMessage(String message) throws IOException,
IllegalBlockSizeException, NoSuchPaddingException,
BadPaddingException, NoSuchAlgorithmException,
InvalidKeyException {
        //send a message that is encrypted using the symmetric
session AES session key
        objectOutputStream.writeObject(new Message(username + ":
" + message, this.sessionKey));
    }

    public void readMessage() throws IOException,
ClassNotFoundException, NoSuchPaddingException,
IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
        Object obj = objectInputStream.readObject();
        if (obj instanceof Message) {
            Message message = (Message) obj;
            if (message.isAnnouncement()) {
                //print out message to see if it is the same
encrypted string that we observe in wireshark
                //System.out.println(message.toString());
                //Decrypt all messages that the server sends to
us(client) and add them to the client log in plain text
                //so that the client can read the messages
                messages.add(message.toString());
                clientView.updateMessages();
            } else {
                //print out message to see if it is the same
encrypted string that we observe in wireshark
                //System.out.println(message.toString());
                //Decrypt all messages that the server sends to
us(client) and add them to the client log in plain text
                //so that the client can read the messages
                messages.add(message.decrypt(this.sessionKey));
                clientView.updateMessages();
            }
        }
        } else if (obj instanceof PublicKey) {
            //if key warden receives public key from client
            PublicKey publicKey = (PublicKey) obj;
            objectOutputStream.writeObject(new

```

```

Message(Base64.getEncoder().encodeToString(sessionKey.getEncoded
()), publicKey));
    } else {
        ArrayList<String> onlineUsers = (ArrayList<String>)
obj;

        this.onlineUsers = onlineUsers;
        clientView.updateUserList();
    }
}

public void listenForMessages() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                while (true) {
                    readMessage();
                }
            } catch (IOException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            } catch (NoSuchPaddingException e) {
                e.printStackTrace();
            } catch (IllegalBlockSizeException e) {
                e.printStackTrace();
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            } catch (BadPaddingException e) {
                e.printStackTrace();
            } catch (InvalidKeyException e) {
                e.printStackTrace();
            }
        }
    }).start();
}

public void disconnect() {
    try {
        if (clientSocket.isConnected()) {
            clientSocket.close();
        }
        if (objectOutputStream != null) {
            objectOutputStream.close();
        }
        if (objectInputStream != null) {
            objectInputStream.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

```

    public String getLatestMessage() {
        return messages.get(messages.size() - 1);
    }

    public ArrayList<String> getOnlineUsers() {
        return onlineUsers;
    }

    //Observer pattern will be necessary because the clientmodel
    needs to notify the clientview when
    //it has read a new message. The client controller can not
    do that because the client controller does not know
    // when the client has read a new message from the server.
    Therefore the client model needs to notify the view.
    public void addObserver(ClientView clientView) {
        this.clientView = clientView;
    }

    public void saveChat() throws IOException {
        JFileChooser jFileChooser = new JFileChooser();

        if (jFileChooser.showSaveDialog(null) ==
        JFileChooser.APPROVE_OPTION) {
            File file = jFileChooser.getSelectedFile();
            BufferedWriter bufferedWriter = new
            BufferedWriter(new FileWriter(file));
            for (String msg : messages) {
                bufferedWriter.write(msg);
            }
            bufferedWriter.close();
        }
    }
}

```

Файл «ClientView.java»

```

package client;

import javax.swing.*.*;
import java.awt.*.*;

public class ClientView {
    private JPanel clientPanel;
    private JPanel connectPanel;
    private JTextField usernameField;
    private JTextField serverAddressField;
    private JButton connectButton;
    private JLabel errorMessage;
    private JPanel chatPanel;
}

```

```

private JTextArea chatArea;
private JTextArea onlineUsersArea;
private JTextField messageField;
private JButton sendButton;
private JButton backButton;
private JMenuBar menuBar;
private JMenuItem saveMenuItem;
private JMenuItem homeMenuItem;

private ClientModel clientModel;

public ClientView(ClientModel clientModel) {
    this.clientModel = clientModel;
}

//GETTER METHODS

public JPanel getClientPanel() {
    return clientPanel;
}

public String getUsername() {
    return usernameField.getText().trim();
}

public String getServerIPAddress() {
    if (serverAddressField.getText().isBlank() ||
serverAddressField.getText().equals("Default: Localhost")) {
        return "localhost";
    } else {
        return serverAddressField.getText();
    }
}

public JButton getConnectButton() {
    return connectButton;
}

public JButton getBackButton() {
    return backButton;
}

public String getMessage() {
    return messageField.getText();
}

public JMenuBar getMenuBar() {
    return menuBar;
}

public JButton getSendButton() {
    return sendButton;
}

```

```

    }

    public JTextField getServerAddressField() {
        return serverAddressField;
    }

    public void showChat() {
        CardLayout cardLayout = (CardLayout)
clientPanel.getLayout();
        cardLayout.show(clientPanel, "chat");
    }

    public void showHome() {
        CardLayout cardLayout = (CardLayout)
clientPanel.getLayout();
        cardLayout.show(clientPanel, "home");
    }

    public void updateMessages() {
        chatArea.append(clientModel.getLatestMessage());
    }

    public void updateUserList() {
        onlineUsersArea.setText("");
        for (String user : clientModel.getOnlineUsers()) {
            onlineUsersArea.append(user);
        }
    }

    public JTextField getMessageField() {
        return messageField;
    }

    public JMenuItem getSaveMenuItem() {
        return saveMenuItem;
    }

    public void setErrorMessage(String errorMessage) {
        this.errorMessage.setText(errorMessage);
    }

    public JMenuItem getHomeMenuItem() {
        return homeMenuItem;
    }
}

```

Файл «ClientView.form»

```

<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://www.intellij.com/uide signer/form/"
version="1" bind-to-class="client.ClientView">
    <grid id="27dc6" binding="clientPanel" layout-
manager="CardLayout" hgap="0" vgap="0">

```

```

<constraints>
  <xy x="20" y="20" width="638" height="834"/>
</constraints>
<properties/>
<border type="none"/>
<children>
  <grid id="ed8fc" binding="connectPanel" layout-
manager="GridLayoutManager" row-count="10" column-count="1"
same-size-horizontally="false" same-size-vertically="false"
hgap="-1" vgap="-1">
    <margin top="0" left="0" bottom="0" right="0"/>
    <constraints>
      <card name="home"/>
    </constraints>
    <properties/>
    <border type="none"/>
    <children>
      <component id="9ed67" class="javax.swing.JLabel">
        <constraints>
          <grid row="2" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0"
use-parent-layout="false"/>
        </constraints>
        <properties>
          <font size="16" style="1"/>
          <text value="Username"/>
        </properties>
      </component>
      <vspacer id="58da2">
        <constraints>
          <grid row="9" column="0" row-span="1" col-span="1"
vsize-policy="6" hsize-policy="1" anchor="0" fill="2" indent="0"
use-parent-layout="false"/>
        </constraints>
      </vspacer>
      <component id="622c9" class="javax.swing.JTextField"
binding="usernameField">
        <constraints>
          <grid row="3" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="6" anchor="0" fill="0" indent="0"
use-parent-layout="false">
            <preferred-size width="150" height="-1"/>
          </grid>
        </constraints>
        <properties>
          <font size="16"/>
        </properties>
      </component>
      <component id="9881b" class="javax.swing.JLabel">
        <constraints>
          <grid row="4" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0"

```

```

use-parent-layout="false"/>
    </constraints>
    <properties>
        <font size="16" style="1"/>
        <text value="Server IP Address"/>
    </properties>
</component>
<component id="76a83" class="javax.swing.JTextField"
binding="serverAddressField">
    <constraints>
        <grid row="5" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="6" anchor="0" fill="0" indent="0"
use-parent-layout="false">
            <preferred-size width="150" height="-1"/>
        </grid>
    </constraints>
    <properties>
        <font size="16"/>
        <foreground color="-8355712"/>
        <text value="Default: Localhost"/>
    </properties>
</component>
<component id="52441" class="javax.swing.JButton"
binding="connectButton" default-binding="true">
    <constraints>
        <grid row="7" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="3" anchor="0" fill="0" indent="0"
use-parent-layout="false"/>
    </constraints>
    <properties>
        <font size="18" style="1"/>
        <text value="Connect"/>
    </properties>
</component>
<vspacer id="55e17">
    <constraints>
        <grid row="1" column="0" row-span="1" col-span="1"
vsize-policy="6" hsize-policy="1" anchor="0" fill="2" indent="0"
use-parent-layout="false"/>
    </constraints>
</vspacer>
<component id="23887" class="javax.swing.JLabel"
binding="errorMessage">
    <constraints>
        <grid row="6" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0"
use-parent-layout="false"/>
    </constraints>
    <properties>
        <font size="16" style="1"/>
        <text value=""/>
    </properties>

```

```

        </component>
        <component id="fe4ce" class="javax.swing.JLabel">
            <constraints>
                <grid row="8" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0"
use-parent-layout="false"/>
            </constraints>
            <properties>
                <icon value="end_to_end_encrypted_chat_logo.jpg"/>
                <text value=""/>
            </properties>
        </component>
        <grid id="d52fa" class="javax.swing.JMenuBar"
binding="menuBar" layout-manager="GridLayoutManager" row-
count="1" column-count="4" same-size-horizontally="false" same-
size-vertically="false" hgap="-1" vgap="-1">
            <margin top="0" left="0" bottom="0" right="0"/>
            <constraints>
                <grid row="0" column="0" row-span="1" col-span="1"
vsize-policy="3" hsize-policy="3" anchor="0" fill="1" indent="0"
use-parent-layout="false"/>
            </constraints>
            <properties/>
            <border type="none"/>
            <children>
                <grid id="d7b48" class="javax.swing.JMenu" layout-
manager="FlowLayout" hgap="5" vgap="5" flow-align="1">
                    <constraints>
                        <grid row="0" column="0" row-span="1" col-
span="1" vsize-policy="3" hsize-policy="3" anchor="0" fill="0"
indent="0" use-parent-layout="false"/>
                    </constraints>
                    <properties>
                        <text value="Main"/>
                    </properties>
                    <border type="none"/>
                    <children>
                        <component id="f1d9c"
class="javax.swing.JMenuItem" binding="homeMenuItem">
                            <constraints/>
                            <properties>
                                <text value="Home"/>
                            </properties>
                        </component>
                        <component id="43a09"
class="javax.swing.JMenuItem" binding="saveMenuItem">
                            <constraints/>
                            <properties>
                                <text value="Save"/>
                            </properties>
                        </component>
                    </children>

```

```

        </grid>
        <hspacer id="fde4c">
            <constraints>
                <grid row="0" column="3" row-span="1" col-
span="1" vsize-policy="1" hsize-policy="6" anchor="0" fill="1"
indent="0" use-parent-layout="false"/>
            </constraints>
        </hspacer>
        <grid id="bc4ed" class="javax.swing.JMenu" layout-
manager="FlowLayout" hgap="5" vgap="5" flow-align="1">
            <constraints>
                <grid row="0" column="1" row-span="1" col-
span="1" vsize-policy="3" hsize-policy="3" anchor="0" fill="0"
indent="0" use-parent-layout="false"/>
            </constraints>
            <properties>
                <text value="Settings"/>
            </properties>
            <border type="none"/>
            <children>
                <grid id="dba0f" class="javax.swing.JMenu"
layout-manager="FlowLayout" hgap="5" vgap="5" flow-align="1">
                    <constraints/>
                    <properties>
                        <text value="Sound"/>
                    </properties>
                    <border type="none"/>
                    <children>
                        <component id="94d09"
class="javax.swing.JMenuItem">
                            <constraints/>
                            <properties>
                                <text value="On"/>
                            </properties>
                        </component>
                        <component id="d51db"
class="javax.swing.JMenuItem">
                            <constraints/>
                            <properties>
                                <text value="Off"/>
                            </properties>
                        </component>
                    </children>
                </grid>
                <grid id="69f13" class="javax.swing.JMenu"
layout-manager="FlowLayout" hgap="5" vgap="5" flow-align="1">
                    <constraints/>
                    <properties>
                        <text value="Timestamp"/>
                    </properties>
                    <border type="none"/>
                    <children>

```

```

        <component id="4eb6b"
class="javax.swing.JMenuItem">
        <constraints/>
        <properties>
            <text value="On"/>
        </properties>
        </component>
        <component id="14889"
class="javax.swing.JMenuItem">
        <constraints/>
        <properties>
            <text value="Off"/>
        </properties>
        </component>
    </children>
</grid>
<grid id="3d624" class="javax.swing.JMenu"
layout-manager="FlowLayout" hgap="5" vgap="5" flow-align="1">
    <constraints/>
    <properties>
        <text value="Color"/>
    </properties>
    <border type="none"/>
    <children>
        <component id="15c6d"
class="javax.swing.JMenuItem">
        <constraints/>
        <properties>
            <text value="White"/>
        </properties>
        </component>
        <component id="f5a97"
class="javax.swing.JMenuItem">
        <constraints/>
        <properties>
            <text value="Black"/>
        </properties>
        </component>
    </children>
</grid>
</children>
</grid>
<grid id="c510b" class="javax.swing.JMenu" layout-
manager="FlowLayout" hgap="5" vgap="5" flow-align="1">
    <constraints>
        <grid row="0" column="2" row-span="1" col-
span="1" vsize-policy="3" hsize-policy="3" anchor="0" fill="0"
indent="0" use-parent-layout="false"/>
    </constraints>
    <properties>
        <text value="Help"/>
    </properties>

```

```

        <border type="none"/>
        <children>
            <component id="5b236"
class="javax.swing.JMenuItem">
                <constraints/>
                <properties>
                    <text value="About"/>
                </properties>
            </component>
        </children>
    </grid>
</children>
</grid>
</children>
</grid>
    <grid id="f61fc" binding="chatPanel" layout-
manager="GridLayoutManager" row-count="4" column-count="2" same-
size-horizontally="false" same-size-vertically="false" hgap="-1"
vgap="-1">
        <margin top="10" left="10" bottom="10" right="10"/>
        <constraints>
            <card name="chat"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children>
            <scrollpane id="48985">
                <constraints>
                    <grid row="1" column="0" row-span="2" col-span="1"
vsize-policy="7" hsize-policy="7" anchor="0" fill="3" indent="0"
use-parent-layout="false"/>
                </constraints>
                <properties/>
                <border type="none"/>
                <children>
                    <component id="cb4ae"
class="javax.swing.JTextArea" binding="chatArea">
                        <constraints/>
                        <properties>
                            <editable value="false"/>
                            <font size="16"/>
                            <lineWrap value="true"/>
                            <margin top="0" left="5" bottom="0"
right="0"/>
                        </properties>
                    </component>
                </children>
            </scrollpane>
            <component id="6ed59" class="javax.swing.JTextArea"
binding="onlineUsersArea">
                <constraints>
                    <grid row="1" column="1" row-span="1" col-span="1"

```

```

vsize-policy="6" hsize-policy="0" anchor="4" fill="2" indent="0"
use-parent-layout="false">
    <preferred-size width="150" height="50"/>
</grid>
</constraints>
<properties>
    <editable value="false"/>
    <font size="20" style="1"/>
    <lineWrap value="true"/>
    <margin top="0" left="5" bottom="0" right="0"/>
</properties>
</component>
<component id="d25a4" class="javax.swing.JLabel">
    <constraints>
        <grid row="0" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0"
use-parent-layout="false"/>
    </constraints>
    <properties>
        <font size="16" style="1"/>
        <text value="Chat"/>
    </properties>
</component>
<component id="11d0" class="javax.swing.JLabel">
    <constraints>
        <grid row="0" column="1" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0"
use-parent-layout="false"/>
    </constraints>
    <properties>
        <font size="16" style="1"/>
        <text value="Online Users"/>
    </properties>
</component>
<component id="e0644" class="javax.swing.JTextField"
binding="messageField">
    <constraints>
        <grid row="3" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="6" anchor="8" fill="1" indent="0"
use-parent-layout="false">
        <preferred-size width="150" height="-1"/>
    </grid>
    </constraints>
    <properties>
        <font size="16"/>
    </properties>
</component>
<component id="a3a5a" class="javax.swing.JButton"
binding="sendButton" default-binding="true">
    <constraints>
        <grid row="3" column="1" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="3" anchor="0" fill="1" indent="0"

```

```

use-parent-layout="false"/>
    </constraints>
    <properties>
        <font size="18" style="1"/>
        <text value="Send"/>
    </properties>
</component>
<component id="a8a2e" class="javax.swing.JButton"
binding="backButton" default-binding="true">
    <constraints>
        <grid row="2" column="1" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="3" anchor="0" fill="1" indent="0"
use-parent-layout="false"/>
    </constraints>
    <properties>
        <font size="18" style="1"/>
        <text value="Back"/>
    </properties>
</component>
</children>
</grid>
</children>
</grid>
</form>

```

Файл «Message.java»

```

package message;

import javax.crypto.*;
import java.io.Serializable;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Base64;

/**
 * Class for adding timestamp to messages as well as newline
 * character necessary to display different messages on different
 * lines
 */
public class Message implements Serializable {

    private final String message;
    private boolean isSessionKey = false;
    private boolean announcement = false;

    /**
     * Prepends the current time to the string and stores it.
     */

```

```

    * @param message given String
    */
    public Message(String message) {
        announcement = true;
        String time;
        DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("HH:mm:ss");
        LocalDateTime now = LocalDateTime.now();
        time = "<" + dtf.format(now) + "> ";
        this.message = time + message + "\n";
    }

    /**
     * This constructor is used to encrypt a message sent from
the client to the server using the symmetric session key
generated by the server
     *
     * @param message
     * @param sessionKey A symmetric session key generated by
the server and securely shared with the client
     */
    public Message(String message, SecretKey sessionKey) throws
IllegalBlockSizeException, NoSuchPaddingException,
BadPaddingException, NoSuchAlgorithmException,
InvalidKeyException {
        String time;
        DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("HH:mm:ss");
        LocalDateTime now = LocalDateTime.now();
        time = "<" + dtf.format(now) + "> ";
        this.message = encrypt(time + message + "\n",
sessionKey);
    }

    /**
     * This constructor is used to encrypt messages sent from
the server to the client using the Public key generated by the
client
     * This constructor is only used once by the server to
encrypt the session key and share it securely with the client
     *
     * @param message
     * @param publicKey An assymetric Public Key generated by
the client and shared with the server
     */
    public Message(String message, PublicKey publicKey) throws
NoSuchPaddingException, IllegalBlockSizeException,
BadPaddingException, NoSuchAlgorithmException,
InvalidKeyException {
        //the only thing you ever encrypt with a public key is
the session key.
        this.isSessionKey = true;
    }

```

```

        this.message = encrypt(message, publicKey);
    }

    /**
     * Helper function used to encrypt messages with a symmetric
     session key
     *
     * @param message
     * @param sessionKey
     * @return
     */
    public static String encrypt(String message, SecretKey
sessionKey) throws IllegalBlockSizeException,
BadPaddingException, InvalidKeyException,
NoSuchPaddingException, NoSuchAlgorithmException {
        //Create a new cipher instance that uses AES
        encryption(since our sessionKey is an AES key)
        Cipher cipher = Cipher.getInstance("AES");
        //Initialize the cipher to be used with encryption and
        to use our session key
        cipher.init(Cipher.ENCRYPT_MODE, sessionKey);
        //Encrypt the message(you have to provide the message
        bytes) and store the encrypted bytes
        byte[] encryptedBytes =
        cipher.doFinal(message.getBytes());
        //Convert the encrypted message bytes into its
        corresponding string representation so that it can be sent as a
        message
        return
        Base64.getEncoder().encodeToString(encryptedBytes);
    }

    /**
     * Helper function used to encrypt messages with an
     assymetric public key
     *
     * @param message
     * @param publicKey
     * @return
     */
    public static String encrypt(String message, PublicKey
publicKey) throws BadPaddingException, InvalidKeyException,
NoSuchPaddingException, NoSuchAlgorithmException,
IllegalBlockSizeException {
        //Create a new Cipher that uses RSA encryption(The
        reason is that our public key will be an RSA key)
        Cipher cipher = Cipher.getInstance("RSA");
        //Initialize the cipher to be used with encryption and
        to use our public key
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        //Encrypt the message and store the encrypted message
        bytes

```

```

        byte[] encryptedBytes =
cipher.doFinal(message.getBytes());
        //Convert the encrypted message bytes into corresponding
encrypted string so that it can be sent as message
        return
Base64.getEncoder().encodeToString(encryptedBytes);
    }

    /**
     * Instance method used to decrypt message and return
    decrypted message
     *
     * @param sessionKey AES symmetric session key
     * @return
     */
    public String decrypt(SecretKey sessionKey) throws
NoSuchPaddingException, NoSuchAlgorithmException,
InvalidKeyException, IllegalBlockSizeException,
BadPaddingException {
        //Create a new Cipher instance that uses AES encryption
        Cipher cipher = Cipher.getInstance("AES");
        //Initialize the cipher to be used for decryption with
the provided session key
        cipher.init(Cipher.DECRYPT_MODE, sessionKey);
        //Get the encrypted bytes back from the string message
        //Here you must use Base64.Decoder, since the bytes were
encoded to string using Base64 encoder
        byte[] encryptedBytes =
Base64.getDecoder().decode(this.message);
        //decrypt the encrypted bytes using cipher
        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
        //return the decrypted bytes as a string message
        //Here you CAN and SHOULD use the String constructor
since the decrypted bytes can
        //be represented as a string.
        return new String(decryptedBytes);
    }

    /**
     * Instance method used to decrypt message and return
    decrypted message
     * This method will be used when the client decrypts the
encrypted session key that the server sends to it
     *
     * @param privateKey RSA assymetric private key
     * @return
     */
    public String decrypt(PrivateKey privateKey) throws
NoSuchPaddingException, NoSuchAlgorithmException,
InvalidKeyException, IllegalBlockSizeException,
BadPaddingException {
        //Create a new Cipher that uses RSA encryption

```

```

        Cipher cipher = Cipher.getInstance("RSA");
        //Initialize the cipher to be used with decryption and
use the private RSA key
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        //Decode the String message to its corresponding
encrypted bytes
        //Important to use Base64 decoder and not
this.message.getBytes()
        //The reason is that the String has been encoded with
Base64.encoder
        byte[] encryptedBytes =
Base64.getDecoder().decode(this.message);
        //decrypt the encrypted bytes and store the decrypted
bytes
        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
        //return the decrypted bytes as a regular String using
the String constructor
        //Do not use Base64.encoder here. We want a regular
string.
        return new String(decryptedBytes);
    }

    /**
     * Returns the encapsulated String message
     *
     * @return String object message
     */
    public String toString() {
        return message;
    }

    public String getMessagePart() {
        return message.substring(message.indexOf(">") + 1);
    }

    public boolean isSessionKey() {
        return this.isSessionKey;
    }

    public boolean isAnnouncement() {
        return announcement;
    }
}

```

Файл «ClientHandler.java»

```

package server;

import message.Message;

import javax.crypto.BadPaddingException;

```

```

import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.PublicKey;
import java.util.ArrayList;

public class ClientHandler implements Runnable {

    private ObjectInputStream objectInputStream;
    private ObjectOutputStream objectOutputStream;
    private Socket clientSocket;
    private String username;
    //Can't be null, otherwise we will not be able to add items
    to it
    private static ArrayList<ClientHandler> clientHandlers = new
    ArrayList<>();
    private ServerModel serverModel;
    private SecretKey sessionKey;
    private ClientHandler keyReceiver;

    public ClientHandler(Socket clientConnection, ServerModel
    serverModel) throws IOException, ClassNotFoundException,
    NoSuchPaddingException, IllegalBlockSizeException,
    BadPaddingException, NoSuchAlgorithmException,
    InvalidKeyException {
        this.clientSocket = clientConnection;
        this.serverModel = serverModel;

        clientHandlers.add(this);

        this.objectInputStream = new
        ObjectInputStream(clientSocket.getInputStream());
        this.objectOutputStream = new
        ObjectOutputStream(clientSocket.getOutputStream());

        //read the username that the client sends to the server,
        this is the first item that client sends
        this.username = (String) objectInputStream.readObject();

        serverModel.addOnlineUser(username);
        broadCastUsers(serverModel.getOnlineUsers());

        //the announcement should be sent unencrypted because
        the server does not have any encryption key
        //if the client instead sent the announcement to the
        server to broadcast it, the server would not be able to decrypt

```

```

it
    //we want the announcement to be visible in server log
amongst all the other encrypted ciphertext-
    Message announcement = new Message("SERVER: " + username
+ " has connected to the server");
    serverModel.addMessage(announcement);
    broadCastMessage(announcement);

    if (serverModel.getOnlineUsers().size() > 1) {
        // Only read public key from clients that are not
the key warden
        //if client replies with their public key, this
means they don't have a session key
        PublicKey publicKey = (PublicKey)
objectInputStream.readObject();

        // Find the key warden
        ClientHandler keyWarden = getKeyWarden();
        if (keyWarden != null) {
            // Forward the new client's public key to the
key warden
            keyWarden.setKeyReceiver(this);
            keyWarden.sendPublicKey(publicKey);
        }
    }

    public void setKeyReceiver(ClientHandler clientHandler) {
        this.keyReceiver = clientHandler;
    }

    private void sendPublicKey(PublicKey publicKey) throws
IOException {
        objectOutputStream.writeObject(publicKey);
    }

    private ClientHandler getKeyWarden() {
        // Assuming the first client in the list is the key
warden
        if (!clientHandlers.isEmpty()) {
            return clientHandlers.get(0);
        }
        return null;
    }

    private void broadCastUsers(ArrayList<String> onlineUsers)
throws IOException {
        for (ClientHandler clientHandler : clientHandlers) {
            clientHandler.sendUsers(onlineUsers);
        }
    }

```

```

        public void sendUsers(ArrayList<String> onlineUsers) throws
IOException {
            ObjectOutputStream.writeObject(onlineUsers);
            //reset() is necessary to prevent the ObjectOutputStream
from "remembering" an old version of
            //the onlineUsers array that it has sent, since we will
be sending it multiple times. We always want it to
            //use the newest, up-to-date version of the onlineUsers
array
            ObjectOutputStream.reset();
        }

        private void broadcastMessage(Message message) throws
IOException {
            for (ClientHandler clientHandler : clientHandlers) {
                clientHandler.sendMessage(message);
            }
        }

        public void sendMessage(Message message) throws IOException
{
            //These messages do not need to be encrypted with
session key since they are already encrypted
            //The server simply receives encrypted message from
client and broadcasts it to all other clients.
            ObjectOutputStream.writeObject(message);
        }

        public void readMessage() throws IOException,
ClassNotFoundException, NoSuchPaddingException,
IllegalBlockSizeException, NoSuchAlgorithmException,
BadPaddingException, InvalidKeyException {
            while (true) {
                Message message = (Message)
objectInputStream.readObject();
                System.out.println(message.isSessionKey());
                if (message.isSessionKey()) {
                    //if the message is a session key, we want to
send it to the client that sent their public key

keyReceiver.objectOutputStream.writeObject(message);
                } else {
                    broadcastMessage(message);
                    //the server can not decrypt the message so we
simply add the encrypted ciphertext to the server chat log.
                    serverModel.addMessage(message);
                }
            }
        }

        @Override
        public void run() {

```

```

        try {
            readMessage();
        } catch (IOException e) {
            closeConnection();
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            closeConnection();
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            closeConnection();
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            closeConnection();
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            closeConnection();
            e.printStackTrace();
        } catch (BadPaddingException e) {
            closeConnection();
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            closeConnection();
            e.printStackTrace();
        }
    }

    private void closeConnection() {
        try {
            if (clientSocket.isConnected()) {
                clientSocket.close();
            }
            if (objectInputStream != null) {
                objectInputStream.close();
            }
            if (objectOutputStream != null) {
                objectOutputStream.close();
            }
            //remove the client handler
            clientHandlers.remove(this);
            //remove the username and update the server online
            user list
            serverModel.removeUser(username);
            Message announcement = new Message("SERVER: " +
            username + " has disconnected from the server");
            serverModel.addMessage(announcement);
            broadcastMessage(announcement);
            broadcastUsers(serverModel.getOnlineUsers());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```
}
```

Файл «Server.java»

```
package server;

/**
 * The main class for running our server
 */
public class Server {
    public static void main(String[] args) {
        new ServerController();
    }
}
```

Файл «ServerController.java»

```
package server;

import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

/**
 * The ServerController deals with user input and connects the
 * serverView and the serverModel together.
 * For example, when a user clicks on a button in the
 * ServerView, the controller will call a method
 * in the serverModel.
 */
public class ServerController extends JFrame {

    private ServerModel serverModel;
    private ServerView serverView;

    public ServerController() {
        this.serverModel = new ServerModel();
        this.serverView = new ServerView(serverModel);

        this.serverModel.addObserver(serverView);

        serverView.getSaveChatButton().addActionListener(new
        ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
```

Изм.	Лист	№ докум.	Подпись	Дата

```

        serverModel.saveChat();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}

});

serverView.getLoadChatButton().addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            serverModel.loadChat();
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        }
    }
});

serverView.getStartServerButton().addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        startServer();
    }
});

serverView.getShutDownButton().addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        System.out.println("Shut down the
server...TODO");
    }
});

//JFRAME METHODS
add(serverView.getServerPanel());
pack();
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setTitle("Server");
setVisible(true);

}

private void startServer() {
    new Thread(new Runnable() {
        @Override

```

```

        public void run() {
            //We run this on a separate thread because it is
a blocking operation
            //We want to be able to use the other buttons as
well, so we need to run this on a
            //separate thread
            try {
                serverModel.startServer();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (NoSuchPaddingException e) {
                e.printStackTrace();
            } catch (IllegalBlockSizeException e) {
                e.printStackTrace();
            } catch (BadPaddingException e) {
                e.printStackTrace();
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            } catch (InvalidKeyException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }).start();
}
}

```

Файл «ServerModel.java»

```

package server;

import message.Message;

import javax.crypto.*;
import javax.swing.*;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.InvalidKeyException;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;

/**
 * The server model stores all the data of our application, such
as all the messages that have been sent
 * between the clients and the server as well as a list of all
online users
 */
public class ServerModel {

    private final int PORT_NUMBER = 1234;

```

```

private ArrayList<String> messages = new ArrayList<>();
private ArrayList<String> onlineUsers = new ArrayList<>();
private ServerSocket serverSocket;
private ServerView serverView;

public void startServer() throws IOException,
NoSuchPaddingException, IllegalBlockSizeException,
BadPaddingException, NoSuchAlgorithmException,
InvalidKeyException, ClassNotFoundException {
    serverSocket = new ServerSocket(PORT_NUMBER);
    addMessage(new Message("SERVER: " + "Server has been
started and is listening for connections on port " +
PORT_NUMBER));
    //blocking operation
    while (true) {
        Socket clientConnection = serverSocket.accept();
        ClientHandler clientHandler = new
ClientHandler(clientConnection, this);
        Thread thread = new Thread(clientHandler);
        thread.start();
    }
}

public void addOnlineUser(String username) {
    //The line break is necessary so that the users display
on new lines in the online users text area.
    onlineUsers.add(username + "\n");
    serverView.updateUsers();
}

public void removeUser(String username) {
    onlineUsers.remove(username + "\n");
    serverView.updateUsers();
}

public ArrayList<String> getOnlineUsers() {
    return onlineUsers;
}

public void addMessage(Message message) {
    if (message.isAnnouncement()) {
        messages.add(message.toString());
    } else {
        messages.add(message.toString() + "\n");
    }
    serverView.updateMessages();
}

public String getLatestMessage() {
    return messages.get(messages.size() - 1);
}

```

```

public void addObserver(ServerView serverView) {
    this.serverView = serverView;
}

public void saveChat() throws IOException {
    JFileChooser jFileChooser = new JFileChooser();

    if (jFileChooser.showSaveDialog(null) ==
JFileChooser.APPROVE_OPTION) {
        File file = jFileChooser.getSelectedFile();
        BufferedWriter bufferedWriter = new
BufferedWriter(new FileWriter(file));
        for (String msg : messages) {
            bufferedWriter.write(msg.toString());
        }
        bufferedWriter.close();
    }
}

public void loadChat() throws IOException,
ClassNotFoundException {
    JFileChooser jFileChooser = new JFileChooser();

    if (jFileChooser.showSaveDialog(null) ==
JFileChooser.APPROVE_OPTION) {
        //when loading a chat log, we clear the messages
array so that it only contains the messages in the loaded chat
log
        messages = new ArrayList<>();
        //we have to clear the serverView as well, otherwise
the old messages will not dissappear
        serverView.clearChat();
        File file = jFileChooser.getSelectedFile();
        BufferedReader bufferedReader = new
BufferedReader(new FileReader(file));
        String line = bufferedReader.readLine();
        while (line != null) {
            //we can not add a message again as a message.
this will put 2 timestamps on it
            //therefore we don't store Message class in the
messages array, but simply string
            messages.add(line + "\n");
            serverView.updateMessages();
            line = bufferedReader.readLine();
        }
        bufferedReader.close();
    }
}
}

```

Файл «ServerView.java»

```
package server;

import javax.swing.*.*;
import java.awt.*.*;

public class ServerView {
    private JPanel serverPanel;
    private JTextArea chatArea;
    private JTextArea onlineUsersArea;
    private JButton shutDownButton;
    private JButton startServerButton;
    private JButton loadChatButton;
    private JButton saveChatButton;

    private ServerModel serverModel;

    public ServerView(ServerModel serverModel){
        this.serverModel = serverModel;
        serverPanel.setPreferredSize(new Dimension(450,500));
    }

    public JPanel getServerPanel(){
        return serverPanel;
    }

    public JButton getSaveChatButton(){
        return saveChatButton;
    }

    public JButton getLoadChatButton() {
        return loadChatButton;
    }

    public JButton getShutDownButton() {
        return shutDownButton;
    }

    public JButton getStartServerButton() {
        return startServerButton;
    }

    public void updateUsers() {
        onlineUsersArea.setText("");
        for (String user : serverModel.getOnlineUsers()){
            onlineUsersArea.append(user);
        }
    }

    public void updateMessages() {
        chatArea.append(serverModel.getLatestMessage());
    }
}
```

```

    }

    public void clearChat(){
        chatArea.setText("");
    }

}

```

Файл «ServerView.form»

```

<?xml version="1.0" encoding="UTF-8"?>
<form xmlns="http://www.intellij.com/uideigner/form/"
version="1" bind-to-class="server.ServerView">
    <grid id="27dc6" binding="serverPanel" layout-
manager="GridLayoutManager" row-count="6" column-count="2" same-
size-horizontally="false" same-size-vertically="false" hgap="-1"
vgap="-1">
        <margin top="10" left="10" bottom="10" right="10"/>
        <constraints>
            <xy x="20" y="20" width="500" height="400"/>
        </constraints>
        <properties/>
        <border type="none"/>
        <children>
            <scrollpane id="b730b">
                <constraints>
                    <grid row="1" column="0" row-span="5" col-span="1"
vsize-policy="7" hsize-policy="7" anchor="0" fill="3" indent="0"
use-parent-layout="false"/>
                </constraints>
                <properties/>
                <border type="none"/>
                <children>
                    <component id="ffc40" class="javax.swing.JTextArea"
binding="chatArea">
                        <constraints/>
                        <properties>
                            <editable value="false"/>
                            <font size="16"/>
                            <lineWrap value="true"/>
                            <margin top="0" left="5" bottom="0" right="0"/>
                        </properties>
                    </component>
                </children>
            </scrollpane>
            <component id="35b06" class="javax.swing.JTextArea"
binding="onlineUsersArea">
                <constraints>
                    <grid row="1" column="1" row-span="1" col-span="1"
vsize-policy="6" hsize-policy="0" anchor="4" fill="2" indent="0"
use-parent-layout="false">
                        <preferred-size width="150" height="50"/>
                    </grid>

```

```

        </constraints>
        <properties>
            <editable value="false"/>
            <font size="20" style="1"/>
            <lineWrap value="true"/>
            <margin top="0" left="5" bottom="0" right="0"/>
        </properties>
    </component>
    <component id="af74" class="javax.swing.JLabel">
        <constraints>
            <grid row="0" column="0" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0"
use-parent-layout="false"/>
        </constraints>
        <properties>
            <font size="16" style="1"/>
            <text value="End-to-End Encrypted Chat"/>
        </properties>
    </component>
    <component id="7fdf6" class="javax.swing.JLabel">
        <constraints>
            <grid row="0" column="1" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="0" anchor="0" fill="0" indent="0"
use-parent-layout="false"/>
        </constraints>
        <properties>
            <font size="16" style="1"/>
            <text value="Online Users"/>
        </properties>
    </component>
    <component id="42f2f" class="javax.swing.JButton"
binding="shutDownButton" default-binding="true">
        <constraints>
            <grid row="5" column="1" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="3" anchor="0" fill="1" indent="0"
use-parent-layout="false"/>
        </constraints>
        <properties>
            <font size="18" style="1"/>
            <text value="Shut Down"/>
        </properties>
    </component>
    <component id="8707a" class="javax.swing.JButton"
binding="startServerButton" default-binding="true">
        <constraints>
            <grid row="4" column="1" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="3" anchor="0" fill="1" indent="0"
use-parent-layout="false"/>
        </constraints>
        <properties>
            <font size="18" style="1"/>
            <text value="Start Server"/>

```

```

        </properties>
    </component>
    <component id="62d0c" class="javax.swing.JButton"
binding="loadChatButton" default-binding="true">
        <constraints>
            <grid row="3" column="1" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="3" anchor="0" fill="1" indent="0"
use-parent-layout="false"/>
        </constraints>
        <properties>
            <font size="18" style="1"/>
            <text value="Load Chat"/>
        </properties>
    </component>
    <component id="78027" class="javax.swing.JButton"
binding="saveChatButton" default-binding="true">
        <constraints>
            <grid row="2" column="1" row-span="1" col-span="1"
vsize-policy="0" hsize-policy="3" anchor="0" fill="1" indent="0"
use-parent-layout="false"/>
        </constraints>
        <properties>
            <font size="18" style="1"/>
            <text value="Save Chat"/>
        </properties>
    </component>
</children>
</grid>
</form>

```

Приложение В

(обязательное)

Мессенджер с шифрованием

Презентация

Мессенджер с шифрованием

Автор работы: студент гр. 21ВВП1 Брянцев А. Г.

Научный руководитель: к.т.н. доцент кафедры ВТ Акифьев И. В.

Цель и актуальность проекта

В условиях роста киберугроз всё больше пользователей нуждаются в безопасных каналах связи. Большинство мессенджеров не дают полного контроля над конфиденциальностью.

Целью проекта является разработка кроссплатформенного мессенджера на Java с шифрованием сообщений (AES + RSA) и клиент-серверной архитектурой. Сервер не имеет доступа к содержимому сообщений.

Проект демонстрирует на практике принципы сетевого взаимодействия и криптографии, подходит для учебных и исследовательских задач.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.05.001 ПЗ

Лист

91

Краткая постановка задачи

- подключение пользователей к серверу по имени;
- обмен зашифрованными сообщениями между клиентами (AES + RSA);
- передачу сессионных ключей через сервер без доступа к содержимому;
- отображение списка онлайн-пользователей;
- сохранение и загрузку истории переписки.

Обзор аналогов

- **Signal** – открытый код, полное сквозное шифрование, минимум метаданных.
- **Telegram** – шифрование только в «секретных чатах», часть кода закрыта.
- **Threema** – высокая анонимность, платный, не требует номера телефона.
- **WhatsApp** – массовый, шифрование по умолчанию, принадлежит Meta.
- **Matrix** – децентрализация, открытый протокол, сложен в настройке.
- **Retroshare** – P2P, высокая анонимность, сложный интерфейс.

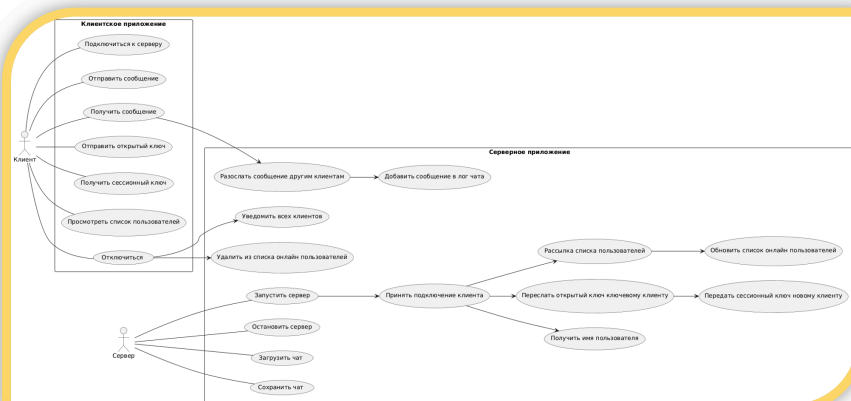


Изм.	Лист	№ докум.	Подпись	Дата

Описание функционала программы

Программа представляет собой клиент-серверный мессенджер с шифрованием. Клиенты подключаются к серверу, указывая имя и IP-адрес. После подключения происходит обмен публичными ключами и передача сессионного ключа, с помощью которого шифруются все сообщения (AES + RSA). Сервер лишь ретранслирует зашифрованные данные, не имея доступа к содержимому. Пользовательский интерфейс, реализованный на Java Swing, позволяет просматривать чат, видеть список онлайн-пользователей и сохранять историю переписки. Все соединения обрабатываются в отдельных потоках для стабильной работы системы.

Диаграмма прецедентов



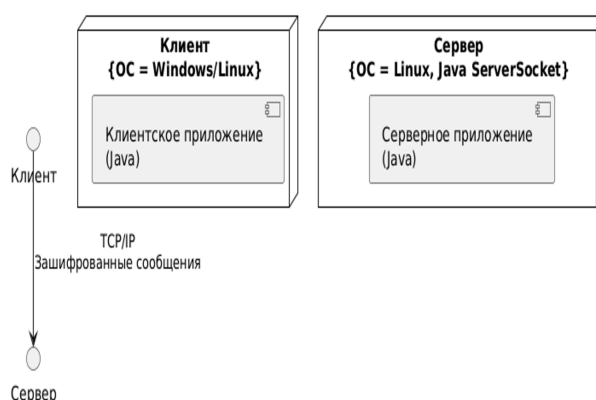


Стек технологий

Клиентская часть мессенджера разработана на языке Java с использованием архитектурного шаблона MVC. Графический интерфейс реализован с помощью библиотеки Java Swing. Для установления соединения с сервером и обмена данными применяется протокол TCP/IP и механизм сокетов. Шифрование сообщений выполняется на стороне клиента: используется симметричный алгоритм AES, а обмен ключами осуществляется с помощью RSA. Вся криптография реализована через стандартную Java Cryptography Architecture (JCA). Обработка входящих сообщений вынесена в отдельный поток, что обеспечивает стабильность пользовательского интерфейса и возможность приёма данных в реальном времени.

Серверная часть также реализована на Java и использует многопоточную архитектуру: для каждого клиента создаётся отдельный поток (ClientHandler). Сервер принимает подключения через ServerSocket и пересылает зашифрованные сообщения между клиентами, не расшифровывая их. Графический интерфейс администратора реализован на Swing и отображает список пользователей и лог сообщений. Передача сообщений между клиентом и сервером осуществляется в виде сериализованных Java-объектов. Кроме того, сервер поддерживает сохранение и загрузку истории переписки через файловую систему, без использования базы данных.

Диаграмма развёртывания



```
sequenceDiagram
    participant Client
    participant ClientModel
    participant ClientController
    participant ClientView
    participant Сетев
    participant ServerModel
    participant ClientHandler

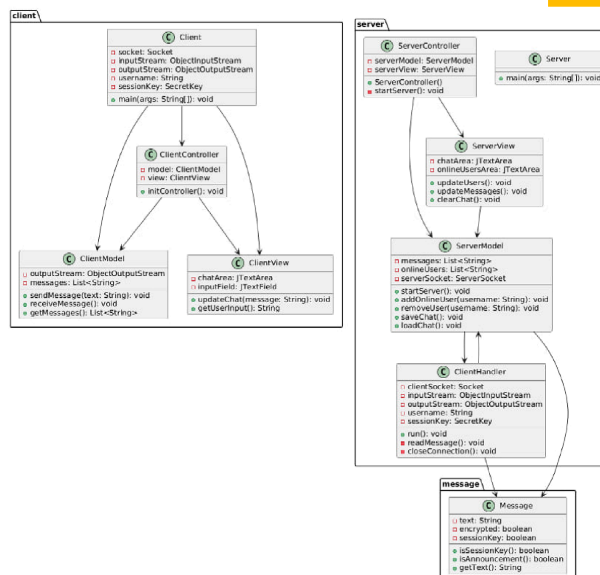
    Client->>ClientModel: Ввод сообщения
    activate ClientModel
    ClientModel->>ClientController: onSendMessage()
    activate ClientController
    ClientController->>ClientModel: sendMassage(text)
    deactivate ClientController
    ClientModel->>Сетев: Отправка зашифрованного сообщения
    activate Сетев
    Сетев->>ServerModel: получить сообщение
    activate ServerModel
    ServerModel->>ClientHandler: передать клиенту
    activate ClientHandler
    ClientHandler->>ServerModel: расшифровка и обработка (ответ)
    deactivate ClientHandler
    ServerModel->>ClientHandler: отправка другим клиентам (ответ)
    deactivate ServerModel
    ClientHandler->>ClientModel: зашифрованное сообщение
    deactivate ClientHandler
    ClientModel->>ClientView: показать сообщение
    deactivate ClientModel
```

```

graph TD
    subgraph "Клиентское приложение"
        Client[Client] --> ClientController[ClientController]
        ClientController --> ClientModel[ClientModel]
        ClientModel --> Message[Message]
    end
    subgraph "Серверное приложение"
        Server[Server] --> ClientHandler[ClientHandler]
    end
    ClientController -. "REST / Socket соединение" .-> Server
    ClientHandler -. "Ответ с сообщением" .-> Client
    Message -.->|Использует встроенную логику AES+RSA| Encryption[Шифрование (внутри Message/ClientModel)]

```

Диаграмма классов



Итоги разработки программы

Создано клиент-серверное десктопное приложение — мессенджер с встроенным шифрованием, обеспечивающий защищённую и конфиденциальную передачу сообщений между пользователями.

Для защиты данных применены алгоритмы AES и RSA: шифрование выполняется на стороне клиента, что исключает возможность перехвата и расшифровки сообщений на сервере.

Использованы технологии Java и Java Swing для реализации функционального и удобного интерфейса, а также Socket API и сериализация объектов для стабильного сетевого взаимодействия.

Приложение работает на Windows и Linux, отличается простотой использования и базовым уровнем безопасности без привлечения сторонних библиотек.

В перспективе предусмотрено расширение функционала: групповые чаты, обмен файлами, история сообщений и дополнительные механизмы аутентификации для повышения удобства и безопасности.



Изм.	Лист	№ докум.	Подпись	Дата