

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

ANDROID - ПРИЛОЖЕНИЕ

«МЕССЕНДЖЕР ДЛЯ СТУДЕНТОВ ПГУ»

Студент

25.05.26
(подпись, дата)

Корнилов Владислав Михайлович
(ФИО полностью)

Руководитель

[подпись]
(подпись, дата)

30.05.26

А.В. Кузнецов
(фамилия, инициалы)

Нормоконтролёр

[подпись]
(подпись, дата)

30.05.26

А.С. Бычков
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой

[подпись]
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

19 июня 2026, № 36)

Секретарь ГЭК

[подпись]

Е.И. Гурин
(подпись)

(фамилия, инициалы)

Пенза 2026

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 7 » апреля 20 26 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Корнилов Владислав Михайлович гр. 22ВВП2 факультета вычислительной техники

2. Тема работы Android - приложение «Мессенджер для студентов ПГУ»

Тема утверждена приказом ПГУ № 347/о _____ от "07" _____ апреля _____ 2026 г.

3. Руководитель бакалаврской работы доцент кафедры ВТ ПГУ Кузнецов А.В.

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

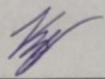
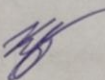
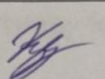
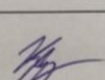
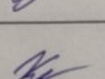
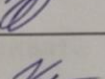
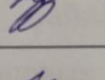
Разработать Android - приложение «Мессенджер для студентов ПГУ», предоставляющее среду для общения и совместной работы студентов, и его серверную часть. Язык программирования Kotlin, среда разработки клиентской части Android Studio, среда разработки серверной части IntelliJ IDEA. Программа должна обладать интуитивно понятным интерфейсом и содержать следующие технологии:

1. Firebase Cloud Messaging
2. Асинхронное программирование
3. JSON Web Token
4. Material design 3

5. Перечень подлежащих разработке вопросов

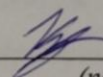
Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки мобильного приложения и серверной части. Выбрать язык программирования, среду разработки и базу данных

6. Календарный график выполнения работы

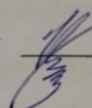
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	26	07.04.26 – 09.04.26	
2	Анализ требований	20	10.04.26 – 12.04.26	
3	Выбор технологий, применяемых при проектировании и разработке	16	13.04.26 – 14.04.26	
4	Разработка серверной части	82	15.04.26 – 30.04.26	
5	Создание графического интерфейса	52	01.05.26 – 08.05.26	
6	Разработка мобильного приложения	92	09.05.26 – 18.05.26	
7	Оформление пояснительной записки	36	19.05.26 – 21.05.26	

Дата выдачи задания "07" апреля 2026г.

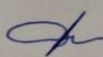
Руководитель бакалаврской работы

 25.05.2026
(подпись, дата)

Задание к исполнению принял студент

 25.05.2026
(подпись, дата)

Работу к защите допустить
Декан факультета

 Л.Р. Писарев
19.06.2020 (подпись, дата)

Реферат

Бакалаврская работа содержит 194 листа, 47 рисунков, 0 формул, 5 таблиц, 10 источников, 3 приложения.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, МЕССЕНДЖЕР, СТУДЕНЧЕСКОЕ СООБЩЕСТВО, ANDROID, KOTLIN, SPRING BOOT, MYSQL, WEBSOCKET, RETROFIT, MVVM, АВТОРИЗАЦИЯ, БЕЗОПАСНОСТЬ ДАННЫХ

Цель работы — проектирование и разработка клиент-серверного программного комплекса «Студенческий мессенджер» для внутривузовской коммуникации.

Мобильное приложение должно предоставлять изолированную и безопасную среду для общения студентов, в которой регистрация и последующее взаимодействие пользователей жестко привязаны к их факультетам и учебным группам. Мессенджер должен обеспечивать обмен мгновенными текстовыми сообщениями и мультимедийными файлами в реальном времени, поддерживать создание и модерирование групповых чатов, предоставлять инструменты для поиска сообщений по истории переписки, а также позволять редактировать данные личного профиля. Чтобы данные пользователей надежно сохранялись, а сессии оставались защищенными, необходимо реализовать клиент-серверную систему авторизации.

В конечном итоге был разработан клиент-серверный программный комплекс, состоящий из мобильного Android-приложения и серверной части на базе фреймворка Spring Boot. Для хранения информации развернута СУБД MySQL. Безопасность сессий обеспечена использованием JWT-токенов и фильтров Spring Security, а архитектура клиентской части построена на базе паттерна MVVM с использованием StateFlow и корутин.

					ПГУ1.090301.37.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата	Android-Приложение «Мессенджер для студентов ПГУ» Пояснительная записка		
Разраб.		Корнилов В.М.		25.05.21			
Провер.		Кузнецов А.В.		30.05.21			
Н. Контр.		Бычков А.С.		30.05.21			
Утверд.		Митрохин М.А.		30.05.21			
					Лит.	Лист	Листов
						1	75
					ФВТ, каф. ВТ гр. 22ВВП2		

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Мобильное приложение «Мессенджер для студентов ПГУ».....	8
1.1 Анализ темы	8
1.2 Анализ аналогов.....	9
1.2.1 Мессенджер «Telegram».....	10
1.2.2 Мессенджер «VK»	11
1.2.3 Мессенджер «MAX».....	13
1.2.4 Мессенджер «WhatsApp».....	15
1.2.5 Вывод	17
1.3 Постановка задачи	18
2 Выбор инструментов разработки	20
2.1 Выбор языка программирования.....	20
2.1.1 Язык программирования «Kotlin».....	20
2.1.2 Вывод	21
2.2 Выбор фреймворка для разработки серверной части	21
2.3 Выбор среды разработки.....	22
2.3.1 Среда разработки Android Studio	22
2.3.2 Среда разработки IntelliJ IDEA	24
2.4 Выбор системы хранения данных	25
2.5 Настройка среды разработки	26
2.5.1 Настройка среды для разработки клиентской части	26
2.5.2 Настройка среды для разработки серверной части	27
3 Архитектура приложения.....	29
3.1 Архитектурный паттерн клиентской части «MVVM».....	29
3.2 Архитектура серверной части «Layered Architecture»	30
4 Реализация приложения	32

4.1	Описание функционала	32
4.2	Описание работы приложения.....	37
4.3	Основная реализация приложения	39
4.3.1	Слой представления (View)	39
4.3.2	Слой модели представления (ViewModel)	41
4.3.3	Слой данных (Model).....	42
5	Тестирование приложения	44
5.1	Регистрация.....	44
5.2	Просмотр и редактирование профиля.....	51
5.3	Отправка сообщений	52
5.4	Создание и редактирование групп	60
5.5	Поиск сообщений.....	66
6	Безопасность жизнедеятельности.....	70
6.1	Организация безопасности условий труда при работе с ПК.....	70
6.2	Техника безопасности при эксплуатации мобильных устройств	71
7	Заключение	73
	Список использованных источников	74
	Приложение А – UML диаграммы	75
	Приложение Б – Листинг программы	80
	Приложение В – Презентация.....	189

Перечень сокращений и обозначений

В данной работе применяются следующие сокращения и обозначения.

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

СУБД – система управления базами данных

API – (от англ. Application Programming Interface) – программный интерфейс приложений

ID – (от англ. identifier) – уникальный код или значение

IDE – (от англ. Integrated Development Environment) – интегрированная среда разработки

HTTP – (от англ. HyperText Transfer Protocol) – протокол обмена данными между клиентом и сервером

JPA – (от англ. Java Persistence API) – API для работы с базами данных в Java-приложениях

JSON – (от англ. JavaScript Object Notation) – текстовый формат обмена данными

JWT – (от англ. JSON Web Token) – открытый стандарт для создания токенов доступа

REST – (от англ. Representational State Transfer) – стиль взаимодействия компонентов приложения в сети

SDK – (от англ. Software Development Kit) – набор инструментов, библиотек, документации и других ресурсов

SQL – (от англ. Structured Query Language) – язык структурированных запросов

UI – (от англ. User Interface) – пользовательский интерфейс

UML – (от англ. Unified Modeling Language) – унифицированный язык моделирования

					<i>ПГУ1.090301.37.001 ПЗ</i>	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В эпоху стремительно растущей цифровизации в среде образования коммуникация является одной из важнейших её составляющих. Быстрый обмен информацией между студентами, возможность отправки учебных материалов и координирование работы в группе напрямую влияют на эффективность обучения. Во многом инструменты, позволяющие удобно коммуницировать студентам, напрямую влияют на вовлеченность студентов в образовательный процесс.

Однако использование популярных мессенджеров не всегда дает все необходимое для учебных целей. Наличие не связанного с учебой контента и обилие рекламы снижает уровень концентрации студентов. К тому же не узконаправленные решения не учитывают специфику учебной среды, что усложняет поиск контактов однокурсников и часто сводится к обычному обмену номерами. Это приводит к тому, что личные переписки смешиваются с учебными, из-за чего поиск необходимой информации становится затруднительным, а шансы упустить важное сообщение увеличиваются.

Решением этой проблемы станет разработка специализированного программного обеспечения – мессенджера для студентов определенного образовательного учреждения. Такое приложение должно предоставлять изолированную среду для общения, в которой регистрация и коммуникация будут тесно связаны с группой и факультетом студентов.

Цели выпускной квалификационной работы:

- изучить тему использования продукта;
- определить функциональные и нефункциональные требования к разрабатываемой системе;
- разработать серверную часть с реализацией механизмов авторизации и обработки сообщений в реальном времени, а также безопасного хранения данных;

- разработать клиентскую часть для ОС Android с использованием современных подходов к проектированию;
- провести тестирование на основе сценариев пользовательского взаимодействия;
- оценить практическую применимость и отказоустойчивость приложения;
- проанализировать полученные результаты и предложить варианты дальнейшего развития приложения.

					ПГУ1.090301.37.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Мобильное приложение «Мессенджер для студентов ПГУ»

1.1 Анализ темы

В современной образовательной среде эффективная коммуникация между студентами является ключевым фактором успешного обучения и социализации. Использование специализированных приложений в учебном процессе полезно по нескольким причинам:

- быстрый обмен информацией. Студенты могут делиться учебными материалами, новостями о расписании и важными объявлениями;
- организация коллективной работы. Мессенджер позволяет упростить координирование действий при выполнении групповых проектов, лабораторных работ и подготовке к экзаменам;
- создание учебного сообщества. Взаимодействие в рамках одного приложения помогает наладить связи между студентами разных курсов и групп, создавая единую информационную среду;
- структурирование знаний. Возможность быстрого поиска по истории сообщений и обмена файлами позволяет быстро найти необходимые материалы.

Качественная коммуникация играет важную роль в адаптации первокурсников, повышении успеваемости и формировании профессиональных связей.

Однако зачастую студенты сталкиваются с проблемой «информационного шума» в популярных мессенджерах, где учебные чаты смешиваются с личными переписками, развлекательными каналами и рекламой. Это приводит к потере важных уведомлений и снижению концентрации на учебе. Для решения этой проблемы необходимо выделенное пространство, ориентированное исключительно на академические задачи.

Процесс формирования устойчивого учебного сообщества требует времени и удобных инструментов. Наличие специализированного приложения повышает

вовлеченность студентов в образовательный процесс и снижает уровень стресса, связанного с поиском информации. Мессенджер для студентов позволяет изолировать учебную коммуникацию от повседневного общения и улучшить образовательный процесс.

Внедрение такого приложения решает сразу несколько задач:

- формирует единую, безопасную и структурированную среду для учебного сообщества;
- снижает когнитивную нагрузку за счет отсутствия стороннего контента и рекламы;
- позволяет адаптировать интерфейс и функционал под конкретные нужды обучающихся.

Студенческий Мессенджер — это приложение, в котором пользователь может общаться с однокурсниками, создавать тематические чаты, обмениваться учебными файлами и управлять своим профилем. Тема разработки такого приложения актуальна в эпоху цифровизации образования, когда мобильность и чистота информационного потока становятся главными требованиями к программному обеспечению.

1.2 Анализ аналогов

В 2026 году на рынке представлено множество качественных программных решений для обмена сообщениями, в которых студенты могут создавать общие чаты, вести заметки и обмениваться файлами [1]. В данном пункте рассмотрены такие популярные приложения как:

- позволяет адаптировать интерфейс и функционал под конкретные нужды обучающихся.
- Telegram;
- VK;
- MAX;

– WhatsApp.

В ходе проведенного анализа выявлены слабые и сильные стороны каждого из приложений.

1.2.1 Мессенджер «Telegram»

Приложение представлено на рисунке 1 и 2.

Один из самых популярных кроссплатформенных мессенджеров, предлагающий широкий функционал для общения и потребления контента [2].

Достоинства:

– широкие возможности модерации. Наличие открытого API позволяет создавать ботов для автоматизации учебных процессов, а также детально настраивать права администраторов в группах;

– удобная работа с файлами. Поддержка отправки любых типов файлов и их хранение в облаке, что необходимо для обмена тяжелыми проектами и методическими материалами;

– кроссплатформенность. Быстрый доступ к перепискам и файлам с любого устройства.

Недостатки:

– переизбыток информации. Учебные чаты теряются среди новостных каналов, развлекательного контента и рекламы;

– нестабильность работы. В связи с ограничениями стабильная работа мессенджера зачастую требует использования сторонних сервисов. Это создает технические барьеры для части студентов и может приводить к задержкам в получении важной учебной информации;

– ограниченный функционал. Наличие платной подписки делает определенные функции недоступными или ограниченными для обычных пользователей.

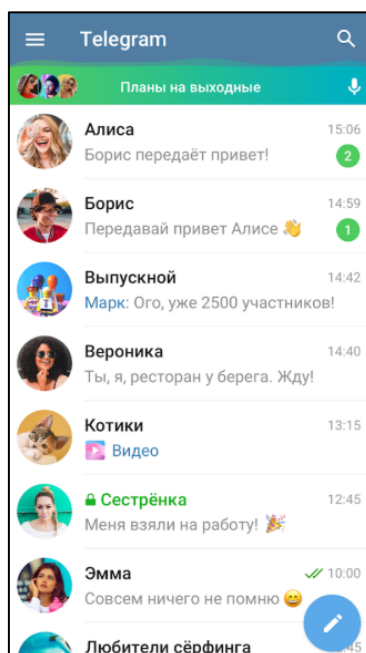


Рисунок 1 - Главная страница со списком чатов

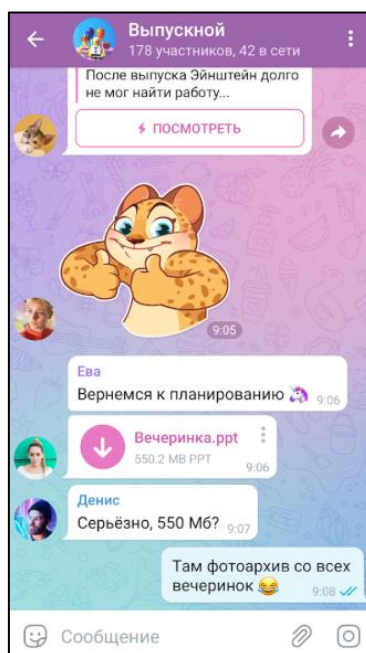


Рисунок 2 - Страница окна чата

1.2.2 Мессенджер «VK»

Многофункциональная платформа, в которой личные сообщения является лишь одной из множества встроенных функций. Часто используется студентами

для общения по инерции, так как большинство групп и бесед создаются именно там.

Достоинства:

- привычная среда. Большинство студентов уже имеют там профиль, что упрощает поиск одногруппников и учащихся в университете;
- интеграция с сервисами. Возможность легко прикреплять документы, а также медиафайлы напрямую из экосистемы социальной сети;
- прозрачность контактов. В отличие от аналогов, пользователи чаще всего используют реальные имена и фамилии, что упрощает идентификацию одногруппников и преподавателей в беседах.

Недостатки:

- перегруженность. Наличие множества функций, которые не относятся к образовательному процессу и перегружают приложение, заставляя его использовать больший объем оперативной памяти;
- работа с файлами. Отсутствие отправки больших архивов данных усложняет и замедляет процесс обмена файлами.

Интерфейс приложения представлен на рисунке 3 и 4.

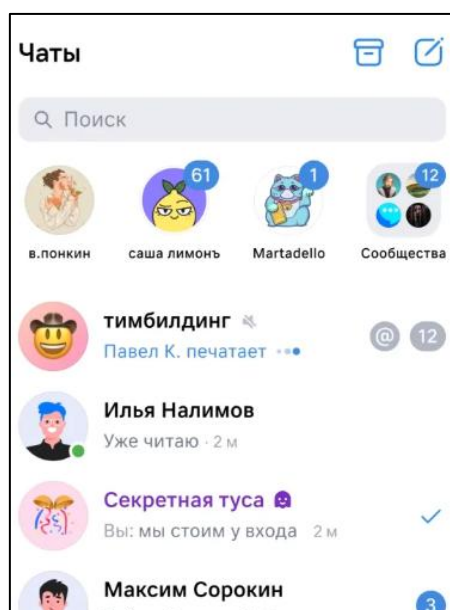


Рисунок 3 – Страница со списком чатов

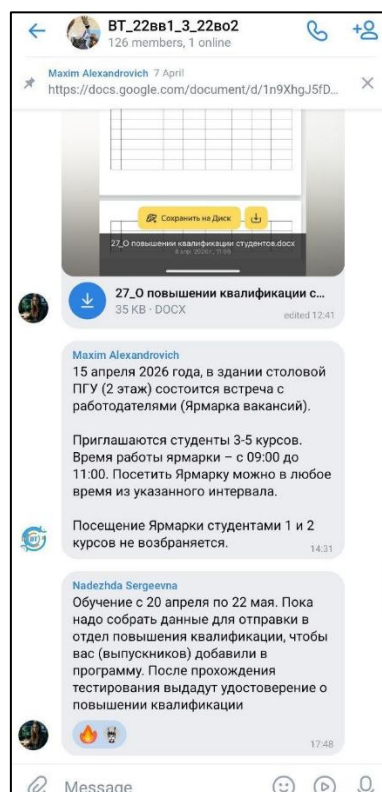


Рисунок 4 - Страница окна чата

1.2.3 Мессенджер «МАХ»

Интерфейс приложения представлен на рисунке 5 и 6.

Современный мессенджер, разработанный компанией VK в 2025 году, объединяющий общение, государственные сервисы и цифровые инструменты. Позиционируется как национальная платформа и аналог зарубежных решений.

Достоинства:

- многофункциональность. Помимо стандартных сообщений и звонков, приложение объединяет платежи, цифровой ID, электронную подпись и доступ к государственным сервисам;
- работа с большим объемом данных. Поддержка отправки файлов до 4 ГБ упрощает учебный процесс;
- единая цифровая экосистема. Возможность использовать мессенджер как приложение «всё в одном», для общения с государственными структурами, учебными заведениями и бизнесом.

Недостатки:

– безопасность и репутация. Из-за распространения недостоверной информации в сети и отсутствия end-to-end шифрования имеет предвзятое отношение среди пользователей;

– перегруженность функциями. Наличие государственных сервисов, платежей и мини-приложений усложняет интерфейс и отвлекает от основной функции общения в контексте учебы.

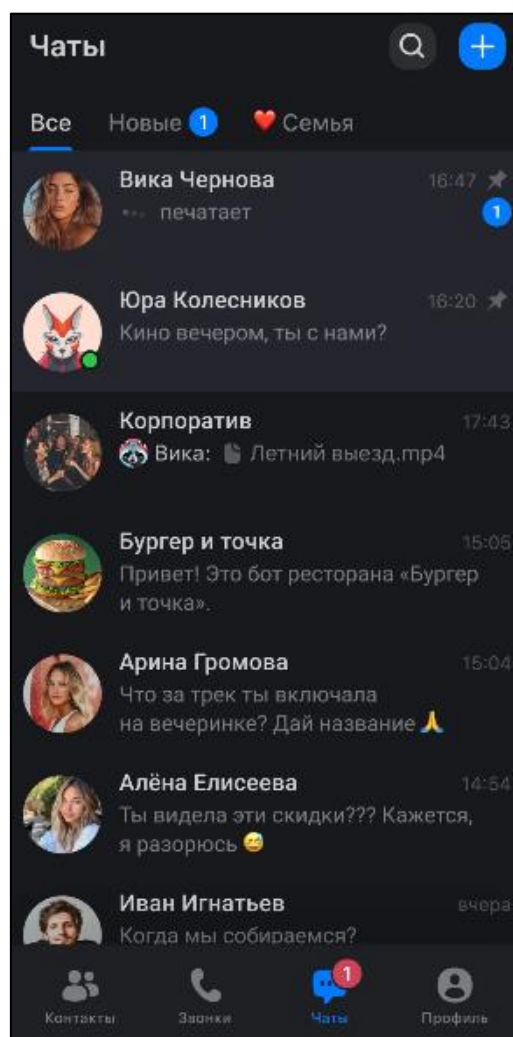


Рисунок 5 - Главная страница со списком чатов

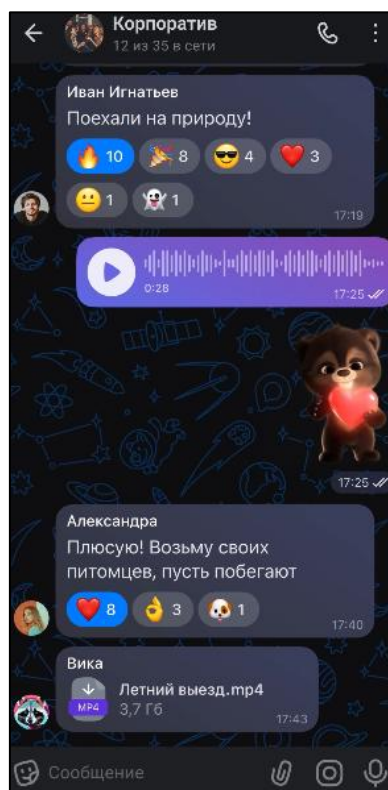


Рисунок 6 – Страница окна чата

1.2.4 Мессенджер «WhatsApp»

Интерфейс приложения представлен на рисунке 7 и 8.

Базовый мессенджер, ориентированный на быстрое и удобное общение. Широко используется как в личных, так и в учебных и рабочих целях.

Достоинства:

- простота использования. Интуитивно понятный интерфейс позволяет быстро освоить приложение даже новым пользователям;
- синхронизация с контактами телефона. Приложение автоматически добавляет существующие контакты, упрощая поиск;
- бесплатное использование. Нет платных тарифов и ограничений на функционал приложения.

Недостатки:

- ограниченный функционал групп. Отсутствие продвинутых инструментов модерации в больших беседах;

- ограничения на передачу файлов. Максимальный размер отправляемых файлов ниже, чем у некоторых аналогов;
- нестабильность работы. Ограниченная стабильная работа мессенджера требует использования сторонних сервисов.

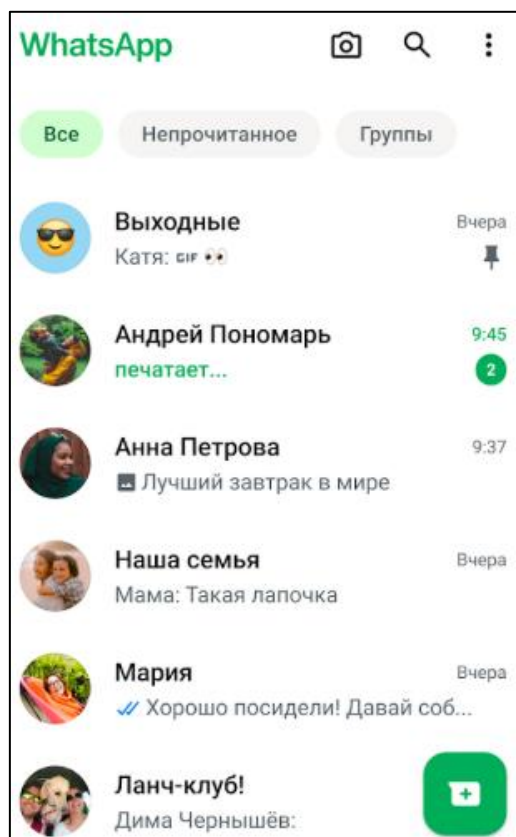


Рисунок 7 – Главная страница со списком чатов

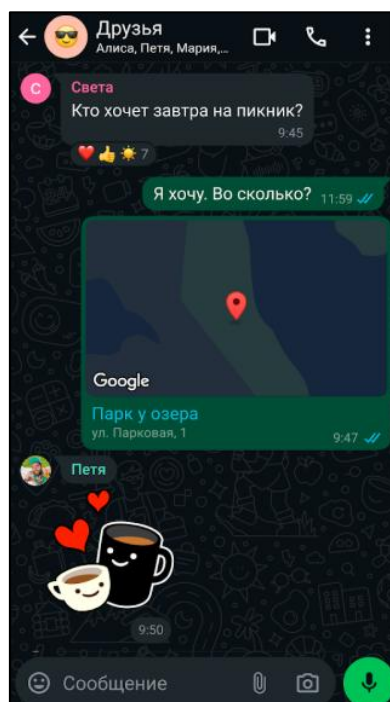


Рисунок 8 - Страница окна чата

1.2.5 Вывод

Таблица 1 – Достоинства и недостатки мессенджеров

	Telegram	VK	MAX	WhatsApp
Отсутствие перегруженности	-	-	-	+
Удобство обмена файлами	+	-	+	-
Специализирован под учебу	-	-	-	-
Стабильность	-	+	+	-
Отсутствие платных ограничений	-	+	+	+
Безопасность	+	-	-	+

Проанализировав существующие мессенджеры, можно отметить, что базовый функционал обмена сообщениями реализован во всех аналогах на высоком уровне. Основные отличия заключаются в экосистеме, дополнительных модулях и изначальном позиционировании продуктов. Также важно понимать, что мессенджер в контексте учебы — это рабочий инструмент. Он должен быть простым, сфокусированным на конкретных задачах и комфортным для студента, помогая структурировать коммуникацию, а не отвлекать от нее.

Поэтому создание локального студенческого мессенджера, который обеспечит отсутствие отвлекающих факторов, стабильную работу и удобный файлообмен с четким разделением пользователей по факультетам и учебным группам, является актуальной задачей и полностью оправдано.

1.3 Постановка задачи

Необходимо разработать специализированное мобильное приложение-мессенджер для устройств под управлением операционной системы Android, предназначенное для организации учебной коммуникации студентов. Выбор платформы обусловлен тем, что Android сохраняет позиции абсолютного лидера на мировом и отечественном рынках мобильных ОС, охватывая подавляющее большинство целевой студенческой аудитории.

В рамках разрабатываемого программного продукта должны быть реализованы следующие функции:

- регистрация и авторизация, с возможностью выбора факультета и группы;
- создание и редактирование личного профиля студента (установка аватара, изменение отображаемых данных);
- обмен текстовыми сообщениями в рамках личных и групповых бесед;
- прикрепление и передача файлов различного формата и объема, их просмотр и загрузка;
- настройка и получение push-уведомлений о новых сообщениях;

- поиск необходимой информации по истории переписки;
- интуитивно понятный пользовательский интерфейс с использованием Material Design 3.

Минимальные системные требования для клиента:

- операционная система Android 12.0;
- наличие интернета.

Минимальные системные требования для сервера:

- операционная система Linux Ubuntu Server 22.04;
- наличие стабильного интернет-соединения;
- поддержка до 1000 одновременно активных пользователей.

2 Выбор инструментов разработки

При разработке программного обеспечения и оптимизации работы необходимо выбрать наиболее эффективные средства разработки. Правильный выбор инструментов позволяет обеспечить масштабируемость мессенджера, безопасность передаваемых данных и сократить время на реализацию серверной и клиентской логики.

2.1 Выбор языка программирования

Для разработки мессенджера необходимо выбрать современный и производительный язык программирования, который обеспечит стабильную работу системы как на стороне мобильного клиента, так и на стороне сервера. В данном проекте в качестве основного технологического стека был выбран язык Kotlin.

2.1.1 Язык программирования «Kotlin»

Язык программирования Kotlin, разработанный компанией JetBrains, представляет собой современный статически типизированный инструмент, который с 2017 года признан компанией Google приоритетным для разработки под Android [3]. Его популярность в профессиональной среде обусловлена полной совместимостью с экосистемой Java, что позволяет использовать любые существующие библиотеки и фреймворки, одновременно предлагая более совершенные механизмы написания кода. Это означает возможность создания приложения, соответствующего современным индустриальным стандартам.

Одной из ключевых особенностей Kotlin является встроенная система безопасности типов, направленная на предотвращение ошибок, связанных с обращением к пустым ссылкам (Null Safety). Это важно, ведь в архитектуре мессенджера данные о пользователях, статусах доставки или содержанием чатов могут передаваться в неполном виде. Язык помогает явно обрабатывать

возможные «пустые» значения на этапе компиляции, что сводит к минимуму риск внезапного прекращения работы приложения на устройстве пользователя.

Не менее важным является наличие корутин. Это потоки, которые позволяют выполнять асинхронные операции — например, отправку сообщений, получение push-уведомлений или загрузку тяжелых файлов — без блокировки основного интерфейса приложения. Благодаря этому приложение сохраняет плавность и отзывчивость даже при активном обмене данными с сервером, что напрямую влияет на качество пользовательского опыта.

Кроме того, Kotlin демонстрирует высокую эффективность не только в мобильной разработке, но и на стороне серверной части при работе с фреймворком Spring Boot. Использование единого языка программирования для написания логики клиента и сервера позволяет унифицировать структуры данных. Это упрощает процесс передачи информации между компонентами системы, снижает вероятность ошибок при сериализации данных и значительно сокращает общее время, необходимое на написание и поддержку исходного кода всего проекта.

2.1.2 Вывод

На основании проведенного анализа для реализации проекта был выбран язык программирования Kotlin. Его использование позволяет создать современное, масштабируемое и безопасное приложение, минимизировав при этом затраты времени на написание и поддержку исходного кода.

2.2 Выбор фреймворка для разработки серверной части

Для реализации серверной части разрабатываемого приложения было принято решение использовать фреймворк Spring Boot [4]. Этот фреймворк имеет официальную и полную поддержку Kotlin, что позволяет применять единые подходы при написании как мобильного клиента, так и бэкенда, существенно экономя время на разработку приложения.

Одной из главных причин выбора этого фреймворка стала его направленность на быстрое развертывание рабочей среды. В отличие от других

серверных решений, требующих длительной ручной настройки, Spring Boot берет на себя большую часть базовых конфигураций благодаря механизму зависимостей-стартеров. Наличие встроенного веб-сервера позволяет запускать серверную часть как обычное приложение прямо из среды разработки, избавляя от необходимости настраивать стороннее программное обеспечение и давая возможность сосредоточиться исключительно на логике работы мессенджера.

Кроме того, экосистема Spring предоставляет мощные готовые модули, которые идеально закрывают все потребности разрабатываемого приложения. Для обеспечения работы чатов в реальном времени используется поддержка протокола WebSockets, гарантирующая доставку сообщений без задержек. Вопросы аутентификации пользователей и защиты их личных переписок легко реализуются с помощью модуля Spring Security, который будет использоваться для настройки входа по JWT-токенам. Так же интеграция со Spring Data JPA ускоряет процесс взаимодействия с базой данных MySQL, упрощая написание SQL-запросов.

2.3 Выбор среды разработки

Выбор программной среды напрямую влияет на скорость написания кода и стабильность итогового продукта. В данном проекте используется связка из двух сред разработки, основанных на платформе IntelliJ от компании JetBrains, что обеспечивает единообразие интерфейса и горячих клавиш при переключении между клиентской и серверной частями.

2.3.1 Среда разработки Android Studio

Для создания клиентской части приложения выбрана Android Studio. Эта IDE является официальной интегрированной средой разработки и создана компанией Google специально для работы с платформой Android [5]. Она обладает глубокой интеграцией с Android SDK, что позволяет оперативно получать обновления библиотек и инструментов сразу после их выхода.

Основные возможности Android Studio, использованные в проекте:

					ПГУ1.090301.37.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

– Система сборки Gradle. Автоматизирует процесс компиляции приложения, управления зависимостями и подготовки итогового файла для установки на смартфон;

– Layout Editor и Compose Preview. Инструменты визуального проектирования интерфейса, которые позволяют видеть изменения в дизайне приложения в реальном времени без необходимости каждый раз запускать его на устройстве;

– Средства профилирования. Встроенные мониторы ресурсов помогают отслеживать потребление памяти и сетевой трафик, что существенно упрощает оптимизацию мессенджера для работы на устройствах с ограниченными ресурсами;

– Встроенный эмулятор. Позволяет тестировать работу чатов и уведомлений на различных конфигурациях экранов и версиях Android непосредственно на компьютере разработчика.

Интерфейс приложения представлен на рисунке 9.

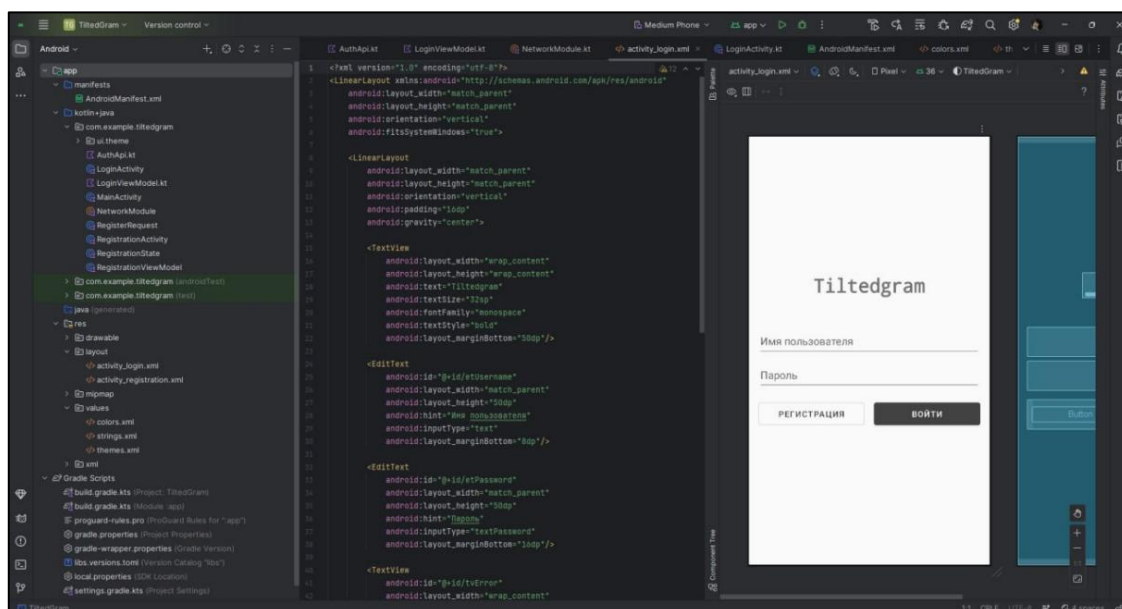


Рисунок 9 – Среда разработки Android Studio

2.3.2 Среда разработки IntelliJ IDEA

Для реализации серверной части на базе фреймворка Spring Boot выбрана IntelliJ IDEA [6]. Поскольку и язык Kotlin, и сама среда разработки созданы одной компанией, IDEA предлагает наиболее полную поддержку синтаксиса и инструментов рефакторинга для этого языка.

Особенности IntelliJ IDEA для разработки серверной части:

- Поддержка Spring Framework. Среда автоматически распознает компоненты Spring, подсвечивает ошибки в конфигурации и предлагает удобную навигацию между контроллерами и сервисами бэкенда.

- Инструменты работы с базами данных. Встроенный клиент позволяет просматривать структуру таблиц MySQL, выполнять SQL-запросы и проверять сохраненные сообщения прямо из окна IDE.

- Интеграция с инструментами тестирования. Удобная среда для написания и запуска Unit-тестов, необходимых для проверки корректности логики авторизации и обработки данных.

Использование этих двух сред в рамках одного проекта позволяет унифицировать рабочий процесс. Разработчик может использовать одинаковые инструменты поиска по коду, системы контроля версий Git и плагины, что сокращает время на освоение инструментов и позволяет сосредоточиться на реализации логики мессенджера.

Интерфейс приложения представлен на рисунке 10.

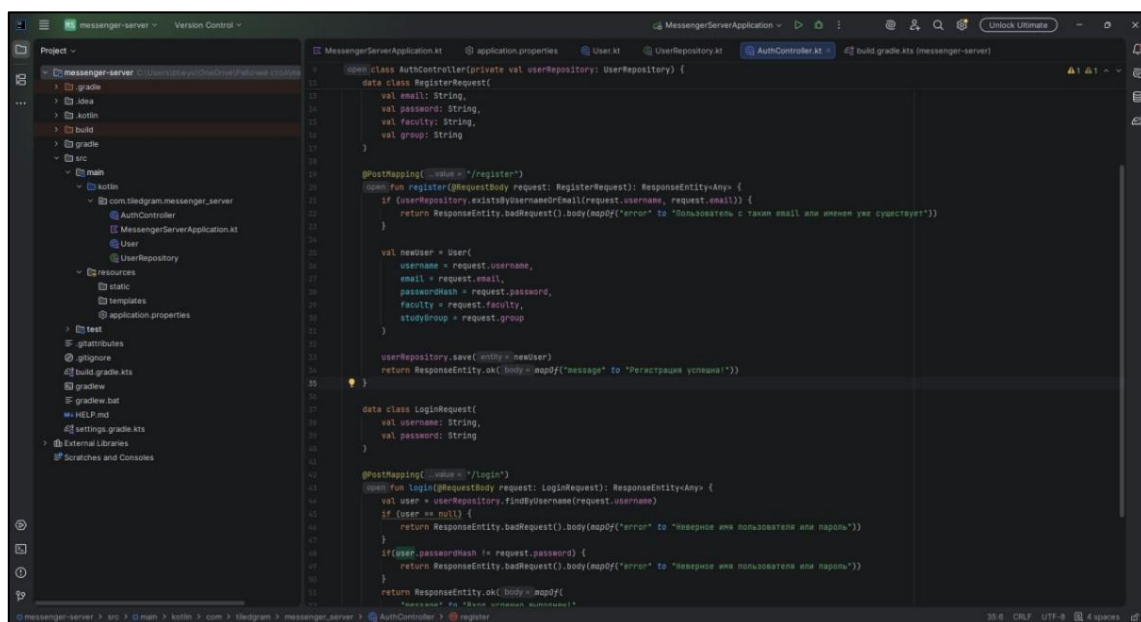


Рисунок 10 – Среда разработки IntelliJ IDEA

2.4 Выбор системы хранения данных

Для хранения данных пользователей выбрана реляционная база данных MySQL [7]. В ней будет храниться как структурированная информация, так и данные, связанные с мультимедийным контентом. Для реализации отправки документов, архивов и прочих вложений в таблицы базы данных предусмотрены специальные поля, хранящие локальные пути и ссылки на файлы. Такой подход позволяет связать текстовые сообщения и вложения на уровне реляционных связей внутри одной системы.

Выбор MySQL обусловлен её высокой скоростью запросов, целостности данных, а также возможностью взаимодействия с фреймворком Spring Boot посредством технологии Spring Data JPA. Использование единой СУБД упрощает процедуру резервного копирования информации и обеспечивает стабильную работу серверной части при множественном одновременном обращении пользователей.

2.5 Настройка среды разработки

2.5.1 Настройка среды для разработки клиентской части

Для разработки мобильного приложения использовалась интегрированная среда разработки Android Studio — официальный инструмент для создания приложений под операционную систему Android, поддерживаемый компанией Google.

После установки среды разработки был создан новый проект мобильного приложения на языке программирования Kotlin. В ходе создания проекта автоматически формировалась базовая структура, содержащая каталоги для исходного кода, файлов пользовательского интерфейса и конфигурационных данных. Этапы создания проекта представлены на рисунках 11–12.

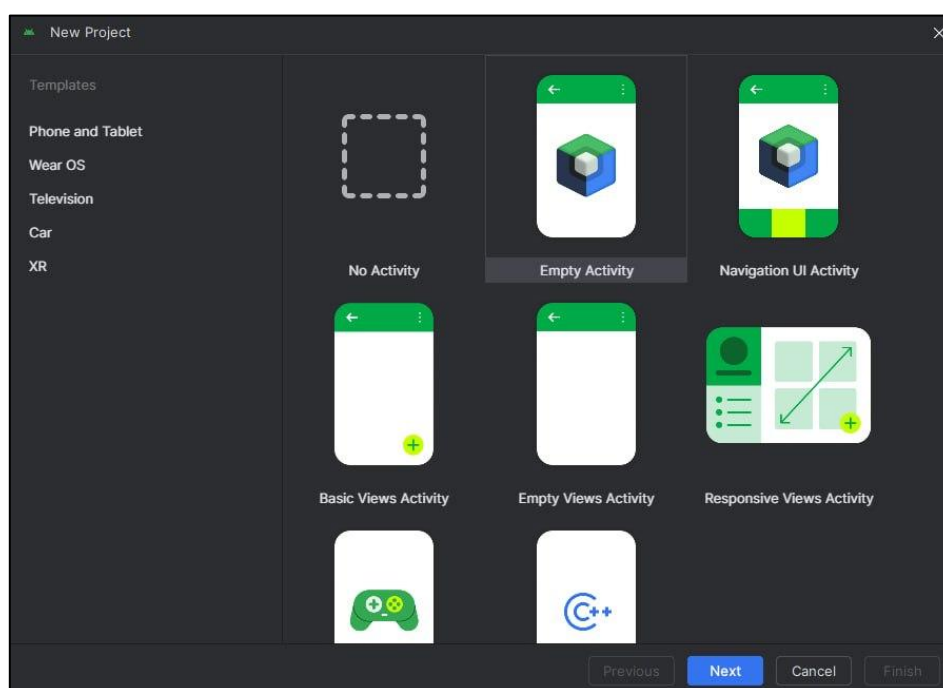


Рисунок 11 – Создание проекта для мобильного приложения

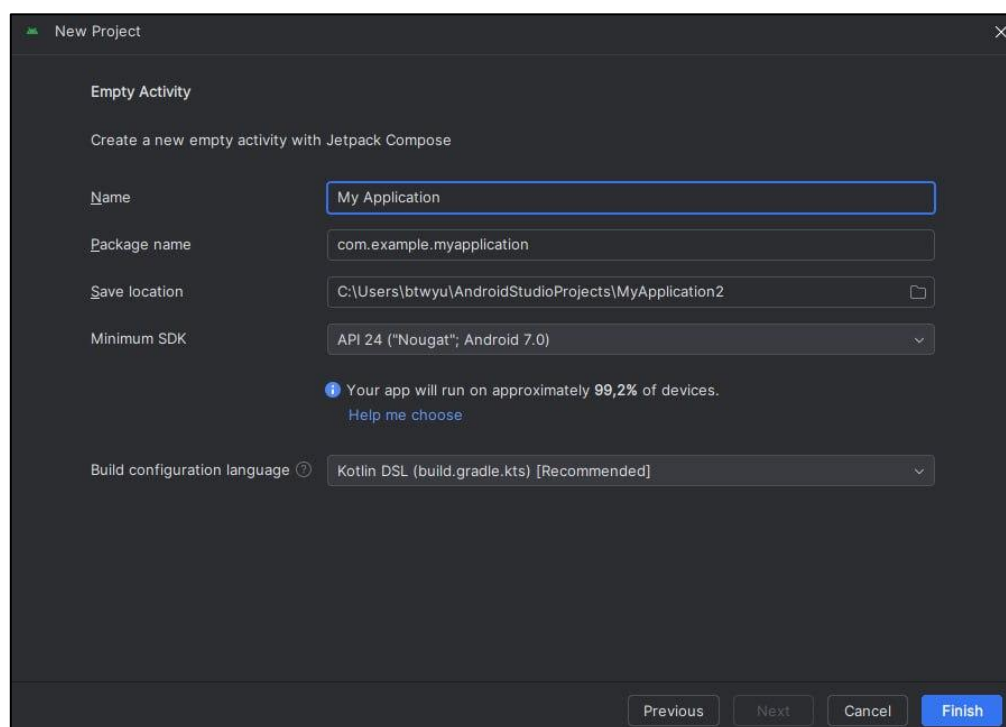


Рисунок 12 – Выбор параметров нового проекта

Среда разработки самостоятельно устанавливает все необходимые компоненты Android SDK, в состав которых входят библиотеки платформы Android, инструменты сборки и средства тестирования приложений.

Для управления зависимостями используется система сборки Gradle. Она обеспечивает подключение требуемых библиотек, контроль версий компонентов и автоматизацию процесса сборки приложения.

2.5.2 Настройка среды для разработки серверной части

Разработка серверной части приложения выполнялась с использованием интегрированной среды IntelliJ IDEA, созданной компанией JetBrains. Дополнительно был установлен комплект разработчика Java Development Kit версии 21, выступающий базовой платформой для выполнения Kotlin-кода в среде виртуальной машины Java.

В среде разработки был создан новый проект на основе фреймворка Spring Boot. Первичная настройка проекта осуществлялась с помощью инструмента Spring Initializer. В ходе конфигурации были подключены основные зависимости,

включающие Spring Web для реализации REST-интерфейса, Spring Data JPA для упрощения взаимодействия с реляционными базами данных, а также Spring Security для обеспечения безопасности сессий и хранения пользовательских данных.

Для управления зависимостями проекта также применяется система сборки Gradle, которая автоматически подключает необходимые библиотеки и компоненты используемого фреймворка.

					ПГУ1.090301.37.001 ПЗ	Лист
						28
Изм.	Лист	№ докум.	Подпись	Дата		

3 Архитектура приложения

3.1 Архитектурный паттерн клиентской части «MVVM»

В качестве основного паттерна проектирования мобильной части мессенджера был выбран MVVM (Model-View-ViewModel) [8]. Данный подход официально рекомендуется компанией Google для разработки гибких и масштабируемых интерфейсов. Архитектура MVVM, как видно из названия, разделяет код на три независимых компонента: Model (Модель), View (Представление), ViewModel (Модель представления).

Первый компонент отвечает за логику работы с данными мессенджера. Модель управляет сетевыми запросами к серверу, которые реализованы с помощью библиотеки Retrofit, загрузкой файлов и локальным хранением информации. Модель полностью изолирована от пользовательского интерфейса. Это означает, что ей неважно как именно будет выглядеть отрисованное сообщение, задача Model вовремя и правильно получить это сообщение из сети или базы данных.

Следующий компонент – View. Это слой пользовательского интерфейса, включающий в себя экраны приложения. Задача этого слоя — отрисовка элементов на экране и регистрация действий пользователя. Он подписывается на изменения данных в ViewModel, благодаря этому автоматически обновляет интерфейс при поступлении новых данных.

Последний компонент нужен для связи между Model и View. ViewModel запрашивает необходимые данные у модели, подготавливает их для отображения и содержит в себе логику их обработки. Особенностью ViewModel является способность сохранять состояние интерфейса и данные даже при системных прерываниях, например при смене ориентации экрана.

3.2 Архитектура серверной части «Layered Architecture»

Серверная часть приложения, написанная с использованием фреймворка Spring Boot, построена на базе классического паттерна многоуровневой архитектуры (Layered/N-Tier Architecture). Этот подход подразумевает строгое разделение бэкенда на логические слои, где каждый слой имеет свою зону ответственности и взаимодействует преимущественно с нижележащим уровнем:

Слой контроллеров (Controllers Layer) является точкой входа для всех запросов от Android-клиента. Он отвечает за прием HTTP запросов и соединение с сокетом, валидацию входящих данных и маршрутизацию. На этом же этапе происходит взаимодействие с фильтрами безопасности для проверки подлинности токенов авторизации.

Центральный компонент серверной архитектуры, слой сервисов (Service Layer), в котором инкапсулирована основная логика приложения. Здесь выполняются проверки принадлежности пользователя к определенным факультетам или группам, логика распределения сообщений между чатами и формирование ответов.

Слой доступа к данным (Repository Layer). Отвечает за взаимодействие с базой данных MySQL. За счет использования абстракций Spring Data JPA, сервер взаимодействует с базой данных через объекты и интерфейсы, что обеспечивает безопасное чтение и запись истории переписок, а также возможность быстрой замены хранилища при необходимости.

Также архитектурной особенностью разработанной серверной части является её функционирование в рамках одноузловой конфигурации. В системе преднамеренно отсутствуют механизмы динамического горизонтального масштабирования и балансировки нагрузки.

Это означает, что при достижении или превышении установленного лимита одновременных активных пользователей, серверная часть не будет автоматически создавать свои реплики для распределения трафика и принудительного

увеличения лимита участников. Такое ограничение расширяемости является обоснованным рамками для локальной образовательной среды, где пиковые нагрузки предсказуемы и ограничены размерами учебного заведения.

					ПГУ1.090301.37.001 ПЗ	Лист
						31
Изм.	Лист	№ докум.	Подпись	Дата		

4 Реализация приложения

Для перехода к этапу реализации приложения требуется описать основные сценарии взаимодействия пользователя. Поэтому необходимо детально рассмотреть функциональные требования к разрабатываемому программному обеспечению.

4.1 Описание функционала

В архитектуре студенческого мессенджера заложена обязательная система аутентификации, обеспечивающая изолированность учебной среды. Для получения доступа к приложению каждый студент должен пройти этап регистрации с обязательным указанием своего факультета и учебной группы, либо выполнить вход в уже существующую учетную запись. Только после успешной авторизации пользователю открывается полный набор предусмотренных возможностей приложения, включающий в себя:

- создание и редактирование личного профиля;
- создание индивидуальных диалогов;
- создание групп и их редактирование;
- обмен мгновенными текстовыми сообщениями;
- прикрепление, отправка и скачивание мультимедийных файлов и учебных документов;
- осуществление поиска необходимой информации по истории переписки;

Визуальное представление описанных сценариев взаимодействия авторизованного и неавторизованного пользователя с системой отражено на диаграмме вариантов использования на рисунке 13.

Диаграмма вариантов использования

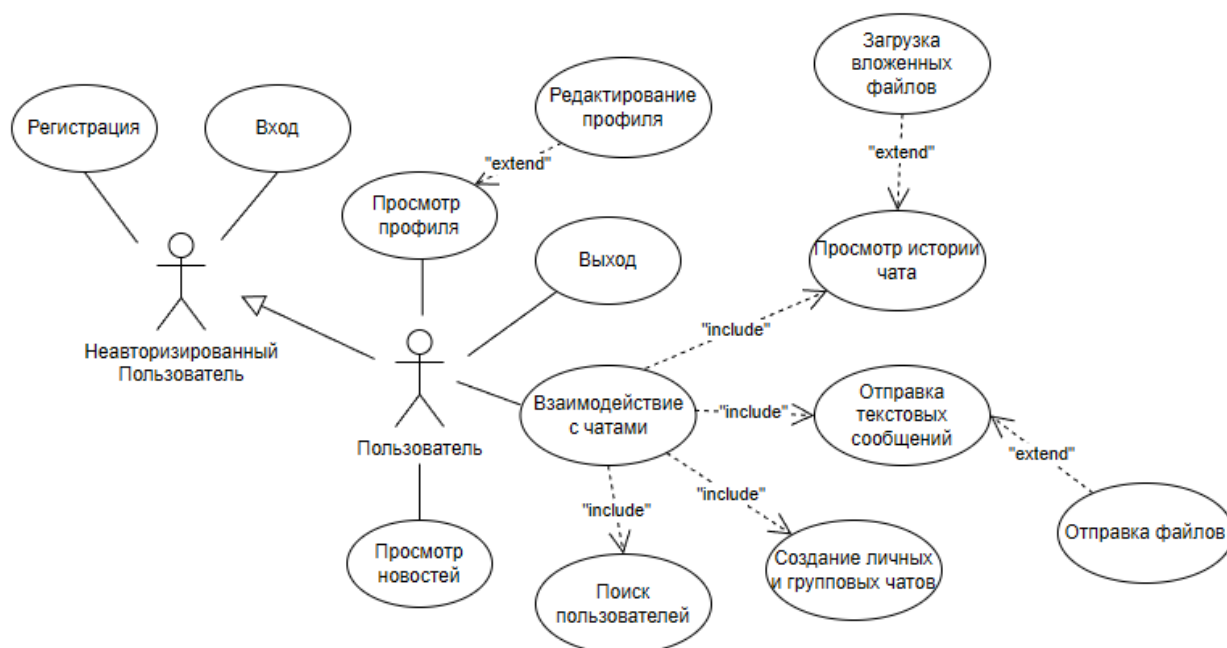


Рисунок 73 - UML диаграмма вариантов использования

Для наглядности на рисунках 14-16 показаны варианты взаимодействия пользователя с функциями приложения.

После запуска приложения пользователем клиент проверяет, авторизовался ли пользователь до этого. Если пользователь был успешно авторизован до этого, то приложение сразу перекинет его на главный экран. В противном случае откроется экран авторизации с двумя вариантами: ввести данные для входа и войти или зарегистрироваться (рисунок 14).

После входа пользователь попадает на главный экран (рисунок 15), он может воспользоваться всеми доступными функциями. Изначально у пользователя загружается экран последних переписок, который обновляется в случае появления новых сообщений. По желанию пользователь может нажать на иконку своего профиля для взаимодействия с ним.

Если пользователь выбирает конкретный чат, то перед ним открывается следующий функционал: просмотр и отправка сообщений, возможность скачать

файл или же открыть профиль собеседника (рисунок 16). В случае, если открыта группа, пользователь так же может посмотреть список участников или выйти из неё. Являясь создателем, можно так же редактировать профиль группы и выгонять участников.

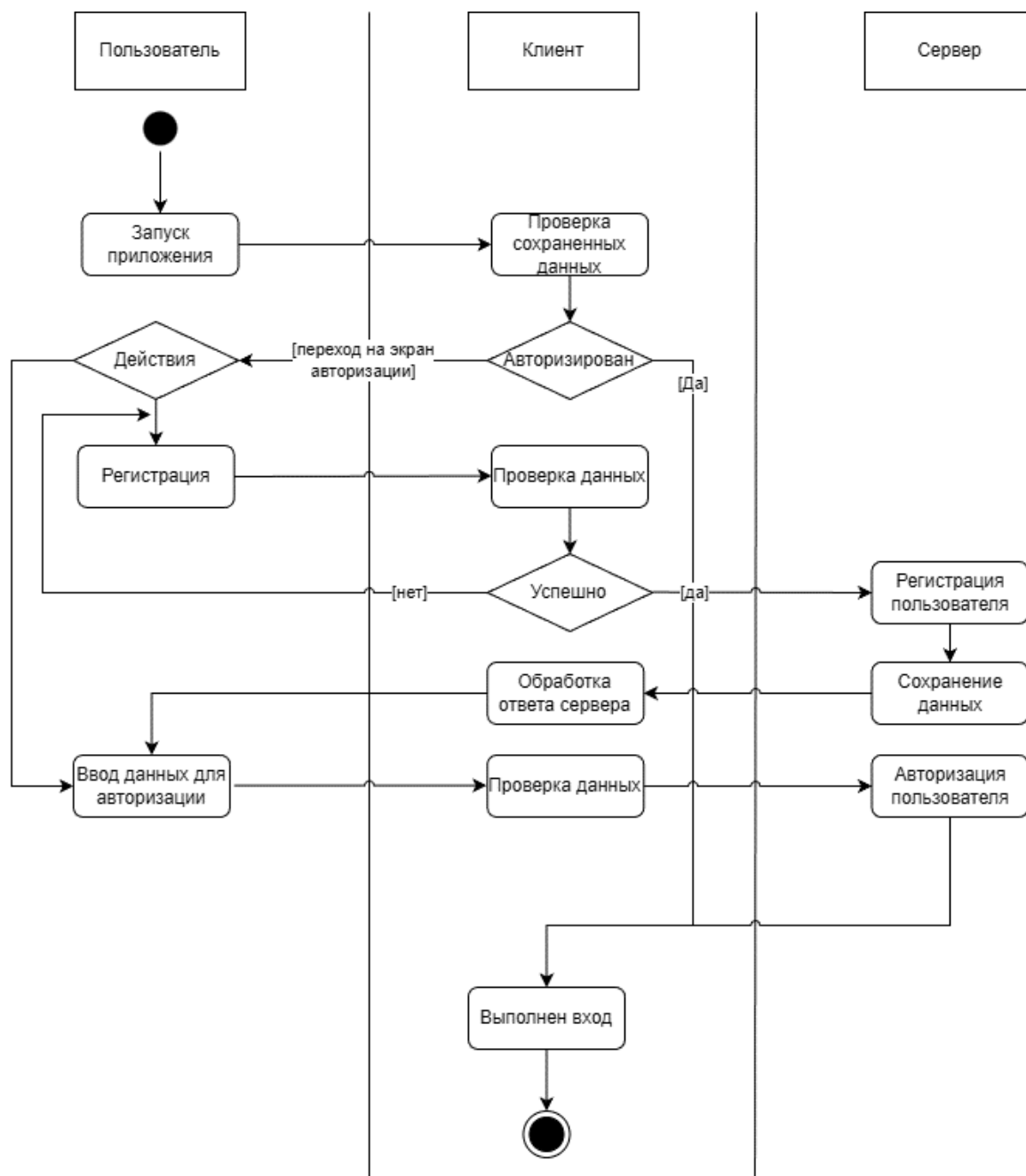


Рисунок 14 - UML диаграмма деятельности для регистрации/авторизации в приложении

Диаграмма деятельности главного экрана

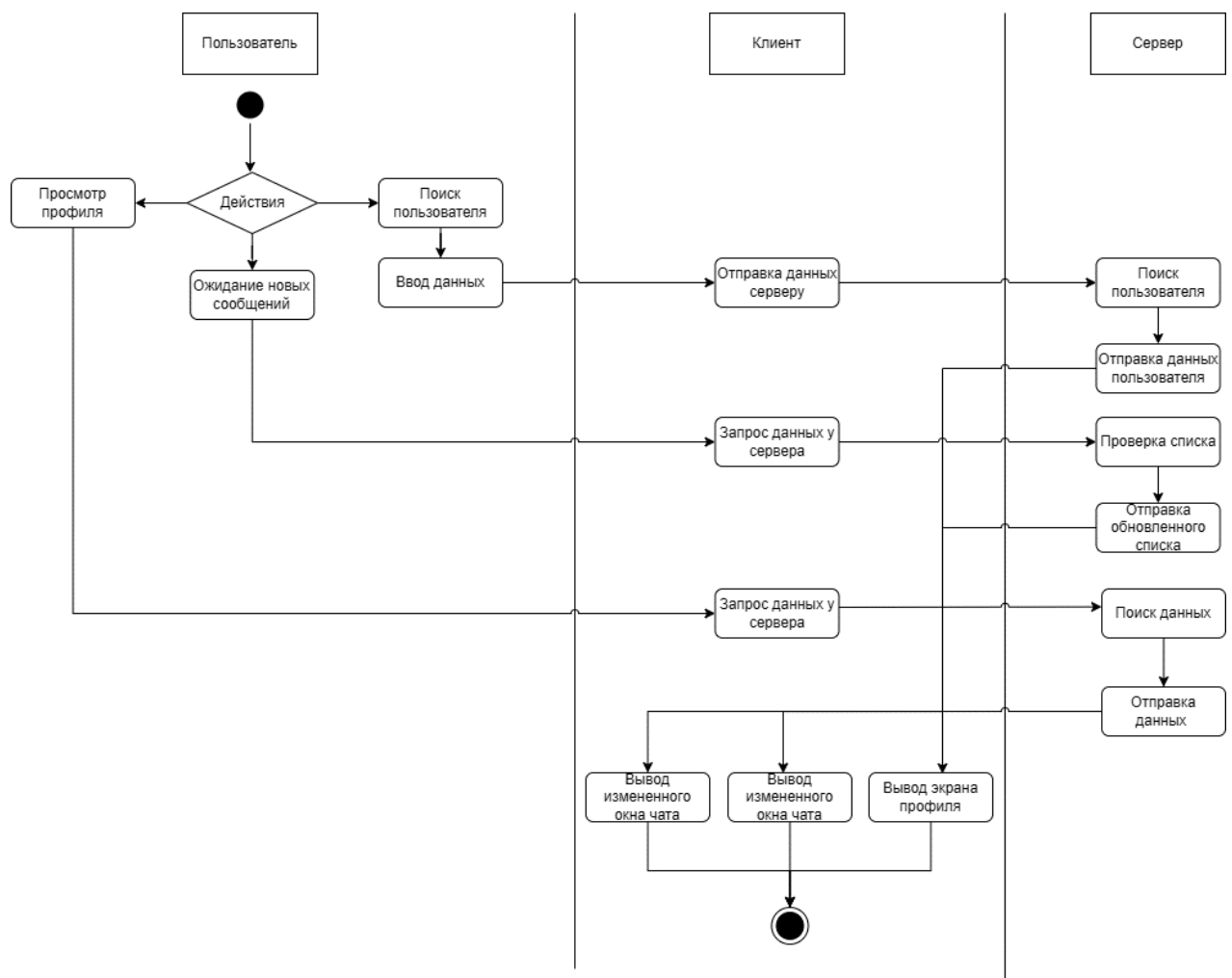


Рисунок 15 - UML диаграмма деятельности для главного экрана приложения

Диаграмма деятельности взаимодействия внутри чата

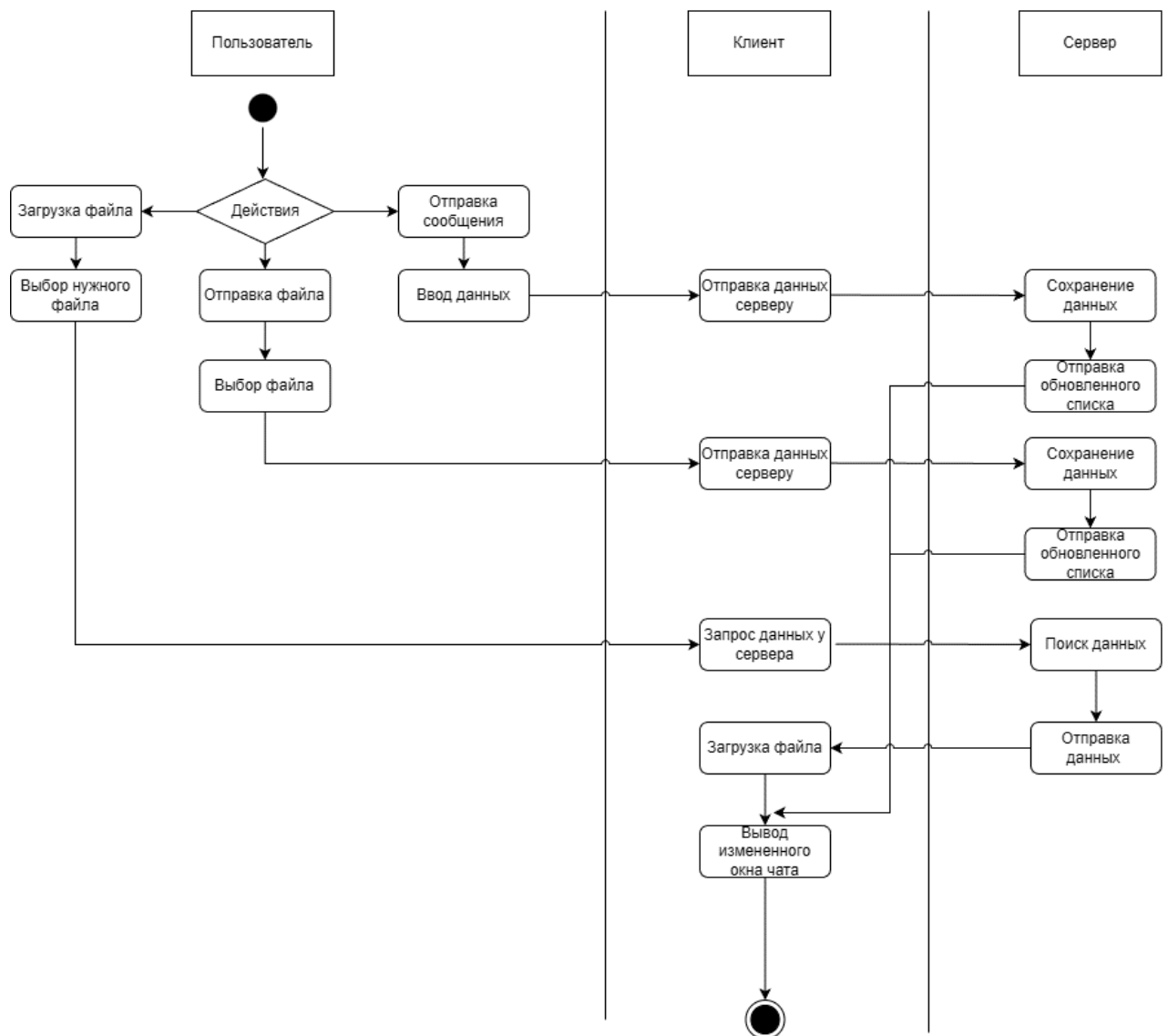


Рисунок 16 - UML диаграмма деятельности для окна переписки

4.2 Описание работы приложения

Взаимодействие пользователя с приложением происходит посредством клиентской части. Пользователю необходимо зайти в приложение на устройстве, которое будет соответствовать запрашиваемым минимальным требованиям. После запуска пользователю требуется авторизоваться или зарегистрироваться, в зависимости от наличия аккаунта, для дальнейшего взаимодействия с приложением. В этот момент клиент производит сравнение с данными сервера, после чего, в случае успеха, происходит вход на главную страницу приложения. Теперь пользователь может полностью использовать доступный ему функционал главной страницы.

Так, если пользователь решит найти новый контакт, то клиент сделает запрос списка всех пользователей на сервер, после чего отправит его обратно. Клиент, в свою очередь, будет фильтровать полученный список, в зависимости от введенных пользователем данных. Если же пользователь захочет написать кому-то сообщение, то клиент сделает запрос о данных получателя и истории их переписки, а после получения данных с сервера отрисует окно чата, в котором пользователь сможет просмотреть его историю переписки, скачать и посмотреть находящиеся в ней файлы, а также отправлять сообщения и файлы. Полный список действий, доступный пользователю представлен на диаграмме последовательности (рисунок 17), физическая модель расположения компонентов же на рисунке 18.

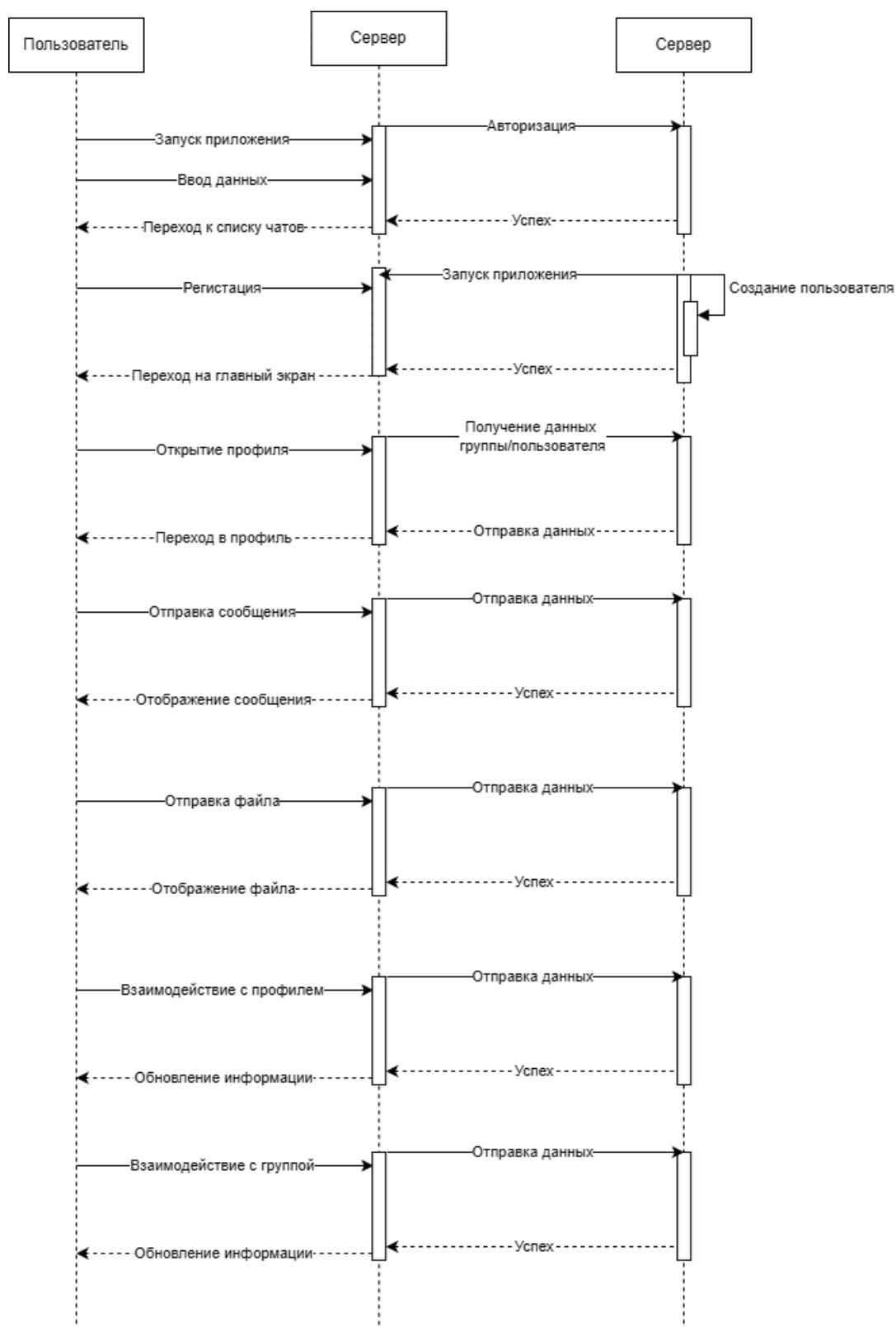


Рисунок 17 - UML диаграмма последовательности

Диаграмма развертывания

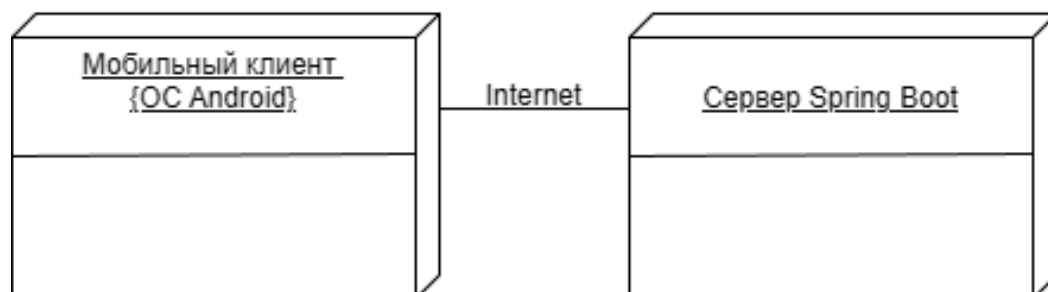


Рисунок 18 - UML диаграмма развертывания

4.3 Основная реализация приложения

Поскольку основным паттерном клиентской части был выбран паттерн MVVM, исходный код так же разделен на три основных компонента:

- Model;
- View;
- ViewModel

Данное разделение позволяет отделить логику работы пользовательского интерфейса от доменной логики и сетевых запросов. Чтобы посмотреть реализацию возьмем в пример процесс авторизации пользователя.

4.3.1 Слой представления (View)

В слое представления находятся классы, отвечающие за отображение и обработку пользовательского интерфейса. В нашем примере запроса авторизации эту роль играют классы LoginActivity и RegistrationActivity. Они отрисовывает интерфейс, обрабатывает данные для авторизации, вводимые пользователем, а так же содержит внутри себя объект класса LoginViewModel и RegistrationViewModel соответственно.

В приложении используется обработка состояний с использованием StateFlow и корутин. LoginActivity подписывается на изменения состояния и автоматически обновляет интерфейс или выполняет переход на главный экран.

При успешной авторизации в SharedPreferences сохраняются данные входа пользователя. Это необходимо для того, чтобы пользователь не вводил свои данные при каждом повторном использовании приложения.

Листинг 1. Обработка состояния в классе LoginActivity:

```
lifecycleScope.launch {
    repeatOnLifecycle(Lifecycle.State.STARTED) {
        viewModel.state.collect { state ->

            binding.btnLogin.isEnabled = !state.isLoading
            if (state.errorMessage != null) {
                Toast.makeText(
                    this@LoginActivity,
                    state.errorMessage,
                    Toast.LENGTH_SHORT).show()
            }

            if (state.isLoginSuccessful) {
                val sharedPrefs =
                    getSharedPreferences("MessengerPrefs", MODE_PRIVATE)
                sharedPrefs.edit()
                    .putLong("USER_ID", viewModel.loggedInUserId)
                    .putString("USER_NAME", viewModel.loggedInUsername)
                    .putString("USER_FIO", viewModel.loggedInFullName)
                    .putString("USER_EMAIL", viewModel.loggedInEmail)
                    .putString("USER_FACULTY",
                        viewModel.loggedInFaculty)
                    .putString("USER_GROUP", viewModel.loggedInGroup)
                    .putString("USER_AVATAR",
                        viewModel.loggedInAvatarUrl)
                    .apply()
                Toast.makeText(this@LoginActivity, "Добро
                    пожаловать!", Toast.LENGTH_SHORT).show()
                goToMainActivity()
            }
        }
    }
}
```

4.3.2 Слой модели представления (ViewModel)

Данный слой выступает в роли связующего звена между пользовательским интерфейсом и данными. Класс `LoginViewModel` хранит текущее состояние экрана в специальном классе данных `LoginState`

Листинг 2. Класс данных `LoginState`:

```
data class LoginState(  
    val username: String = "",  
    val password: String = "",  
    val isLoading: Boolean = false,  
    val errorMessage: String? = null,  
    val isLoginSuccessful: Boolean = false  
)
```

При нажатии на кнопку входа вызывается логика авторизации. Внутри `LoginViewModel` запускается асинхронная корутина, отправляющая запрос на сервер, при этом не блокируя основной поток приложения.

Листинг 3. Выполнение сетевого запроса в `LoginViewModel`:

```
viewModelScope.launch {  
    _state.update { it.copy(isLoading = true, errorMessage = null)  
}  
    try {  
        val request = LoginRequest(currentState.username,  
currentState.password)  
        val response = NetworkModule.authApi.login(request)  
  
        if (response.isSuccessful){  
            val body = response.body()  
            if (body != null){  
                loggedInUserId = body.userId  
                loggedInUsername = body.username  
                loggedInEmail = body.email  
                loggedInFaculty = body.faculty  
                loggedInGroup = body.studyGroup  
                loggedInAvatarUrl = body.avatarUrl  
                _state.update { it.copy(isLoading = false,  
isLoginSuccessful = true) }  
            }  
        } else {  
            _state.update {  
                it.copy(isLoading = false, errorMessage = "Неверное имя  
пользователя или пароль")  
            }  
        }  
    } catch (e: Exception){
```

```

        _state.update { it.copy(isLoading = false, errorMessage =
"Ошибка сети") }
    }
}

```

Помимо обработки асинхронных запросов, здесь реализован модуль валидации вводимых пользователем данных. С помощью регулярных выражений приложение проверяет данные до момента отправки на сервер.

Листинг 3. Валидация данных в RegistrationViewModel:

```

private val nameRegex = "[A-Za-zA-Яa-яЁё\\s]+\\$".toRegex()
private val usernameRegex = "[a-zA-Z0-9_-]+\\$".toRegex()

fun register() {
    if (!nameRegex.matches(currentState.fullName)) {
        _state.update { it.copy(errorMessage = "ФИО должно
содержать только буквы") }
        return
    }
    // запуск корутины для отправки данных
}

```

4.3.3 Слой данных (Model)

Слой данных берет на себя сетевое взаимодействие и работу с внешними структурами. Для взаимодействия с сервером на базе Spring Boot используется REST API, а библиотека Retrofit в свою очередь используется для сериализации данных, а так же для выполнения HTTP запросов. Не менее важным компонентом является NetworkModule, реализованный как объект типа синглтон и настраивающий HTTP клиент.

Листинг 2. Настройка сетевого клиента:

```

object NetworkModule {
    private const val BASE_URL = "http://192.168.1.63:8080"
    private val loggingInterceptor =
HttpLoggingInterceptor().apply {
        level = HttpLoggingInterceptor.Level.HEADERS
    }

    private val client = OkHttpClient.Builder()
        .addInterceptor(loggingInterceptor)
        .connectTimeout(60, TimeUnit.SECONDS)
}

```

```

        .readTimeout(120, TimeUnit.SECONDS)
        .writeTimeout(300, TimeUnit.SECONDS)
        .build()
private val retrofit = Retrofit.Builder()
    .baseUrl(BASE_URL)
    .client(client)
    .addConverterFactory(GsonConverterFactory.create())
    .build()

val authApi: AuthApi = retrofit.create(AuthApi::class.java)
}

```

Не менее важным компонентом выступает интерфейс `AuthApi`, в котором находятся специальные объекты передачи и точки взаимодействия с сервером. В нашем случае для авторизации используются `RegisterRequest` и `LoginResponse`, которые нужны для типизации данных при обмене информацией с сервером. Это предотвращает ошибки при превращении JSON ответов сервера в Kotlin объекты

Листинг 3. Интерфейс `AuthApi` и модели данных:

```

data class RegisterRequest(
    val username: String,
    val fullName: String,
    val email: String,
    val password: String,
    val faculty: String,
    val group: String
)

data class LoginResponse(
    val message: String,
    val userId: Long,
    val username: String,
    val fullName: String,
    val email: String,
    val faculty: String,
    val studyGroup: String,
    val avatarUrl: String?
)

interface AuthApi {
    @POST("/api/auth/register")
    suspend fun register(@Body request: RegisterRequest):
Response<Any>
    @POST("/api/auth/login")
    suspend fun login(@Body request: LoginRequest):
Response<LoginResponse>
}

```

Такая архитектура облегчает масштабирование и тестирование приложения при добавлении новых функций.

5 Тестирование приложения

Для полноценной обеспечения корректной работы приложения нужно выполнить ряд тестов, для проверки работоспособности его функций. Этот этап очень важен, ведь он напрямую влияет на опыт использования, а также помогает выделить недочеты и ошибки в коде.

Для тестов рассмотрим следующие сценарии:

- регистрация и авторизация;
- просмотр и редактирование профиля;
- отправка сообщений, файлов и их загрузка;
- создание групп и их редактирование;
- поиск сообщений.

5.1 Регистрация

При первом запуске приложения у пользователя нет своего аккаунта. Для того чтобы это исправить нужно нажать соответствующую кнопку, после чего приложение перекинет его на экран регистрации, в которой попросят заполнить все поля, как показано на рисунке 19.

Рисунок 19 – Окно регистрации

Рисунки 20-23 показывают варианты того, что будет если пользователь будет вводить некорректные данные

Рисунок 20 – Неверный формат имени

←

Создать аккаунт

Влад Корнилов

Влад Корнилов

почта)

Выберите факультет

ФВТ

Выберите группу

22ВВП2

Никнейм должен содержать только английские буквы, цифры, '-' и '_'

Зарегистрироваться

Рисунок 21 – Неверный формат никнейма

←

Создать аккаунт

Юля Корнилова

yulyazxc

aaa

Выберите факультет

ФВТ

Выберите группу

22ВВП2

Этот Никнейм или Email уже занят!

Зарегистрироваться

Рисунок 22 – Ошибка имени

Рисунок 23 – Некорректный адрес почты

После того, как пользователь ввел нужные данные и нажал на кнопку регистрации, на сервер сразу идет запрос создания аккаунта, для сохранения данных пользователя (рисунок 24). Далее пользователь переходит обратно на экран авторизации. В случае попытки нажать кнопку войти, не заполнив все поля, пользователь увидит предупреждение, что видно на рисунке 25

id	avatar_url	email	faculty	full_name	password_hash	study_group	username	fc_m_token
1	http://192.1...	yuki.btw@hotmail.com	ФВТ	Влад Корнилов	yukicxz	22ВВП2	yukicxz	NULL
2	NULL	sam_sa@ya.ru	ФВТ	Sasha Samofalova	samsa04	22ВВП2	samsa04	NULL
4	http://192.1...	yulya@cxz.com	ФВТ	Юля Корнилова	666	22ВВП2	yulyazxc	NULL
5	http://192.1...	vanya59@ya.ru	ФВТ	Иванов Иван	111	22ВВП2	vanya	NULL
6	NULL	sa@ya.ru	ФВТ	Sasha Samofalova	\$2a\$10\$K2MYc...	22ВВП2	sam_sa04	fpwgZt86QHyp2kK2YD
7	http://2.26....	yuki@cxz.com	ФВТ	Влад Корнилов	\$2a\$10\$bsrmU...	22ВВП2	666yuki	ecZjyQH5Tii4uISnfUZw
8	http://2.26....	damn48623@gmail.com	ФВТ	Жирнс Люблюфонк	\$2a\$10\$DAbdk...	22ВВП2	upsSEX	fKn5w_q3RFmDbzLJR0
9	http://193.6...	samsa@gmail.com	ФВТ	Самса Самселева	\$2a\$10\$eGQ2...	22ВВП2	samsa_04	NULL
10	http://2.26....	aleksavs2004@mail.ru	ФВТ	Самса Самс Самсовна	\$2a\$10\$PRjPR...	22ВВП2	samsa__04	fbL6lYJaSvaSy63lxheX

Рисунок 24 – Регистрация пользователя в базе данных

Рисунок 25 – Ошибка входа

Затем после авторизации пользователь попадает на главный экран мессенджера (рисунок 26). Новые чаты на экране появятся только после того, как пользователю кто-то написал или наоборот пользователь кому-то напишет. Так же пользователю сразу доступна беседа группы (рисунок 27), в которую он автоматически попадает после регистрации.

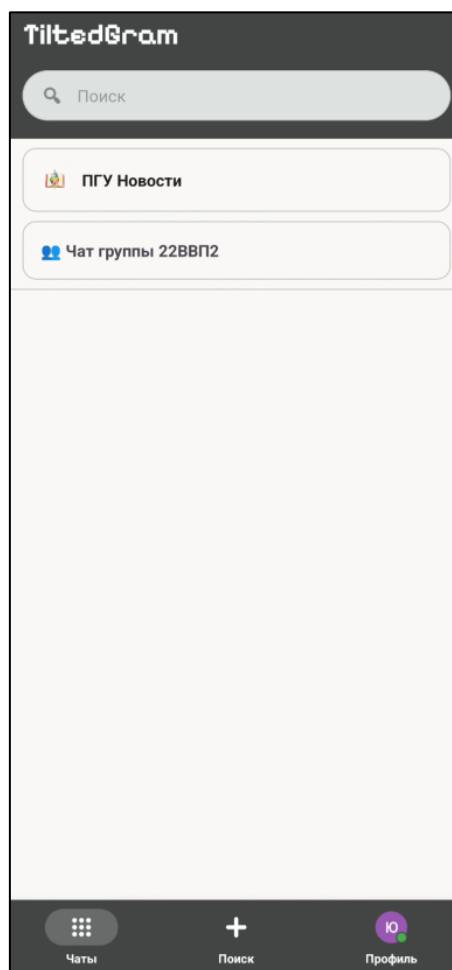


Рисунок 26 – Главное меню нового пользователя

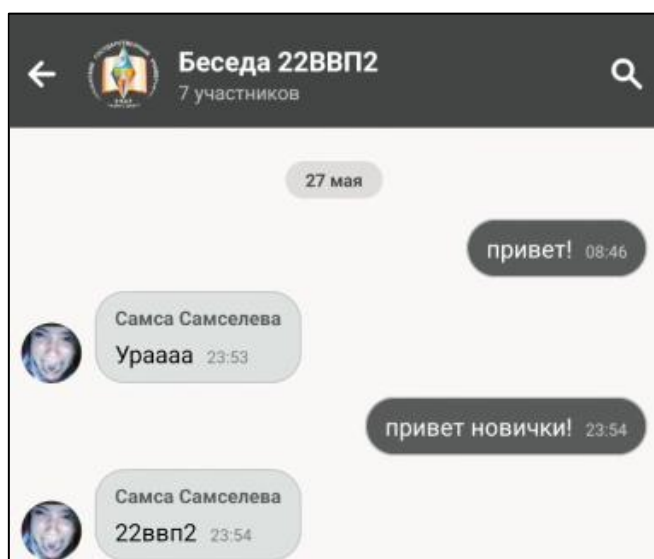


Рисунок 27 – Беседа группы

Процесс тестирования наглядно представлен в таблице 2.

Таблица 2 – Тестирования регистрации и авторизации

№	Сценарий и шаги тестирования	Ожидаемый результат	API	Статус
1	Успешная регистрация 1. Открыть окно регистрации 2. Ввести валидные данные 3. Нажать «Зарегистрироваться»	После успешного ответа вывод уведомления об успехе и возврат	Отправка POST запроса на /api/auth/register Ожидаемый ответ: HTTP код 200 OK	Успешно
2	Ошибки валидации при регистрации 1. Ввести недопустимые символы 2. Ввести занятый логин	Вывод ошибки	При локальных ошибках запрос не отправляется. При занятом логине ожидаемый ответ: 400 Bad Request	Успешно
3	Успешная авторизация 1. Ввести нужные данные 2. Нажать кнопку «Войти»	Сохранение данных пользователя в SharedPreferences, переход на главный экран	Отправка POST запроса на /api/auth/login Ожидаемый ответ: HTTP 200 OK и объект LoginResponse	Успешно

5.2 Просмотр и редактирование профиля

Если пользователь захочет изменить свой профиль, то ему нужно нажать на свой аватар в нижнем тулбаре, который по умолчанию отсутствует. После перехода на экран профиля открывается возможность редактирования контактной информации, а также смена аватара. Затем, когда необходимые изменения будут внесены, пользователь увидит свой обновленный профиль. Данный процесс показан на рисунках 28-29.

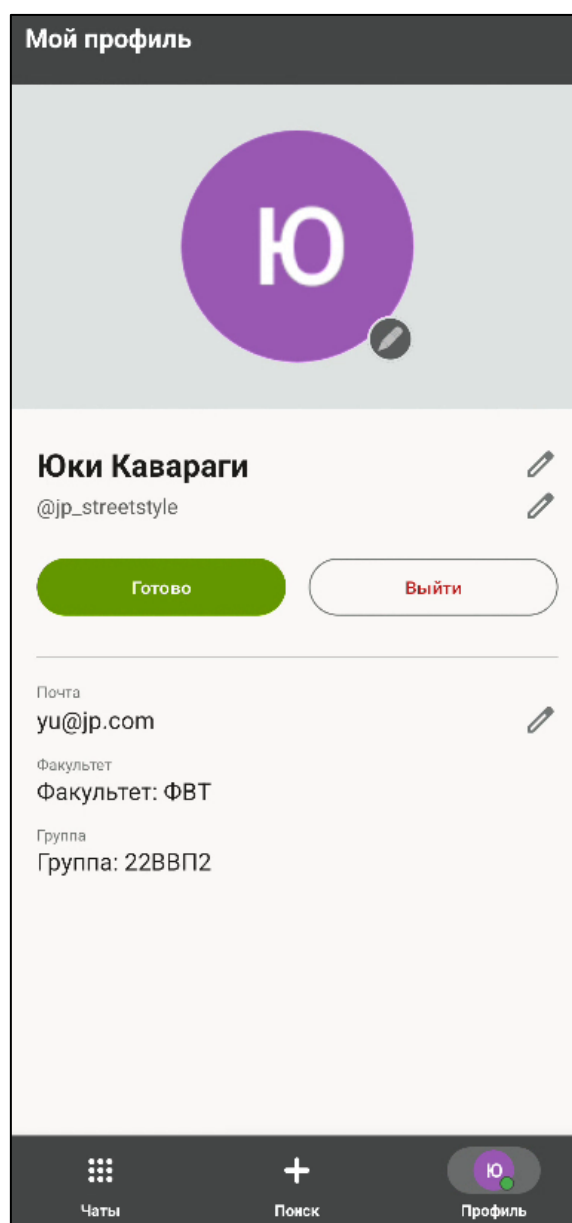


Рисунок 28 – Профиль пользователя

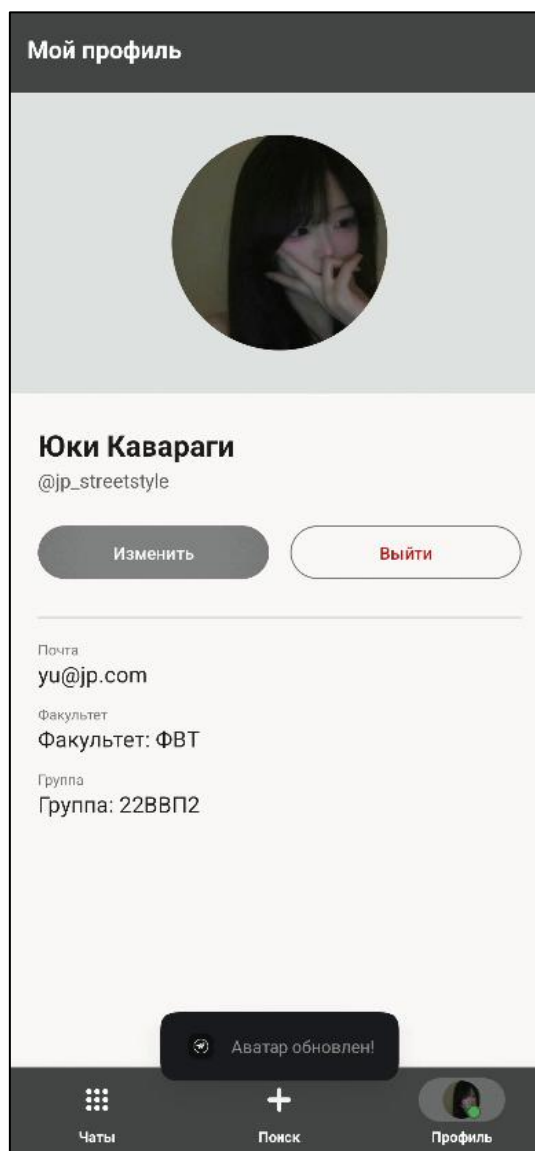


Рисунок 29 – Обновленный профиль

5.3 Отправка сообщений

Для отправки сообщений пользователю сначала нужно найти получателя, после чего он сможет ему написать. При переходе на экран поиска пользователю предлагается листать список всех пользователей, либо воспользоваться функцией поиска и ввести имя или никнейм получателя (рисунки 30-31).

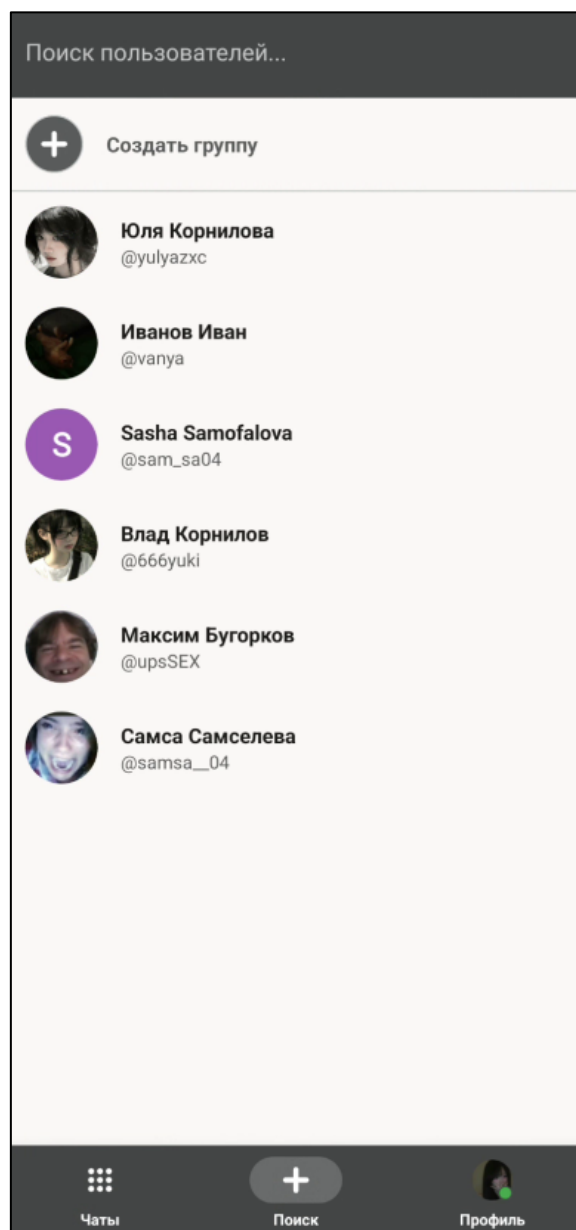


Рисунок 30 – Поиск пользователя

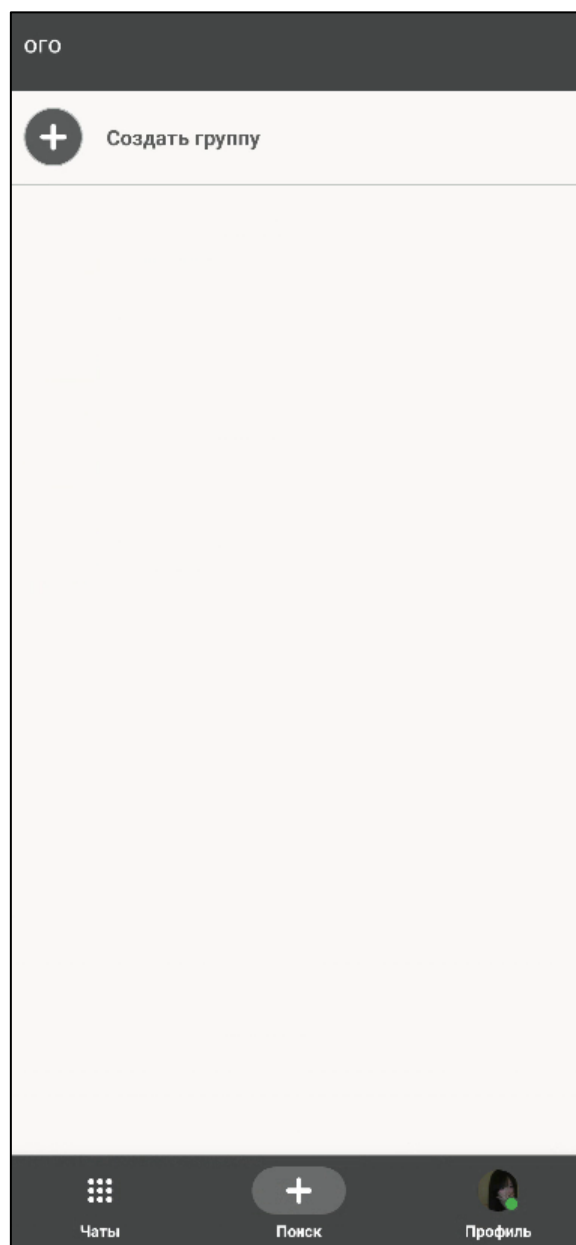


Рисунок 31 – Поиск несуществующего пользователя

Как только он найдет нужного получателя произойдет переход на экран чата, где пользователи смогу обмениваться сообщениями и файлами, как показано на рисунках 32-34.

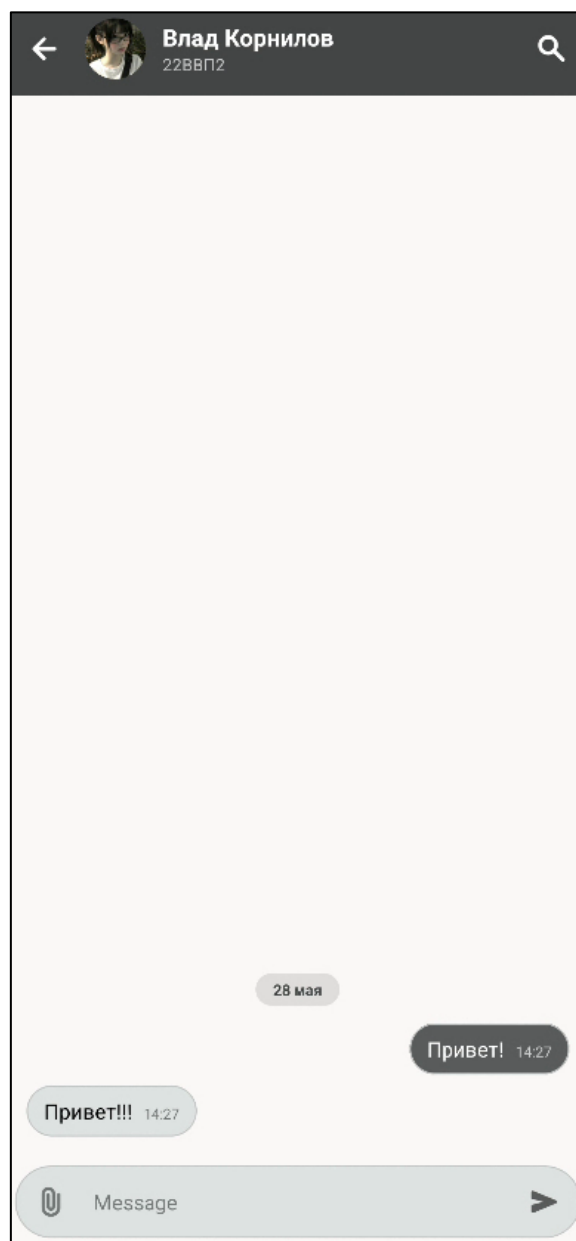


Рисунок 32 – Переписка с пользователем

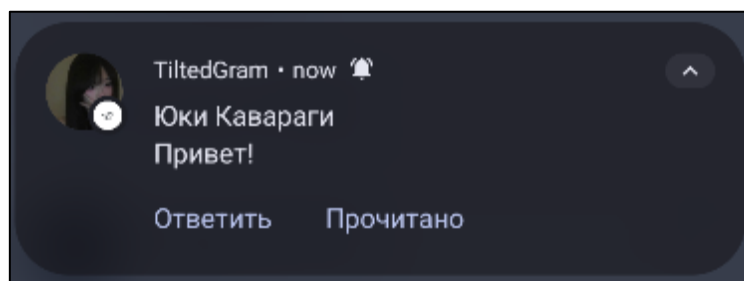


Рисунок 33 – Уведомление о новом сообщении

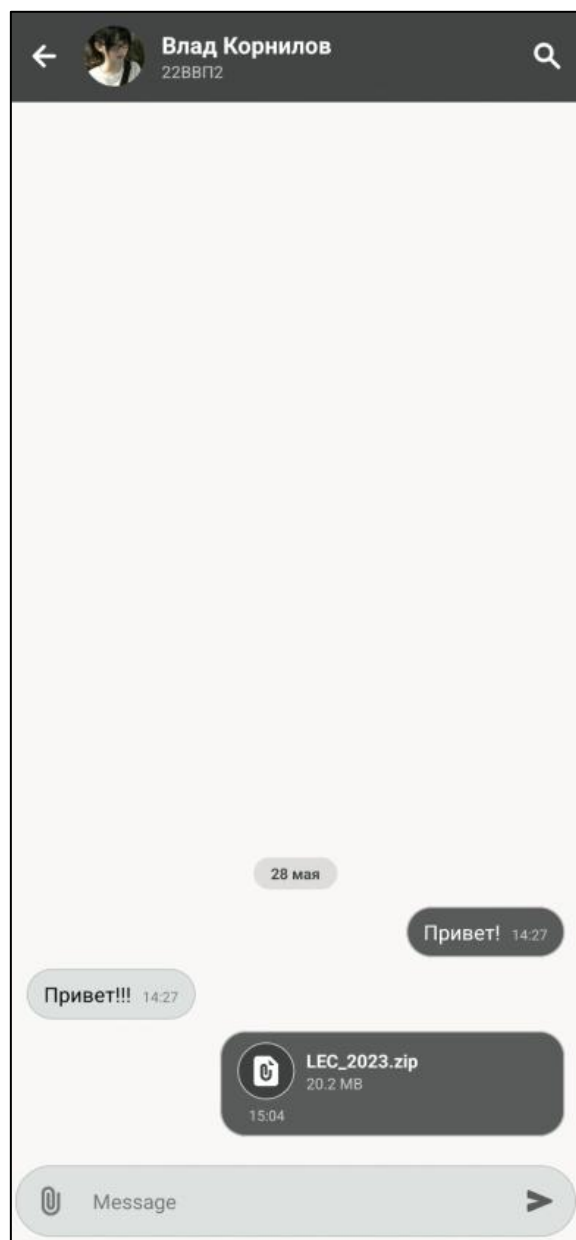


Рисунок 34 – Отправка файла

Если пользователь захочет открыть полученный файл, то при нажатии ему нужно будет подтвердить загрузку, после чего открыть файл. Процесс загрузки отображается рядом с весом файла и в шторке уведомлений (рисунки 35-36).

В случае если он хочет скачать изображение, то для этого нужно удерживать нужное изображение и принять загрузку. Обычный же клик просто откроет изображение (рисунок 37).

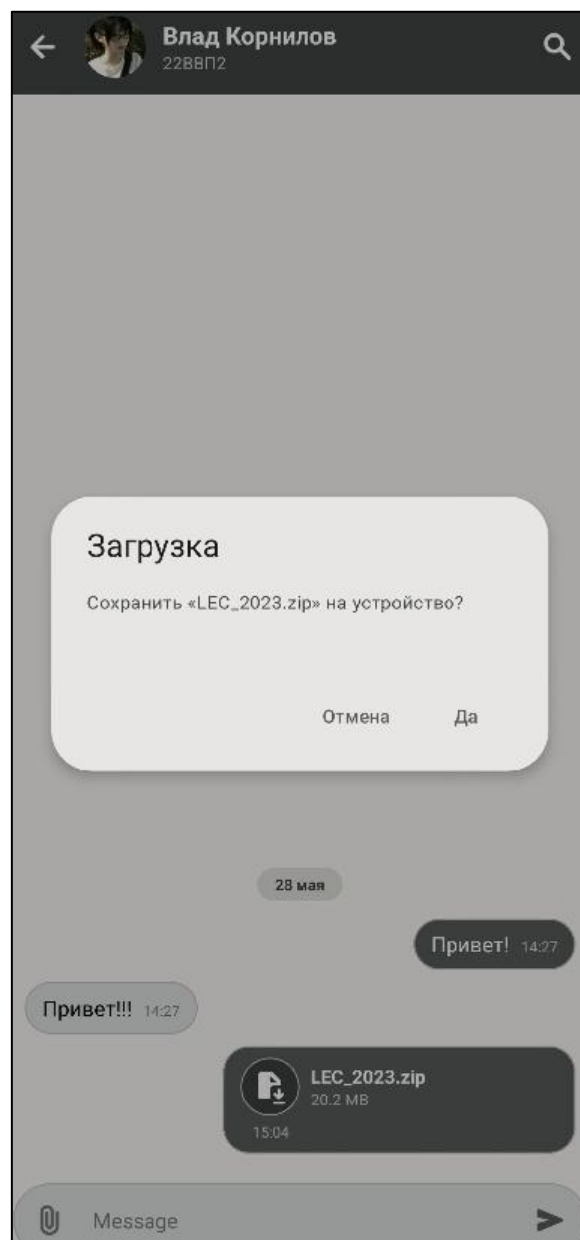


Рисунок 35 – Соглашение на загрузку файла

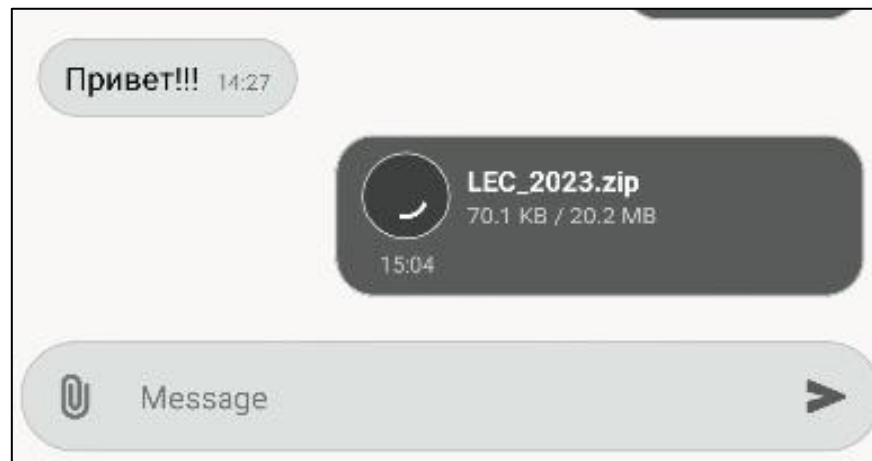


Рисунок 36 – Загрузка файла

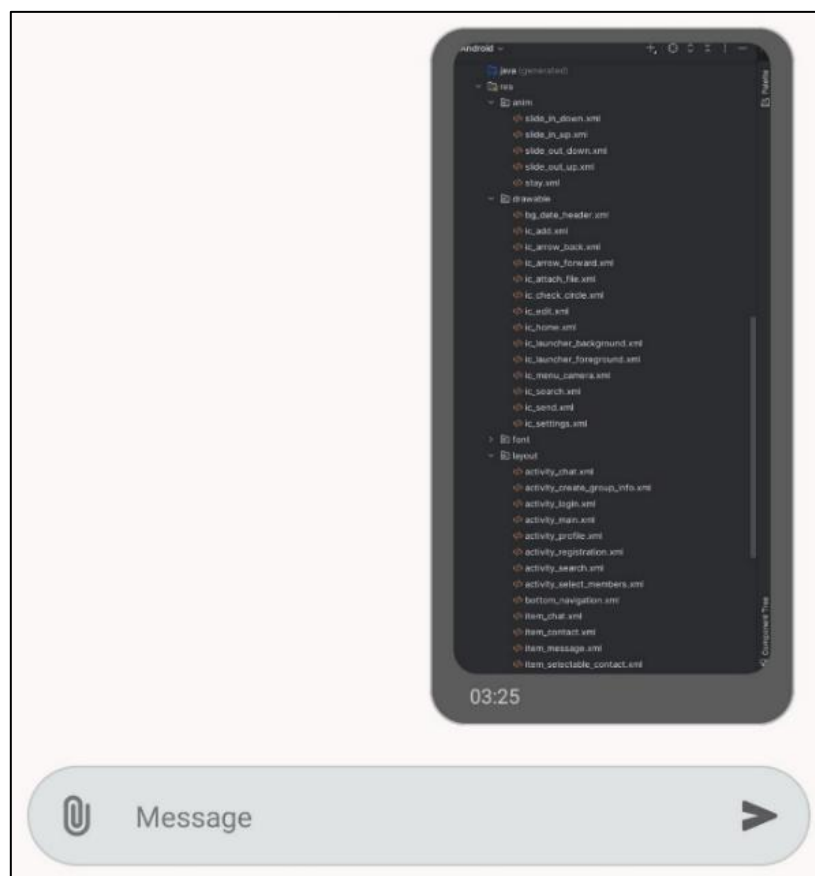


Рисунок 37 – Изображение в чате

Подробное тестирование данного блока показано в таблице 3.

Таблица 3 – Тестирования поиска и отправки сообщений

№	Сценарий и шаги тестирования	Ожидаемый результат	API	Статус
1	Поиск пользователей 1. Перейти на экран поиска 2. Ввести имя существующего пользователя	Динамическое обновление списка на основе полученных данных	Отправка GET запроса. Сервер фильтрует базу и возвращает массив объектов ChatUserResponse	Успешно
2	Отправка сообщений и файлов 1. Открыть чат 2. Отправить текст или прикрепить файл	Сообщение выводится сразу, загрузка файла сопровождается уведомлением с процессом загрузки файла	Отправка SendMessageRequest на сервер. Файл отправляется через Multipart запрос	Успешно
3	Загрузка вложений 1. Нажать на полученный файл или удерживать на картинке 2. Выбрать пункт скачать 3. Подтвердить загрузку	Появление системного уведомления о скачивании и отображение процесса загрузки, открытие файла сторонним приложением.	Отправка GET запроса по URL файла для получения байтового потока	Успешно

5.4 Создание и редактирование групп

Чтобы создать групповой чат следует нажать на соответствующую кнопку на экране поиска получателя. После чего необходимо выбрать участников группы из списка контактов (людей, с которыми уже есть диалог) и заполнить необходимые для группы поля. Создание нового группового чата показано на рисунках 38-40.

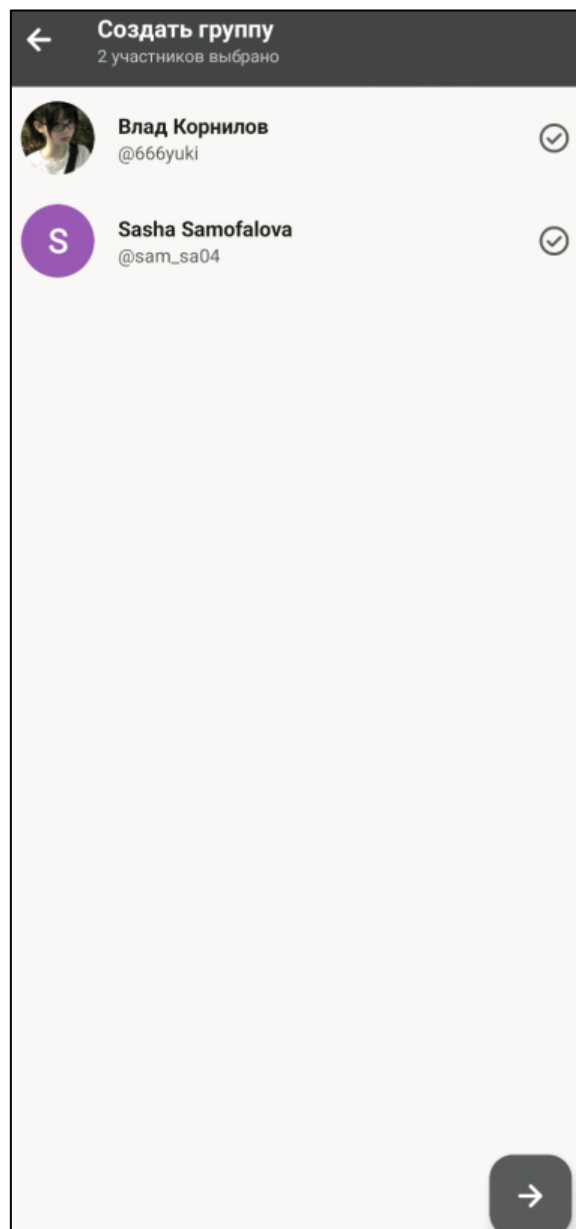


Рисунок 38 – Выбор участников группы

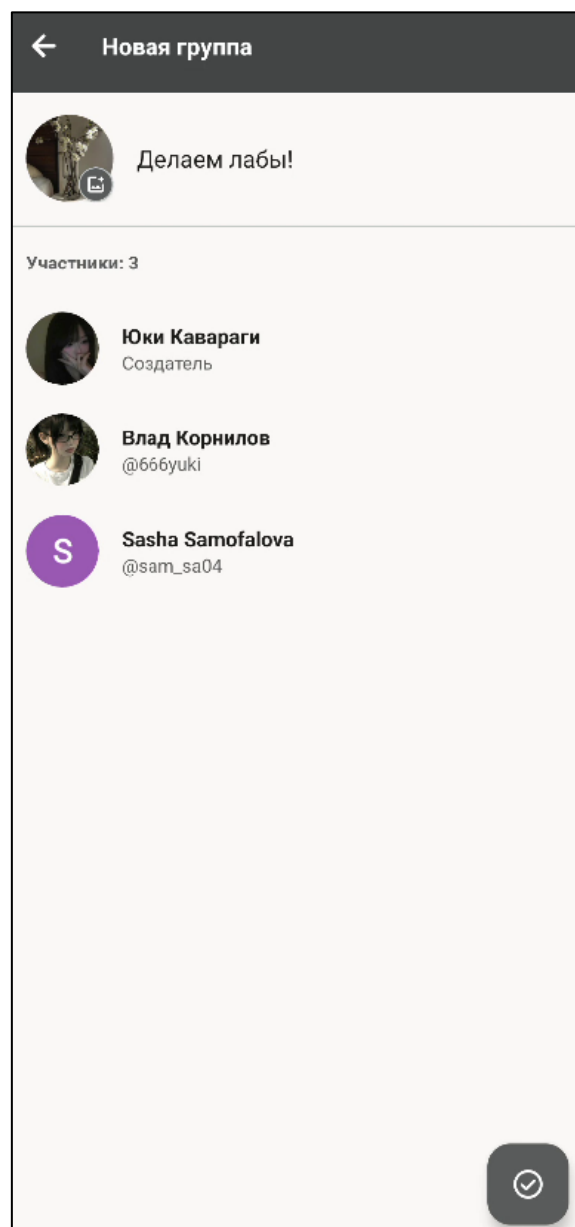


Рисунок 39 – Экран создания новой группы

					ПГУ1.090301.37.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

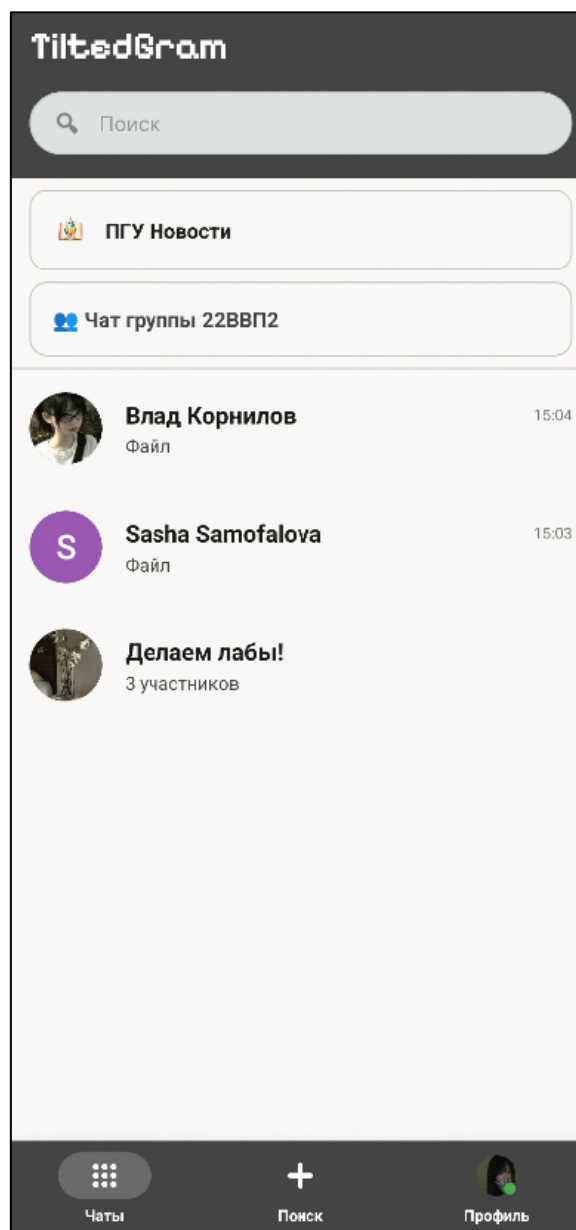


Рисунок 40 – Успешное создание группы

Функционал групп идентичен тому, что есть в личных переписках, за исключением возможности редактирования группы, открытие которой происходит при нажатии пользователем на шапку группы и выбора кнопки редактировать (рисунки 41-42)

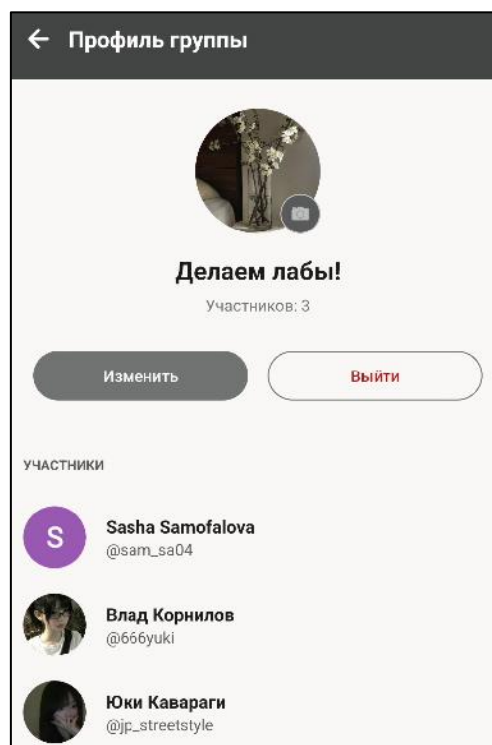


Рисунок 41 – Просмотр группы в качестве создателя

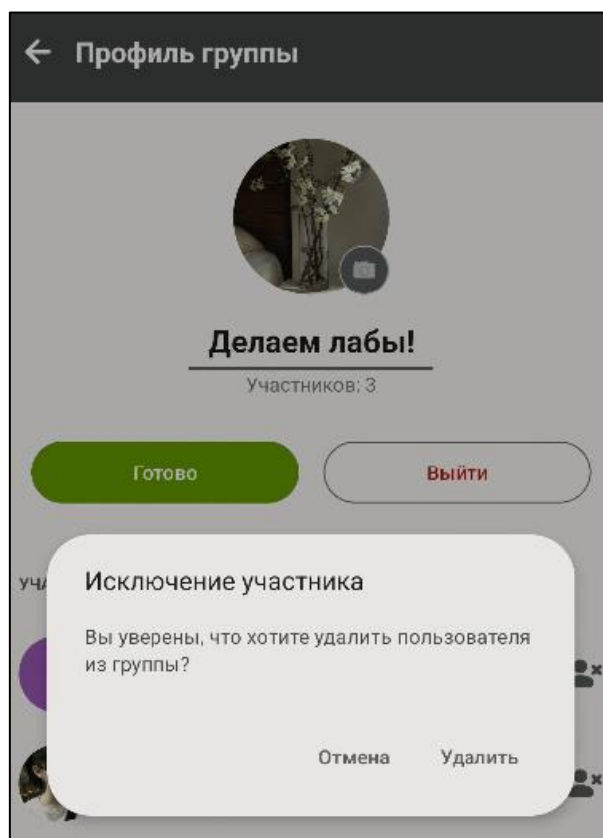


Рисунок 42 – Исключение участника из группы

Если же пользователь не является создателем группы, то функции редактирования ему недоступны, а при исключении из группы самого пользователя приложение вернется на главный экран с соответствующим оповещением (рисунок 43), а сама группа пропадет из списка активных чатов.

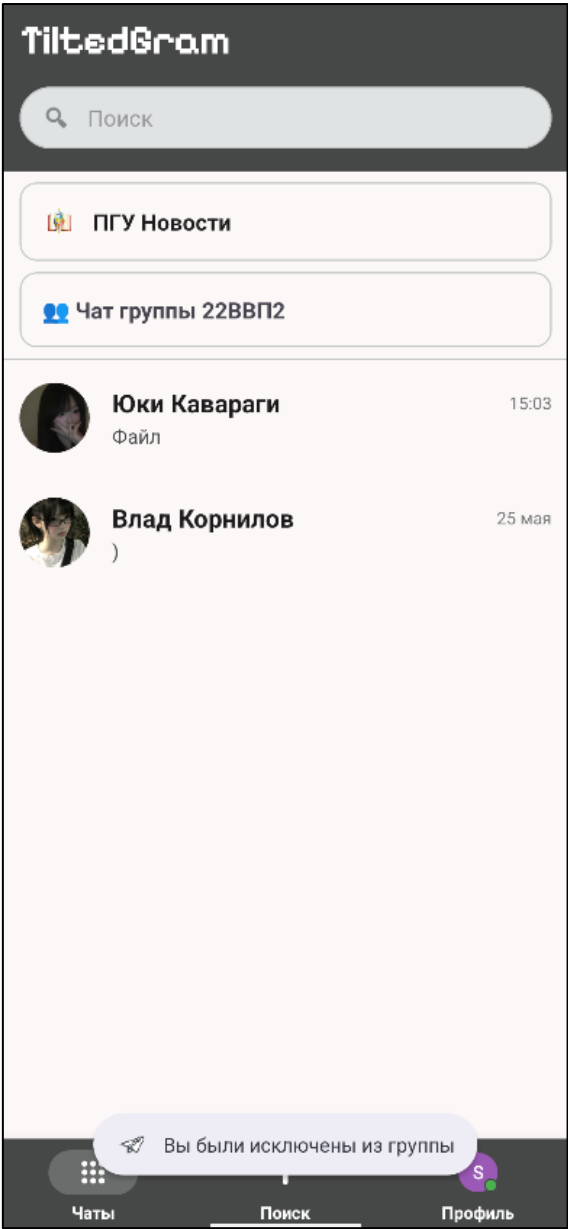


Рисунок 43 – Исключение из группы

Наглядно процесс тестирования групп приведен в таблице 4.

Таблица 4 – Тестирования создания и редактирования групп

№	Сценарий и шаги тестирования	Ожидаемый результат	API	Статус
1	Создание группы 1. Нажать кнопку создания 2. Выбрать участников 3. Ввести название	Появление новой группы в списке диалогов	Отправка POST запроса с массивом ID участников. Сервер создает новую сущность и связывает пользователей	Успешно
2	Управление группой 1. Зайти в настройки группы 2. Удалить пользователя	Обновление списка участников	Отправка DELETE запроса на /api/auth/groups/{groupId}/kick с KickMemberRequest. Ожидаемый ответ: HTTP 200 OK	Успешно
3	Исключение из группы 1. Зайти в приложение 2. Дождаться исключения создателем	Автоматический выход из чата группы на главный экран с уведомлением. Группа пропадает из списка	Сервер рассылает событие об удалении, а клиент не получает эту группу при обновлении диалогов	Успешно

5.5 Поиск сообщений

Для повышения удобства функция поиска сообщений разделена на два уровня: глобальный поиск на главном экране, который ищет нужное по всем диалогам, и локальный поиск внутри конкретного чата.

Глобальный поиск позволяет находить сообщения по всей базе данных текущего пользователя. Для осуществления данного поиска пользователю нужно воспользоваться поисковой строкой на главном экране. При вводе ключевого слова приложение фильтрует список диалогов, оставляя только те, где нашлись сходства. Когда пользователь найдет нужный диалог и откроет его, нужное сообщение выделится на несколько секунд. Этот процесс продемонстрирован на рисунках 44-45

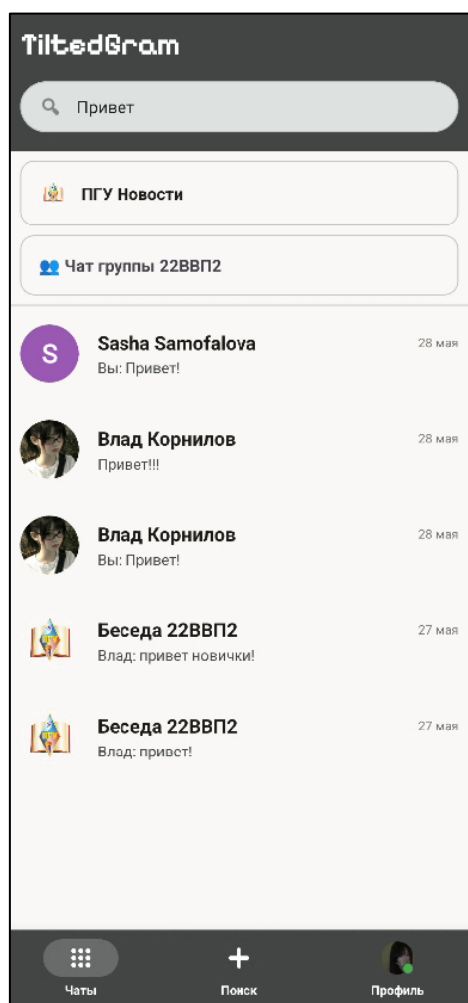


Рисунок 44 – Поиск на главном экране

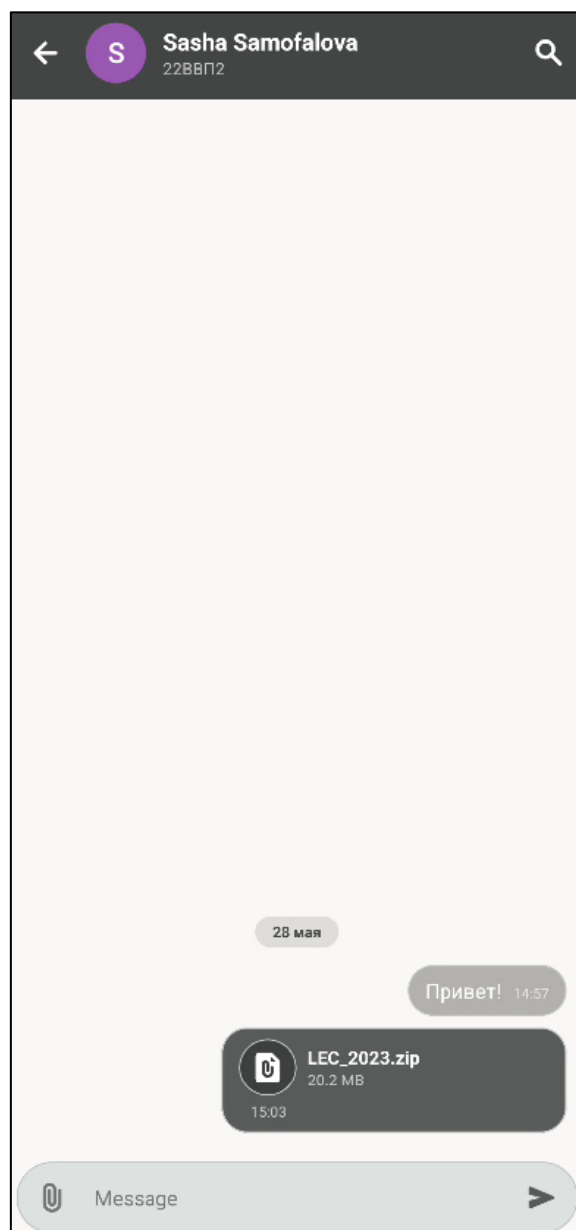


Рисунок 45 – Найденное сообщение

Локальный же поиск работает непосредственно внутри конкретного чата (личного или группового). Данная функция вызывается посредством нажатия на верхнее меню переписки и позволяет быстро находить нужную информацию. Если же найдено несколько совпадений, то пользователь может перемещаться между найденными сообщениями с помощью навигационных стрелок, как показано на рисунках 46-47

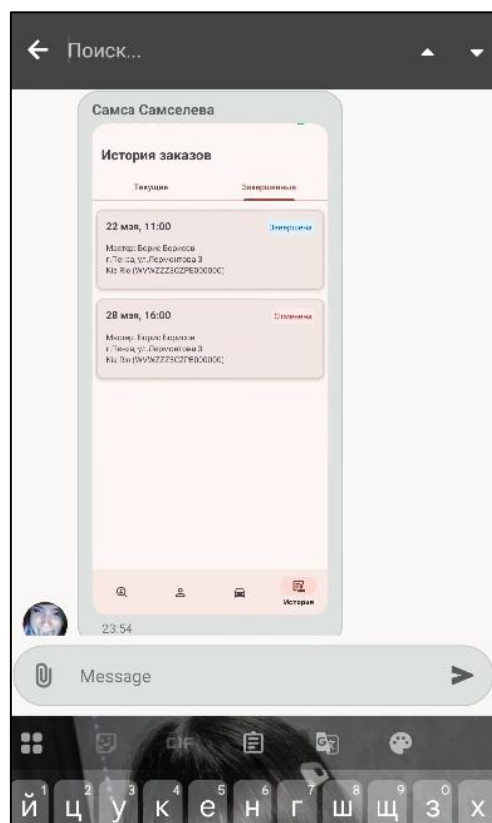


Рисунок 46 – Локальный поиск

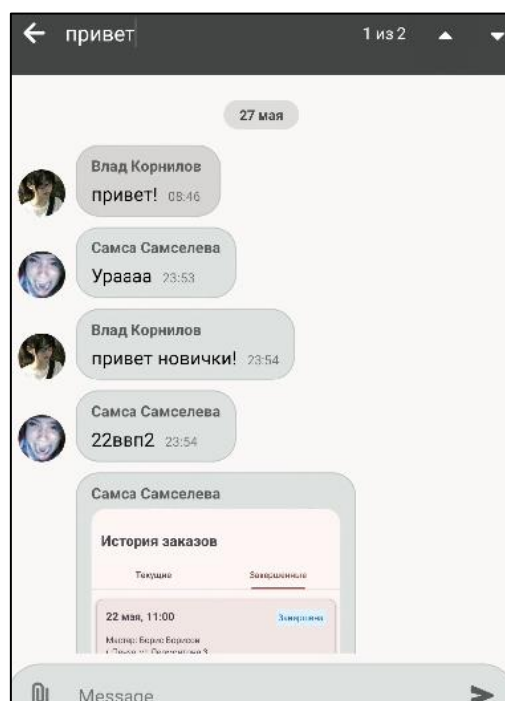


Рисунок 47 – Найденные сообщение

Наглядно процесс тестирования функций глобального и локального поиска сообщений приведен в таблице 5.

Таблица 5 – Тестирования создания и редактирования групп

№	Сценарий и шаги тестирования	Ожидаемый результат	API	Статус
1	Глобальный поиск 1. Находясь на главном экране нажать на строку поиска 2. Ввести данные	Список активных диалогов динамически фильтруется, оставляя только чаты, содержащие введенные данные	Отправка GET запроса на /api/auth/messages/search Возврат массива подходящих сообщений.	Успешно
2	Локальный поиск 1. Открыть нужный диалог 2. Вызвать строку поиска 3. Ввести данные	Экран чата фокусируется на первом найденном сообщении, текст подсвечивается. Доступны кнопки для навигации по совпадениям	Запрос и получение списка от сервера	Успешно

6 Безопасность жизнедеятельности

Разработка и последующая эксплуатация современного программного обеспечения неразрывно связаны с длительным нахождением человека за экранами ПК и мобильных устройств. Для минимизации негативного воздействия технических средств на здоровье разработчиков и конечных пользователей применяются специализированные санитарные правила, гигиенические нормативы и правила техники безопасности [9]. Игнорирование данных требований может повлечь за собой ухудшение самочувствия, развитие профессиональных заболеваний или порчу материального имущества.

6.1 Организация безопасности условий труда при работе с ПК

В процессе проектирования и написания исходного кода серверной и клиентской частей инженер-программист подвергается воздействию ряда опасных и вредных производственных факторов. К основным из них относятся повышенное зрительное напряжение, длительное статическое напряжение мускулатуры опорно-двигательного аппарата, воздействие электромагнитных полей, а также риск поражения электрическим током при неисправности оборудования.

Для снижения рисков развития патологий зрения рабочее помещение должно быть обеспечено сбалансированным естественным и искусственным освещением. Расположение монитора относительно оконных проемов выбирается таким образом, чтобы полностью исключить появление бликов на экране.

Не менее важным является общая эргономика рабочего места. Геометрические параметры стола и офисного кресла должны регулироваться, обеспечивая удержание естественной осанки: угол в коленных и локтевых суставах должен составлять около 90 градусов, ступни пользователя должны полностью опираться на поверхность пола. Монитор компьютера рекомендуется размещать на расстоянии 60–70 см от глаз, что соответствует примерно длине

					ПГУ1.090301.37.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

вытянутой руки. При этом верхняя кромка экрана должна находиться на уровне глаз или на 15–20 градусов ниже их линии.

Для обеспечения пожарной и электробезопасности запрещается эксплуатация кабелей питания с поврежденной изоляцией. Системные блоки и периферийное оборудование необходимо устанавливать так, чтобы не перекрывать вентиляционные отверстия, предотвращая перегрев аппаратных компонентов. В соответствии с действующими санитарными нормами, при непрерывной работе за компьютером каждые 45–60 минут необходимо устраивать перерывы продолжительностью 10–15 минут для выполнения комплекса физических упражнений, направленных на снятие зрительного и мышечного утомления.

6.2 Техника безопасности при эксплуатации мобильных устройств

Поскольку разрабатываемый программный продукт ориентирован на мобильную платформу Android, необходимо учитывать специфику безопасного взаимодействия пользователя со смартфоном в повседневной жизни.

С технической точки зрения безопасность эксплуатации мобильного устройства напрямую зависит от исправности его аппаратных компонентов, в особенности аккумулятора. Категорически запрещается использование деформированных, вздутых или подвергшихся сильному перегреву устройств, а также неоригинальных зарядных адаптеров, способных вызвать короткое замыкание. Корпус смартфона не должен подвергаться избыточным механическим нагрузкам, падениям или воздействию влаги.

Длительное удержание смартфона в руках вызывает избыточную нагрузку на шейный отдел позвоночника и суставы кистей. При чтении или написании сообщений в мессенджере рекомендуется держать устройство на уровне груди, чтобы минимизировать наклон головы вперед. Для предотвращения спазма аккомодации и сухости глаз экран мобильного телефона должен находиться на

					ПГУ1.090301.37.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

расстоянии не менее 30–40 см от лица, а яркость дисплея должна динамически адаптироваться под уровень внешнего освещения [10].

Современные медицинские исследования указывают на необходимость контроля времени использования мобильных приложений для предотвращения синдрома цифровой усталости и нарушений фаз сна. Пользователям мессенджера рекомендуется ограничивать непрерывное время взаимодействия со смартфоном в вечерние часы и использовать встроенные системные режимы защиты зрения, снижающие интенсивность излучения синего спектра света.

					ПГУ1.090301.37.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

7 Заключение

Выпускная квалификационная работа успешно завершена проектированием и программной реализацией клиент-серверного мессенджера для студентов ПГУ, который представляет собой законченное, полностью работоспособное и протестированное решение. Проведенный на начальных этапах предпроектный анализ предметной области и существующих программных аналогов позволил четко сформулировать функциональные требования и спроектировать систему, эффективно изолирующую учебное пространство от стороннего развлекательного контента.

На стороне мобильного клиента под управлением ОС Android была успешно внедрена архитектура MVVM, реализованная на языке Kotlin. Это позволило отделить логику интерфейса от бизнес-логики, обеспечить отзывчивость и надежность приложения. Разработка серверной части на базе фреймворка Spring Boot с использованием многоуровневой архитектуры обеспечила централизованную обработку запросов и строгое разграничение прав доступа. Были получены и закреплены практические навыки работы с реляционной СУБД MySQL, организации обмена сообщениями в реальном времени, защиты пользовательских данных и сессий с помощью JWT-токенов в Spring Security, а также реализации асинхронных сетевых запросов посредством связки Retrofit, Coroutines и StateFlow.

В дальнейшем программный комплекс может быть модернизирован за счет расширения функциональных возможностей, таких как добавление аудио и видеозвонков, реализация end-to-end шифрования для повышения конфиденциальности, а также интеграция мессенджера с внутренней экосистемой университета.

Список использованных источников

1. Mobile Operating System Market Share Worldwide [Электронный ресурс]
URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide> (дата обращения: 12.04.2026).
2. Telegram APIs and Platform Features [Электронный ресурс] URL:
<https://core.telegram.org/> (дата обращения: 14.04.2026).
3. Kotlin Programming Language: Official Documentation [Электронный
ресурс] URL: <https://kotlinlang.org/docs/home.html> (дата обращения: 20.04.2026).
4. Spring Boot Reference Guide [Электронный ресурс] URL:
<https://docs.spring.io/spring-boot/docs/current/reference/html/> (дата обращения: 30.04.2026).
5. Meet Android Studio [Электронный ресурс] URL:
<https://developer.android.com/studio/intro> (дата обращения: 02.05.2026).
6. IntelliJ IDEA Documentation [Электронный ресурс] URL:
<https://www.jetbrains.com/help/idea/> (дата обращения: 10.05.2026).
7. MySQL 8.0 Reference Manual [Электронный ресурс] URL:
<https://dev.mysql.com/doc/refman/8.0/en/> (дата обращения: 11.05.2026).
8. Guide to App Architecture: Android Architecture Components [Электронный
ресурс] URL: <https://developer.android.com/topic/architecture> (дата обращения: 12.05.2026).
9. Санитарно-эпидемиологические требования к условиям труда
[Электронный ресурс] URL: <https://docs.cntd.ru/document/573215686> (дата
обращения: 18.05.2026).
10. Влияние мобильных устройств на здоровье человека и меры
безопасности [Электронный ресурс] URL: <http://cgon.rospotrebnadzor.ru> (дата
обращения: 26.05.2026).

Приложение А

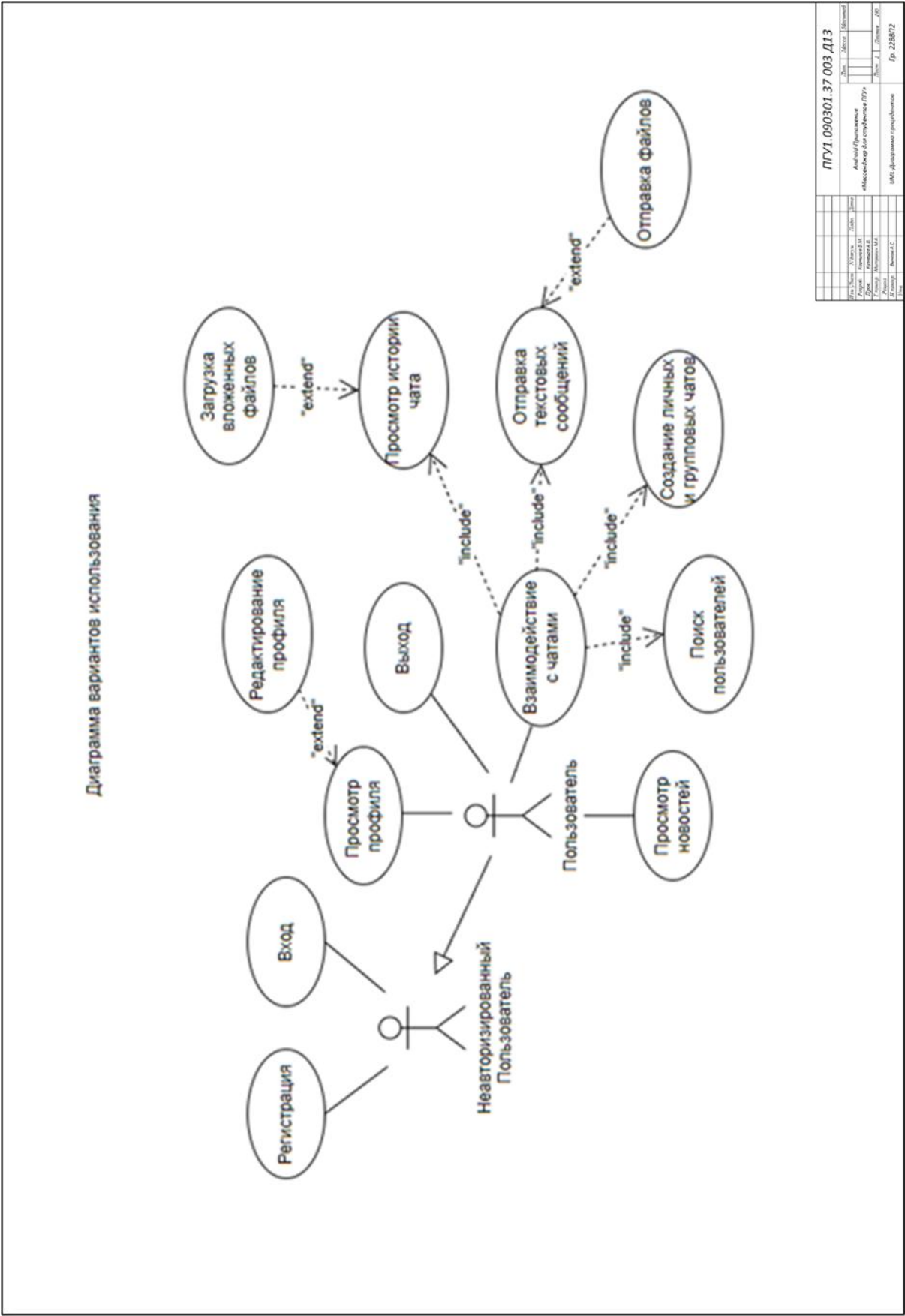
(Обязательное)

Мобильное приложение «Мессенджер для студентов ПГУ» для ОС Android

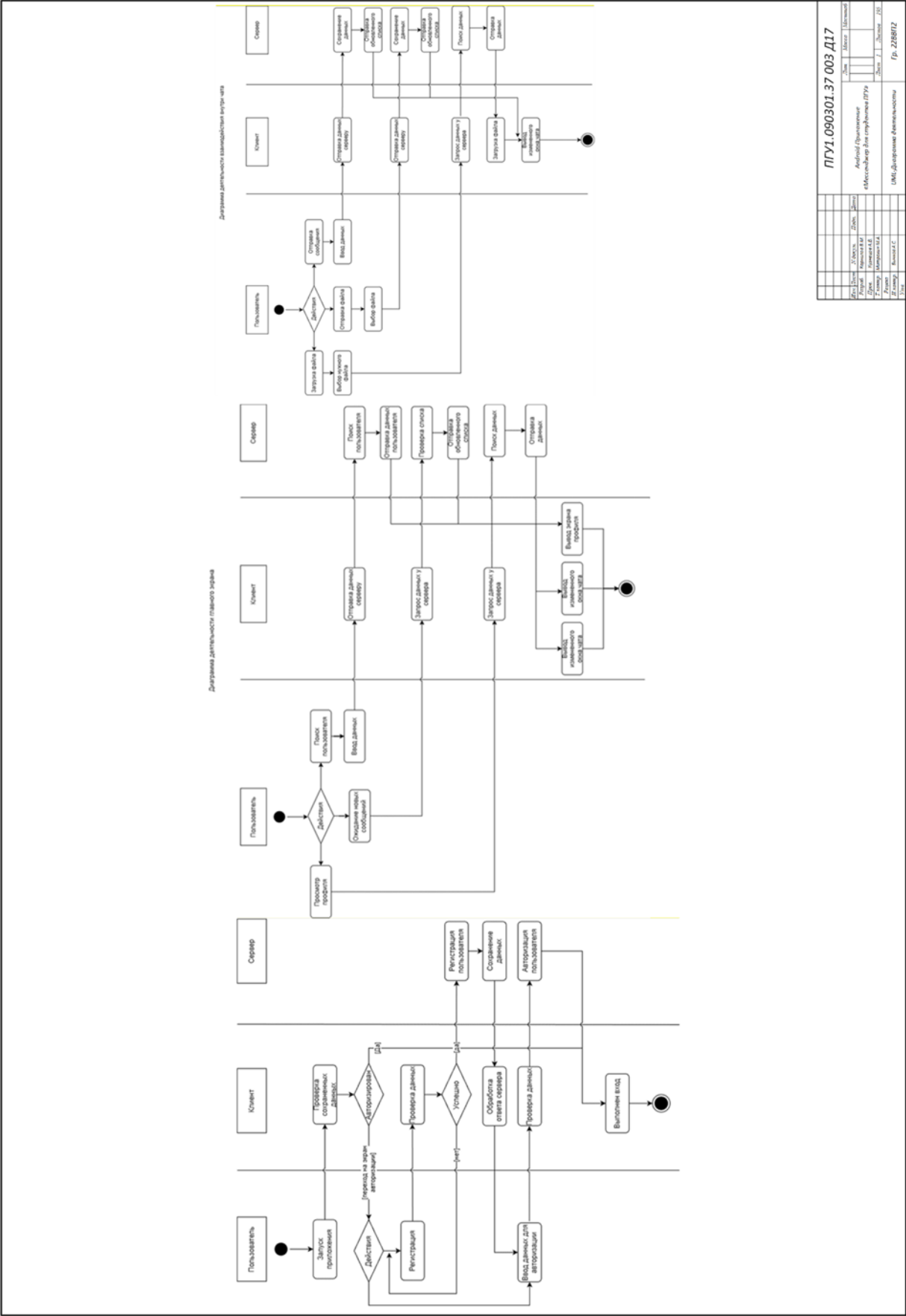
UML – диаграммы

					<i>ПГУ1.090301.37.001 ПЗ</i>	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1-Диаграмма прецедентов



Приложение А.2-Диаграмма деятельности

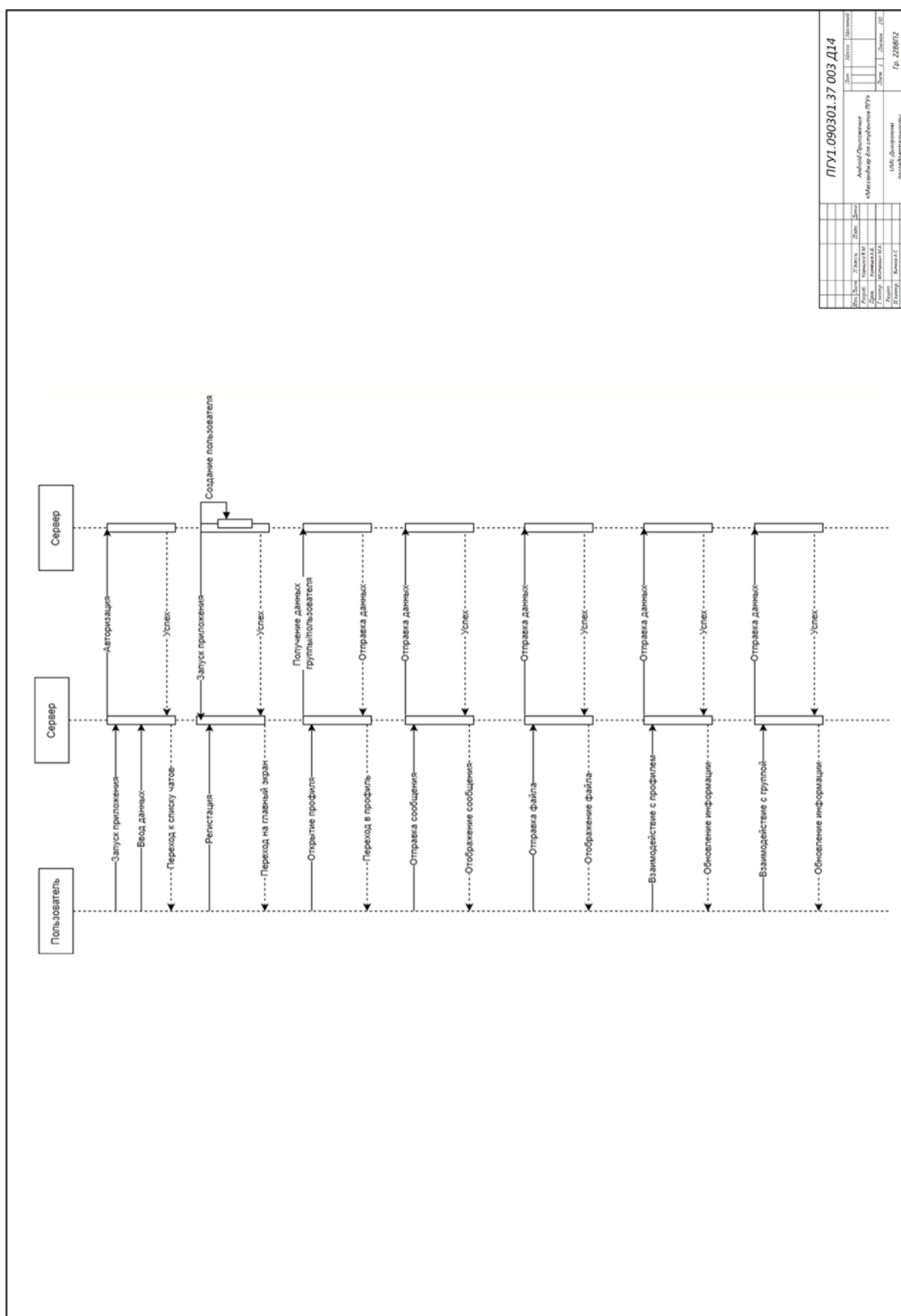


ПГУ1.090301.37.003 Д17									
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель

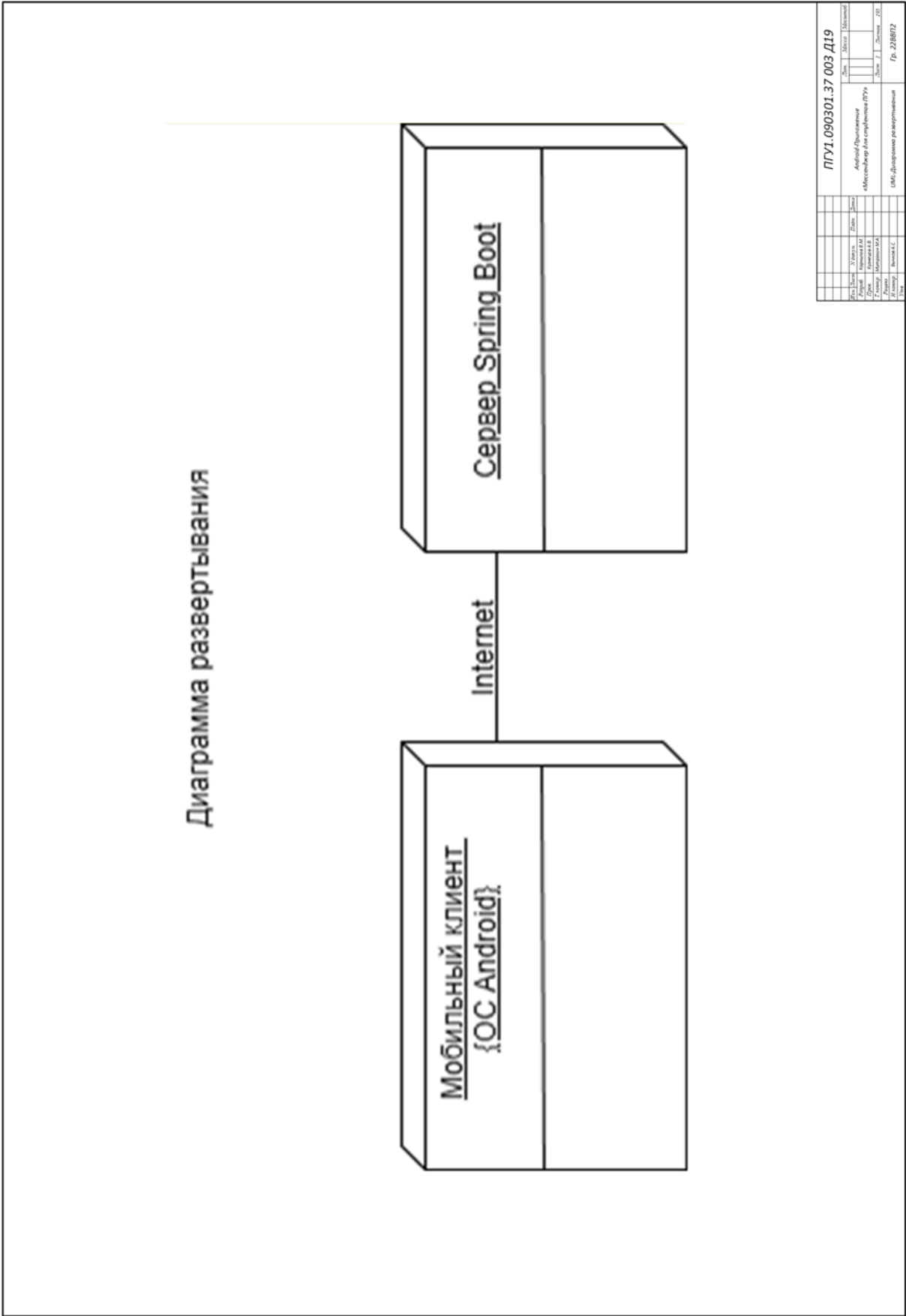
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.37.001 ПЗ

Приложение А.3-Диаграмма последовательностей

[illegible]

Приложение А.4-Диаграмма развертывания



Приложение Б

(Обязательно)

Мобильное приложение «Мессенджер для студентов ПГУ» для ОС Android

Листинг программы

					ПГУ1.090301.37.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «LoginActivity.kt»

```
package com.example.tiltedgram.login

import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.core.widget.doAfterTextChanged
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.LifecycleScope
import androidx.lifecycle.repeatOnLifecycle
import com.example.tiltedgram.main.MainActivity
import com.example.tiltedgram.login.RegistrationActivity
import com.example.tiltedgram.databinding.ActivityLoginBinding
import kotlinx.coroutines.launch
import
androidx.core.splashscreen.SplashScreen.Companion.installSplashScreen
import com.example.tiltedgram.data.NetworkModule
import com.google.firebase.messaging.FirebaseMessaging
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers

class LoginActivity : AppCompatActivity() {

    private lateinit var binding: ActivityLoginBinding
    private val viewModel: LoginViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        val splashScreen = installSplashScreen()
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        val sharedPreferences = getSharedPreferences("MessengerPrefs",
MODE_PRIVATE)
        val savedUserId = sharedPreferences.getLong("USER_ID", -1L)

        if (savedUserId != -1L) {
            uploadFcmTokenToServer(savedUserId)
            goMainActivity()
            return
        }
        binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setupListeners()
        observeViewModel()
    }
}
```

```

FirebaseMessaging.getInstance().token.addOnCompleteListener {
task ->
    if (task.isSuccessful) {
        val token = task.result
        Log.d("FCM_TOKEN", "Актуальный токен: $token")
    } else {
        Log.e("FCM_TOKEN", "Ошибка получения токена",
task.exception)
    }
}

private fun setupListeners() {
    binding.etUsername.doAfterTextChanged { text ->
        viewModel.onUsernameChange(text.toString())
    }
    binding.etPassword.doAfterTextChanged { text ->
        viewModel.onPasswordChange(text.toString())
    }

    binding.btnLogin.setOnClickListener {
        viewModel.login()
    }

    binding.btnGoToRegister.setOnClickListener {
        val intent = Intent(this,
RegistrationActivity::class.java)
        startActivity(intent)
    }
}

private fun observeViewModel() {
    lifecycleScope.launch {
        repeatOnLifecycle(Lifecycle.State.STARTED) {
            viewModel.state.collect { state ->

                binding.btnLogin.isEnabled =
!state.isLoading

                if (state.errorMessage != null) {
                    Toast.makeText(this@LoginActivity,
state.errorMessage, Toast.LENGTH_SHORT).show()
                }

                if (state.isLoginSuccessful) {
                    val sharedPrefs =
getSharedPreferences("MessengerPrefs", MODE_PRIVATE)
                    sharedPrefs.edit()
                        .putLong("USER_ID",
viewModel.loggedInUserId)

```

```

        viewModel.loggedInInUsername).putString("USER_NAME",
viewModel.loggedInInFullName).putString("USER_FIO",
viewModel.loggedInInEmail).putString("USER_EMAIL",
viewModel.loggedInInFaculty).putString("USER_FACULTY",
viewModel.loggedInInGroup).putString("USER_GROUP",
viewModel.loggedInInAvatarUrl).putString("USER_AVATAR",
viewModel.loggedInInToken).putString("JWT_TOKEN",
        .apply()

uploadFcmTokenToServer(viewModel.loggedInInUserId)
        Toast.makeText(this@LoginActivity,
"Добро пожаловать!", Toast.LENGTH_SHORT).show()
        goToMainActivity()

    }

}

}

private fun goToMainActivity(){
    val intent = Intent(this, MainActivity::class.java)
    startActivity(intent)
    finish()
}

private fun uploadFcmTokenToServer(userId: Long) {
    val sharedPrefs = getSharedPreferences("MessengerPrefs",
MODE_PRIVATE)
    val jwtToken = sharedPrefs.getString("JWT_TOKEN", "") ?:
    ""

    FirebaseMessaging.getInstance().token.addOnCompleteListener {
task ->
        if (!task.isSuccessful) return@addOnCompleteListener
        val fcmToken = task.result ?:
return@addOnCompleteListener

        CoroutineScope(Dispatchers.IO).launch {
            try {
                NetworkModule.authApi.updateFcmToken(
                    "Bearer $jwtToken",
                    userId,
                    mapOf("token" to fcmToken)
                )
            }
        }
    }
}

```

```

        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}
}
}

```

Файл «LoginViewModel.kt»

```

package com.example.tiltedgram.login

import androidx.lifecycle.ViewModel
import androidx.lifecycle.ViewModelScope
import com.example.tiltedgram.data.LoginRequest
import com.example.tiltedgram.data.NetworkModule
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.update
import kotlinx.coroutines.launch

data class LoginState(
    val username: String = "",
    val password: String = "",
    val isLoading: Boolean = false,
    val errorMessage: String? = null,
    val isLoginSuccessful: Boolean = false
)

class LoginViewModel : ViewModel(){

    private val _state = MutableStateFlow(LoginState())
    val state: StateFlow<LoginState> = _state.asStateFlow()

    fun onUsernameChange(newValue: String){
        _state.update { it.copy(username = newValue,
        errorMessage = null) }
    }

    fun onPasswordChange(newValue: String){
        _state.update { it.copy(password = newValue,
        errorMessage = null) }
    }

    var loggedInUserId: Long = -1L
    var loggedInUsername: String = ""
    var loggedInFullName: String = ""
    var loggedInEmail: String = ""
    var loggedInFaculty: String = ""
    var loggedInGroup: String = ""
    var loggedInAvatarUrl: String? = null
    var loggedInToken: String = ""

```

```

fun login(){
    val currentState = _state.value

    if (currentState.username.isBlank() ||
currentState.password.isBlank()){
        _state.update { it.copy(errorMessage = "Введите имя
пользователя и пароль") }
        return
    }

    //coroutine (async) network
viewModelScope.launch {
        _state.update { it.copy(isLoading = true,
errorMessage = null) }

        try {
            val request =
LoginRequest(currentState.username, currentState.password)
            val response =
NetworkModule.authApi.login(request)

            if (response.isSuccessful){
                val body = response.body()
                if (body != null){
                    loggedInUserId = body.userId
                    loggedInUsername = body.username
                    loggedInFullName = body.fullName
                    loggedInEmail = body.email
                    loggedInFaculty = body.faculty
                    loggedInGroup = body.studyGroup
                    loggedInAvatarUrl = body.avatarUrl
                    loggedInToken = body.token
                    NetworkModule.token = body.token
                    _state.update { it.copy(isLoading =
false, isLoginSuccessful = true) }
                    println("Успешный вход, id пользователя:
${body?.userId}")
                }
            } else {
                _state.update {
                    it.copy(isLoading = false, errorMessage
= "Неверное имя пользователя или пароль")
                }
            }
        } catch (e: Exception){
            _state.update {
                it.copy(isLoading = false, errorMessage =
"Ошибка сети")
            }
        }
    }
}

```

```
}
}
```

Файл «RegistrationActivity.kt»

```
package com.example.tiltedgram.login

import android.R
import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import android.widget.Toast
import androidx.activity.enableEdgeToEdge
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.core.widget.doAfterTextChanged
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.LifecycleScope
import androidx.lifecycle.repeatOnLifecycle
import com.example.tiltedgram.data.RegistrationState
import com.example.tiltedgram.login.RegistrationViewModel
import
com.example.tiltedgram.databinding.ActivityRegistrationBinding
import kotlinx.coroutines.launch

class RegistrationActivity : AppCompatActivity() {

    private lateinit var binding: ActivityRegistrationBinding
    private val viewModel: RegistrationViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        // Инициализация ViewBinding
        binding =
ActivityRegistrationBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setupDropdowns()
        setupListeners()
        observeViewModel()
    }
    private fun setupDropdowns(){
        val facultyAdapter = ArrayAdapter(
            this,
            R.layout.simple_dropdown_item_1line,
            viewModel.allFaculties
        )
        binding.actvFaculty.setAdapter(facultyAdapter)
    }
    private fun setupListeners() {
        // Передаем ввод текста во ViewModel
```

```

binding.etUsername.doAfterTextChanged { text ->
    viewModel.onUsernameChange(text.toString())
}
binding.etFullName.doAfterTextChanged { text ->
    viewModel.onFullNameChange(text.toString())
}
binding.etEmail.doAfterTextChanged { text ->
    viewModel.onEmailChange(text.toString())
}
binding.etPassword.doAfterTextChanged { text ->
    viewModel.onPasswordChange(text.toString())
}
binding.actvFaculty.setOnItemClickListener { parent, _,
position, _ ->
    val selectedFaculty =
parent.getItemAtPosition(position) as String
    viewModel.onFacultyChange(selectedFaculty)

    binding.actvGroup.setText("", false)

    val groups =
viewModel.facultiesAndGroups[selectedFaculty] ?: emptyList()
    val groupAdapter = ArrayAdapter(
        this,
        R.layout.simple_dropdown_item_1line,
        groups
    )
    binding.actvGroup.setAdapter(groupAdapter)
    binding.actvGroup.isEnabled = groups.isNotEmpty()
}

binding.actvGroup.setOnItemClickListener { parent, _,
position, _ ->
    val selectedGroup =
parent.getItemAtPosition(position) as String
    viewModel.onGroupChange(selectedGroup)
}

// Обработка клика по кнопке
binding.btnRegister.setOnClickListener {
    viewModel.register()
}
binding.btnBack.setOnClickListener {
    finish()
}
}

private fun observeViewModel() {
    lifecycleScope.launch {
        repeatOnLifecycle(Lifecycle.State.STARTED) {
            viewModel.state.collect { state ->

```

```

        updateUser(state)
    }
}

private fun updateUser(state: RegistrationState) {
    // Показываем или прячем загрузку
    if (state.isLoading) {
        binding.progressBar.visibility = View.VISIBLE
        binding.btnRegister.visibility = View.GONE
    } else {
        binding.progressBar.visibility = View.GONE
        binding.btnRegister.visibility = View.VISIBLE
    }

    // Показываем ошибку, если она есть
    if (state.errorMessage != null) {
        binding.tvError.text = state.errorMessage
        binding.tvError.visibility = View.VISIBLE
    } else {
        binding.tvError.visibility = View.GONE
    }

    // Если регистрация успешна - переходим на экран входа
    if (state.isRegistrationSuccessful) {
        Toast.makeText(this, "Успешно! Теперь войдите.",
            Toast.LENGTH_SHORT).show()
        finish() // Закрываем экран регистрации
    }
}
}

```

Файл «RegistrationViewModel.kt»

```

package com.example.tiltedgram.login

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.tiltedgram.data.NetworkModule
import com.example.tiltedgram.data.RegisterRequest
import com.example.tiltedgram.data.RegistrationState
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.update
import kotlinx.coroutines.launch

class RegistrationViewModel : ViewModel() {

    // Внутреннее состояние

```

Изм.	Лист	№ докум.	Подпись	Дата

```

private val _state = MutableStateFlow(RegistrationState())
// Публичное состояние
val state: StateFlow<RegistrationState> =
_state.asStateFlow()

//validation
private val nameRegex = "[A-Za-zA-Яа-яЁё\\s]+\\$".toRegex()
private val usernameRegex = "[a-zA-Z0-9_-]+\\$".toRegex()

val facultiesAndGroups = mapOf(
    "ФВТ" to listOf("22ВВП1", "22ВВП2", "22ВВВ1"),
    "ФЭУ" to listOf("ЭК-21", "МЕХ-22"),
    "ЮрФак" to listOf("ЮР-21", "ЮР-22", "ЮР-23")
)

val allFaculties = facultiesAndGroups.keys.toList()

fun onUsernameChange(newValue: String) {
    _state.update { it.copy(username = newValue,
errorMessage = null) }
}
fun onFullNameChange(newValue: String) {
    _state.update { it.copy(fullName = newValue,
errorMessage = null) }
}
fun onEmailChange(newValue: String) {
    _state.update { it.copy(email = newValue, errorMessage =
null) }
}
fun onPasswordChange(newValue: String) {
    _state.update { it.copy(password = newValue,
errorMessage = null) }
}
fun onFacultyChange(newFaculty: String){
    _state.update {
        it.copy(
            faculty = newFaculty,
            group = "",
            availableGroups = facultiesAndGroups[newFaculty]
?: emptyList(),
            errorMessage = null
        )
    }
}
fun onGroupChange(newGroup: String){
    _state.update { it.copy(group = newGroup, errorMessage =
null) }
}

fun register() {
    val currentState = _state.value

```

```

        if (currentState.username.isBlank() ||
currentState.fullName.isBlank() || currentState.email.isBlank()
||
        currentState.password.isBlank() ||
currentState.faculty.isBlank() || currentState.group.isBlank())
{
    _state.update { it.copy(errorMessage = "Заполните
все поля и выберите группу") }
    return
}
    if (!currentState.fullName.matches(nameRegex)) {
        _state.update { it.copy(errorMessage = "Имя может
содержать только буквы и пробелы") }
        return
    }
    if (!currentState.username.matches(usernameRegex)) {
        _state.update { it.copy(errorMessage = "Никнейм
должен содержать только английские буквы, цифры, '-' и '_'") }
        return
    }
    viewModelScope.launch {
        _state.update { it.copy(isLoading = true,
errorMessage = null) }
        try {
            //request
            val request = RegisterRequest(
                username = currentState.username,
                fullName = currentState.fullName,
                email = currentState.email,
                password = currentState.password,
                faculty = currentState.faculty,
                group = currentState.group
            )
            //send
            val response =
NetworkModule.authApi.register(request)

            //answer
            if (response.isSuccessful) {
                //code 200
                _state.update {
                    it.copy(isLoading = false,
isRegistrationSuccessful = true)
                }
            } else {
                if (response.code() == 400) {
                    var serverMessage = "Неверные данные для
регистрации"

                    try {

```

```

        val errorString =
response.errorBody()?.string()
        if (!errorString.isNullOrEmpty()) {
            val jsonObject =
org.json.JSONObject(errorString)
            if (jsonObject.has("error")) {
                serverMessage =
jsonObject.getString("error")
            }
        }
    } catch (e: Exception) {
        e.printStackTrace()
    }
    _state.update {
        it.copy(isLoading = false,
errorMessage = serverMessage)
    }
    } else {
        _state.update {
            it.copy(isLoading = false,
errorMessage = "Ошибка сервера: ${response.code()}")
        }
    }
}
} catch (e: Exception){
//network err
_state.update {
    it.copy(isLoading = false, errorMessage =
"Ошибка сети: проверьте подключение к серверу")
}
}
}
}
}

```

Файл «MainActivity.kt»

```

package com.example.tiltedgram.main

import android.Manifest
import android.content.Context
import android.content.SharedPreferences
import android.content.pm.PackageManager
import android.graphics.Color
import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.lifecycleScope
import androidx.lifecycle.repeatOnLifecycle
import androidx.navigation.fragment.NavHostFragment

```

```

import coil.Coil
import coil.ImageLoader
import coil.load
import com.example.tiltedgram.R
import com.example.tiltedgram.data.NetworkModule
import com.example.tiltedgram.databinding.ActivityMainBinding
import kotlinx.coroutines.launch
import androidx.navigation.NavOptions
import
androidx.navigation.NavGraph.Companion.findStartDestination
import android.content.res.ColorStateList
import android.os.Build
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.core.content.ContextCompat
import com.example.tiltedgram.utils.loadAvatarOrDefault

class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    private val viewModel: MainViewModel by viewModels()
    private var currentTabId = R.id.chatsFragment

    private val prefListener =
        SharedPreferences.OnSharedPreferenceChangeListener { prefs, key
->
            if (key == "USER_AVATAR") {
                val avatarUrl = prefs.getString("USER_AVATAR", null)
                val fullName = prefs.getString("USER_FIO", "Вы") ?:
"Вы"

                refreshAvatar(avatarUrl, fullName)
            }
        }
    private val requestPermissionLauncher =
        registerForActivityResult(
            ActivityResultContracts.RequestPermission()
        ) { isGranted: Boolean ->
            if (isGranted) {
            } else {
                Toast.makeText(this, "Вы не увидите прогресс
загрузки в фоне", Toast.LENGTH_SHORT).show()
            }
        }
    private fun askNotificationPermission() {
        if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.TIRAMISU) {
            if (ContextCompat.checkSelfPermission(this,
Manifest.permission.POST_NOTIFICATIONS) ==
PackageManager.PERMISSION_GRANTED) {
            } else {

```

```

requestPermissionLauncher.launch(Manifest.permission.POST_NOTIFICATIONS)
    }
}

override fun onCreate(savedInstanceState: Bundle?) {
    window.requestFeature(android.view.Window.FEATURE_ACTIVITY_TRANSITIONS)
    super.onCreate(savedInstanceState)
    askNotificationPermission()
    enableEdgeToEdge()

    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    val imageLoader =
        ImageLoader.Builder(applicationContext)
            .okHttpClient(NetworkModule.client)
            .build()
    Coil.setImageLoader(imageLoader)

    val navHostFragment =
        supportFragmentManager.findFragmentById(R.id.nav_host_fragment)
    as NavHostFragment
    val navController = navHostFragment.navController
    setupCustomBottomNavigation(navController)

    val sharedPrefs = getSharedPreferences("MessengerPrefs",
        Context.MODE_PRIVATE)

    sharedPrefs.registerOnSharedPreferenceChangeListener(prefListener)

    val currentUserId = sharedPrefs.getLong("USER_ID", -1L)
    val studyGroup = sharedPrefs.getString("USER_GROUP", "")
    ?: ""
    NetworkModule.token = sharedPrefs.getString("JWT_TOKEN",
    null)

    val avatarUrl = sharedPrefs.getString("USER_AVATAR",
    null)
    val fullName = sharedPrefs.getString("USER_FIO", "Вы")
    ?: "Вы"
    refreshAvatar(avatarUrl, fullName)

    if (currentUserId != -1L) {
        viewModel.startPolling(currentUserId, studyGroup)
    }
}

```

					ПГУ1.090301.37.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

```

        observeViewModel()
    }

    private fun refreshAvatar(url: String?, name: String) {
        // Используем наше новое расширение вместо дефолтного
        .load()

binding.bottomNav.btnProfileAvatar.loadAvatarOrDefault(url,
name)
    }

    private fun observeViewModel() {
        lifecycleScope.launch {
            repeatOnLifecycle(Lifecycle.State.STARTED) {
                launch {
                    viewModel.isConnected.collect { isConnected
->
                        if (isConnected) {

binding.bottomNav.cvStatusIndicator.setCardBackgroundColor(Color
StateList.valueOf(Color.parseColor("#4CAF50")))
                            val sharedPrefs =
getSharedPreferences("MessengerPrefs", Context.MODE_PRIVATE)
                            val avatarUrl =
sharedPrefs.getString("USER_AVATAR", null)
                            val fullName =
sharedPrefs.getString("USER_FIO", "Вы") ?: "Вы"
                            refreshAvatar(avatarUrl, fullName)
                        } else {

binding.bottomNav.cvStatusIndicator.setCardBackgroundColor(Color
StateList.valueOf(Color.parseColor("#837B7B")))
                    }
                }
            }
        }
    }

    private fun setupCustomBottomNavigation(navController:
androidx.navigation.NavController) {
        val bottomNav = binding.bottomNav

        val tabIndices = mapOf(
            R.id.chatsFragment to 0,
            R.id.searchFragment to 1,
            R.id.myProfileFragment to 2
        )

        fun navigateWithAnimation(targetId: Int) {

```

```

        val realCurrentId =
navController.currentDestination?.id ?: R.id.chatsFragment
        if (realCurrentId == targetId) return

        val currentIndex = tabIndices[realCurrentId] ?: 0
        val targetIndex = tabIndices[targetId] ?: 0

        val navOptionsBuilder = NavOptions.Builder()
            .setLaunchSingleTop(true)

        if (targetIndex > currentIndex) {
            navOptionsBuilder
                .setEnterAnim(R.anim.slide_in_right)
                .setExitAnim(R.anim.slide_out_left)
                .setPopEnterAnim(R.anim.slide_in_left)
                .setPopExitAnim(R.anim.slide_out_right)
        } else {
            navOptionsBuilder
                .setEnterAnim(R.anim.slide_in_left)
                .setExitAnim(R.anim.slide_out_right)
                .setPopEnterAnim(R.anim.slide_in_right)
                .setPopExitAnim(R.anim.slide_out_left)
        }

        currentTabId = targetId
        navController.navigate(targetId, null,
navOptionsBuilder.build())
    }

    bottomNav.tabChats.setOnClickListener {
navigateWithAnimation(R.id.chatsFragment) }
    bottomNav.tabSearch.setOnClickListener {
navigateWithAnimation(R.id.searchFragment) }
    bottomNav.tabProfile.setOnClickListener {
navigateWithAnimation(R.id.myProfileFragment) }

    navController.addOnDestinationChangedListener { _,
destination, _ ->
        val transition =
androidx.transition.AutoTransition()
            transition.duration = 150L

        androidx.transition.TransitionManager.beginDelayedTransition(bot
tomNav.root as android.view.ViewGroup, transition)

        bottomNav.bgChats.setBackgroundResource(android.R.color.transpar
ent)

```

```

bottomNav.bgSearch.setBackgroundResource(android.R.color.transparent)

bottomNav.bgProfile.setBackgroundResource(android.R.color.transparent)

        when (destination.id) {
            R.id.chatsFragment ->
bottomNav.bgChats.setBackgroundResource(R.drawable.bg_tab_active)
            R.id.searchFragment ->
bottomNav.bgSearch.setBackgroundResource(R.drawable.bg_tab_active)
            R.id.myProfileFragment ->
bottomNav.bgProfile.setBackgroundResource(R.drawable.bg_tab_active)
        }
    }

    override fun onDestroy() {
        super.onDestroy()
        viewModel.stopPolling()
        val sharedPreferences = getSharedPreferences("MessengerPrefs",
Context.MODE_PRIVATE)

sharedPrefs.unregisterOnSharedPreferenceChangeListener(prefListener)
    }
}

```

Файл «MainViewModel.kt»

```

package com.example.tiltedgram.main

import android.util.Log
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.tiltedgram.data.NetworkModule
import kotlinx.coroutines.Job
import kotlinx.coroutines.async
import kotlinx.coroutines.coroutineScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.isActive
import kotlinx.coroutines.launch

class MainViewModel : ViewModel() {
    //loading

```

```

private val _isLoading = MutableStateFlow(false)
val isLoading: StateFlow<Boolean> = _isLoading.asStateFlow()

//chat model
private val _chat =
MutableStateFlow<List<Chat>>(emptyList())
val chats: StateFlow<List<Chat>> = _chat.asStateFlow()

//connection state
private val _isConnected = MutableStateFlow(false)
val isConnected: StateFlow<Boolean> =
_isConnected.asStateFlow()

private val _searchResults =
MutableStateFlow<List<Chat>?>(null)
val searchResults: StateFlow<List<Chat>?> =
_searchResults.asStateFlow()

private val _autoGroupId = MutableStateFlow<Long?>(null)
val autoGroupId: StateFlow<Long?> =
_autoGroupId.asStateFlow()

private var searchJob: Job? = null

fun searchChats(userId: Long, query: String) {
    searchJob?.cancel()

    if (query.isBlank()) {
        _searchResults.value = null
        return
    }

    searchJob = viewModelScope.launch {
        delay(400)
        try {
            val response =
NetworkModule.authApi.searchGlobalDialogs(userId, query)
            if (response.isSuccessful) {
                val dialogs = response.body() ?: emptyList()
                val mappedChats = dialogs.map { d ->
                    Chat(
                        id = d.id,
                        username = d.username ?: (if
(d.isGroup) "Группа" else ""),
                        fullName = d.name,
                        lastMessage = d.lastMessage,
                        timestamp = d.timestamp,
                        avatarUrl = d.avatarUrl,
                        faculty = d.faculty ?: "Групповой
чат",
                        group = d.group ?: "",

```

```

        email = d.email,
        isGroup = d.isGroup,
        adminId = d.adminId ?: -1L,
        messageId = d.messageId
    )
    }
    _searchResults.value = mappedChats
}
} catch (e: Exception) {
    e.printStackTrace()
}
}

private var pollingJob: Job? = null

fun startPolling(currentUserId: Long, studyGroup: String) {
    if(pollingJob != null) return

    viewModelScope.launch {
        try {
            val res =
NetworkModule.authApi.getAutoGroupId(studyGroup, currentUserId)
            if (res.isSuccessful) {
                _autoGroupId.value =
res.body()?.get("groupId")
            }
        } catch (e: Exception) { e.printStackTrace() }
    }

    pollingJob = viewModelScope.launch {
        while (isActive) {
            try {
                val response =
NetworkModule.authApi.getDialogs(currentUserId)
                if (response.isSuccessful) {
                    _isConnected.value = true
                    val dialogs = response.body() ?:
emptyList()

                    //mapping
                    val mappedChats = dialogs.map { d ->
                        Chat(
                            id = d.id,
                            username = d.username ?: (if
(d.isGroup) "Группа" else ""),
                            fullName = d.name,
                            lastMessage = d.lastMessage,
                            timestamp = d.timestamp,
                            avatarUrl = d.avatarUrl,
                            faculty = d.faculty ?:
"Групповой чат",

```

```

        group = d.group ?: "",
        email = d.email,
        isGroup = d.isGroup,
        adminId = d.adminId ?: -1L,
        messageId = null
    )
    }
    _chat.value = mappedChats
} else {
    _isConnected.value = false
    NetworkModule.resetClient()
}
} catch (e: Exception) {
    _isConnected.value = false
    NetworkModule.resetClient()
    delay(5000)
}
delay(5000)
}
}
}
fun stopPolling() {
    pollingJob?.cancel()
    pollingJob = null
}
}
}

```

Файл «ChatsFragment.kt»

```

package com.example.tiltedgram.main

import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.graphics.Color
import android.net.Uri
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.core.widget.doAfterTextChanged
import androidx.fragment.app.Fragment
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.LifecycleScope
import androidx.lifecycle.repeatOnLifecycle
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.tiltedgram.R
import com.example.tiltedgram.data.NetworkModule

```

```

import com.example.tiltedgram.databinding.FragmentChatsBinding
import
com.google.android.material.dialog.MaterialAlertDialogBuilder
import kotlinx.coroutines.Job
import kotlinx.coroutines.delay
import kotlinx.coroutines.isActive
import kotlinx.coroutines.launch
import androidx.fragment.app.activityViewModels
import com.google.android.material.transition.MaterialSharedAxis

class ChatsFragment : Fragment() {

    private var _binding: FragmentChatsBinding? = null
    private val binding get() = _binding!!

    private val viewModel: MainViewModel by activityViewModels()
    private lateinit var chatAdapter: ChatAdapter
    private var loadingJob: Job? = null
    private lateinit var sharedPrefs: SharedPreferences

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentChatsBinding.inflate(inflater,
container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        sharedPrefs =
requireContext().getSharedPreferences("MessengerPrefs",
Context.MODE_PRIVATE)
        val currentUserId = sharedPrefs.getLong("USER_ID", -1L)
        val userGroup = sharedPrefs.getString("USER_GROUP", "")

        binding.tvGroupChat.text = "💬 Чат группы $userGroup"

        ViewCompat.setOnApplyWindowInsetsListener(binding.appBarLayout)
        { v, insets ->
            val statusBarHeight =
insets.getInsets(WindowInsetsCompat.Type.statusBars()).top
            v.setPadding(0, statusBarHeight, 0, 0)
            insets
        }
    }

```

```

binding.llPsuNews.setOnClickListener {
    MaterialAlertDialogBuilder(requireContext())
        .setTitle("Переход по ссылке")
        .setMessage("Хотите перейти на официальный
сайт?")
        .setPositiveButton("Да") { _, _ ->
            val intent = Intent(Intent.ACTION_VIEW,
Uri.parse("https://pnzgu.ru/news"))
            startActivity(intent)
        }
        .setNegativeButton("Отмена", null)
        .show()
    }
binding.tvGroupChat.setOnClickListener {
    val autoId = viewModel.autoGroupId.value
    if (autoId != null && autoId != -1L) {
        val intent = Intent(requireContext(),
com.example.tiltedgram.chat.ChatActivity::class.java).apply {
            putExtra("RECEIVER_ID", autoId)
            putExtra("IS_GROUP", true)
            putExtra("RECEIVER_FIO", "Беседа
$userGroup")
            putExtra("RECEIVER_AVATAR", "psu_logo")
            putExtra("ADMIN_ID", -1L)
        }
        startActivity(intent)
    } else {
        Toast.makeText(requireContext(), "Чат группы еще
создается сервером...", Toast.LENGTH_SHORT).show()
    }
}

binding.etSearch.doAfterTextChanged { text ->
    viewModel.searchChats(currentUserId,
text.toString().trim())
}

binding.rvChats.layoutManager =
LinearLayoutManager(requireContext())
chatAdapter = ChatAdapter(emptyList())
binding.rvChats.adapter = chatAdapter

observeViewModel()
}

private fun observeViewModel() {
    viewLifecycleOwner.lifecycleScope.launch {
viewLifecycleOwner.repeatOnLifecycle(Lifecycle.State.STARTED) {
    launch {

```

```

        viewModel.searchResults.collect { searchList
->
            if (searchList != null) {
                chatAdapter.updateData(searchList)
            } else {

chatAdapter.updateData(viewModel.chats.value)
            }
        }
    }
    launch {
        viewModel.chats.collect { chatList ->
            if (viewModel.searchResults.value ==
null) {
                chatAdapter.updateData(chatList)
            }
        }
    }
    launch {
        viewModel.isConnected.collect { isConnected
->
            if (isConnected) {
                loadingJob?.cancel()
                binding.tvAppTitle.text =
"TiltedGram"
            } else {
                if (loadingJob?.isActive != true) {
                    loadingJob = launch {
                        var dots = 0
                        while (isActive) {
                            binding.tvAppTitle.text
= "Загрузка " + ".".repeat(dots)
                                dots = (dots + 1) % 4
                                delay(400)
                        }
                    }
                }
            }
        }
    }
    launch {
        viewModel.autoGroupId.collect { autoId ->
            val userGroup =
sharedPrefs.getString("USER_GROUP", "")
            if (autoId != null) {
                binding.tvGroupChat.text = "👥 Чат
группы $userGroup"
            } else {
                binding.tvGroupChat.text = "👥
Загрузка чата группы..."
            }
        }
    }

```



```

class ChatViewHolder(val binding: ItemChatBinding) :
RecyclerView.ViewHolder(binding.root)

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): ChatViewHolder {
        val binding =
ItemChatBinding.inflate(LayoutInflater.from(parent.context),pare
nt,false)
        return ChatViewHolder(binding)
    }

    override fun onBindViewHolder(holder: ChatViewHolder,
position: Int) {
        val chat = chatList[position]
        holder.binding.tvChatName.text = chat.fullName
        holder.binding.tvLastMessage.text = chat.lastMessage

        if (chat.timestamp.isNotEmpty()) {
            try {
                val dateTime =
LocalDateTime.parse(chat.timestamp)
                val date = dateTime.toLocalDate()
                val today = LocalDate.now()

                val timeText = when {
                    date == today -> {

dateTime.format(DateTimeFormatter.ofPattern("HH:mm"))
                    }
                    date.year == today.year -> {

dateTime.format(DateTimeFormatter.ofPattern("d MMM",
Locale("ru")))
                    }
                    else -> {

dateTime.format(DateTimeFormatter.ofPattern("dd.MM.yy"))
                    }
                }
                holder.binding.tvChatTime.text = timeText
                holder.binding.tvChatTime.visibility =
View.VISIBLE
            } catch (e: Exception) {
                holder.binding.tvChatTime.visibility = View.GONE
            }
        } else {
            holder.binding.tvChatTime.visibility = View.GONE
        }
    }

```

```

holder.binding.ivAvatar.loadAvatarOrDefault(chat.avatarUrl,
chat.fullName)

        //transfer to chat
        holder.binding.root.setOnClickListener { view ->
            val context = view.context

            val intent = Intent(context,
ChatActivity::class.java).apply {
                putExtra("RECEIVER_ID", chat.id)
                putExtra("RECEIVER_NAME", chat.username)
                putExtra("RECEIVER_FIO", chat.fullName)
                putExtra("RECEIVER_AVATAR", chat.avatarUrl)
                putExtra("RECEIVER_FACULTY", chat.faculty)
                putExtra("RECEIVER_GROUP", chat.group)
                putExtra("RECEIVER_EMAIL", chat.email)
                putExtra("IS_GROUP", chat.isGroup)
                putExtra("ADMIN_ID", chat.adminId)
                putExtra("SCROLL_TO_MESSAGE_ID", chat.messageId
?: -1L)
            }

            val activity = context as? android.app.Activity
            if (activity != null) {
                val options =
androidx.core.app.ActivityOptionsCompat.makeSceneTransitionAnima
tion(activity)
                activity.startActivity(intent,
options.toBundle())
            } else {
                context.startActivity(intent)
            }
        }
    }

    fun updateData(newChats: List<Chat>) {
        chatList = newChats
        notifyDataSetChanged()
    }

    override fun getItemCount(): Int {
        return chatList.size
    }
}

```

Файл «MyProfileFragment.kt»

```

package com.example.tiltedgram.profile

import android.content.Context
import android.content.Intent
import android.os.Bundle

```

					ПГУ1.090301.37.001 ПЗ	Лист
						105
Изм.	Лист	№ докум.	Подпись	Дата		

```

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.activity.result.PickVisualMediaRequest
import androidx.activity.result.contract.ActivityResultContracts
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.lifecyclescope
import androidx.lifecycle.repeatOnLifecycle
import com.example.tiltedgram.R
import
com.example.tiltedgram.databinding.FragmentMyProfileBinding
import com.example.tiltedgram.utils.loadAvatarOrDefault
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

class MyProfileFragment : Fragment() {

    private var _binding: FragmentMyProfileBinding? = null
    private val binding get() = _binding!!

    private val viewModel: ProfileViewModel by viewModels()
    private var profileUserId: Long = -1L

    private var isEditMode = false

    private val pickMedia =
registerForActivityResult(ActivityResultContracts.PickVisualMedia()) { uri ->
    if (uri != null) {

viewLifecycleOwner.lifecycleScope.launch(Dispatchers.IO) {
        try {
            val bytes =
requireContext().contentResolver.openInputStream(uri)?.readBytes()
            if (bytes != null) {

viewModel.compressAndUploadAvatar(profileUserId, bytes)
            }
        } catch (e: Exception) {
            withContext(Dispatchers.Main) {
                Toast.makeText(requireContext(), "Ошибка
чтения файла", Toast.LENGTH_SHORT).show()
            }
        }
    }
}

```

```

        }
    }
}

override fun onCreateView(inflater: LayoutInflater,
container: ViewGroup?, savedInstanceState: Bundle?): View {
    _binding = FragmentMyProfileBinding.inflate(inflater,
container, false)
    return binding.root
}

override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    ViewCompat.setOnApplyWindowInsetsListener(binding.profileAppBarLayout) { v, insets ->
        val statusBarHeight =
insets.getInsets(WindowInsetsCompat.Type.statusBars()).top
        v.setPadding(0, statusBarHeight, 0, 0)
        insets
    }

    val sharedPrefs =
requireContext().getSharedPreferences("MessengerPrefs",
Context.MODE_PRIVATE)
    profileUserId = sharedPrefs.getLong("USER_ID", -1L)

    updateProfileUI()

    val defaultBtnColor =
binding.btnEditProfile.backgroundTintList

    binding.btnEditProfile.setOnClickListener {
        isEditMode = !isEditMode
        if (isEditMode) {
            binding.btnEditProfile.text = "Готово"
            binding.btnEditProfile.backgroundTintList =
android.content.res.ColorStateList.valueOf(requireContext().getColor(android.R.color.holo_green_dark))

            binding.btnEditName.visibility = View.VISIBLE
            binding.btnEditNickname.visibility =
View.VISIBLE
            binding.btnEditEmail.visibility = View.VISIBLE
            binding.cvEditBadge.visibility = View.VISIBLE
            binding.btnChangeAvatar.isClickable = true
        } else {
            binding.btnEditProfile.text = "Изменить"

```

```

        binding.btnEditProfile.backgroundTintList =
com.google.android.material.color.MaterialColors.getColorStateLi
st(
            requireContext(),

com.google.android.material.R.attr.colorSecondaryContainer,

android.content.res.ColorStateList.valueOf(android.graphics.Colo
r.GRAY)
        )

        binding.btnEditName.visibility = View.GONE
        binding.btnEditNickname.visibility = View.GONE
        binding.btnEditEmail.visibility = View.GONE
        binding.cvEditBadge.visibility = View.GONE
        binding.btnChangeAvatar.isClickable = false
    }
}

binding.btnLogout.setOnClickListener {
    val jwtToken = sharedPrefs.getString("JWT_TOKEN",
"" ) ?: ""

    binding.btnLogout.isEnabled = false

    viewLifecycleOwner.lifecycleScope.launch {
        try {

com.example.tiltedgram.data.NetworkModule.authApi.updateFcmToken
(
            token = "Bearer $jwtToken",
            userId = profileUserId,
            tokenMap = mapOf("token" to "")
        )
    } catch (e: Exception) {
        e.printStackTrace()
    } finally {

        sharedPrefs.edit().clear().apply()

        try {
            val intent = Intent(requireContext(),
Class.forName("com.example.tiltedgram.login.LoginActivity")).app
ly {
                flags =
Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
            }
            startActivity(intent)
            requireActivity().finish()
        } catch (e: Exception) {

```

```

        val intent = Intent(requireContext(),
Class.forName("com.example.tiltedgram.LoginActivity")).apply {
            flags =
Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
        }
        startActivity(intent)
        requireActivity().finish()
    }
}

binding.btnEditName.setOnClickListener {
    if (isEditMode) {
        val currentName =
binding.tvProfileName.text.toString()
        val currentNickname =
sharedPrefs.getString("USER_NAME", "unknown") ?: "unknown"
        val currentEmail =
binding.tvProfileEmail.text.toString()
        showEditDialog("Изменить имя", currentName) {
newValue ->
            viewModel.updateProfileInfo(profileUserId,
newValue, currentNickname, currentEmail)
        }
    }
}

binding.btnEditNickname.setOnClickListener {
    if (isEditMode) {
        val currentName =
binding.tvProfileName.text.toString()
        val currentNickname =
sharedPrefs.getString("USER_NAME", "unknown") ?: "unknown"
        val currentEmail =
binding.tvProfileEmail.text.toString()
        showEditDialog("Изменить никнейм",
currentNickname) { newValue ->
            viewModel.updateProfileInfo(profileUserId,
currentName, newValue, currentEmail)
        }
    }
}

binding.btnEditEmail.setOnClickListener {
    if (isEditMode) {
        val currentName =
binding.tvProfileName.text.toString()
        val currentNickname =
sharedPrefs.getString("USER_NAME", "unknown") ?: "unknown"

```

```

        val currentEmail =
binding.tvProfileEmail.text.toString()
        showEditDialog("Изменить Email", currentEmail) {
newValue ->
            viewModel.updateProfileInfo(profileUserId,
currentName, currentNickname, newValue)
        }
    }

binding.btnChangeAvatar.setOnClickListener {
    if (isEditMode) {

pickMedia.launch(PickVisualMediaRequest(ActivityResultContracts.
PickVisualMedia.ImageOnly))
    }

    observeViewModel()
}

private fun showEditDialog(title: String, currentValue:
String, onSave: (String) -> Unit) {
    val context = requireContext()

    val textInputLayout =
com.google.android.material.textfield.TextInputLayout(
        context,
        null,

com.google.android.material.R.style.Widget_Material3_TextInputLa
yout_OutlinedBox
    ).apply {
        setPadding(60, 30, 60, 0)
    }

    val input =
com.google.android.material.textfield.TextInputEditText(context)
    .apply {
        setText(currentValue)
        setSelection(currentValue.length)
        isSingleLine = true
    }

    textInputLayout.addView(input)

com.google.android.material.dialog.MaterialAlertDialogBuilder(co
ntext)
        .setTitle(title)
        .setView(textInputLayout)

```

```

        .setPositiveButton("Сохранить") { _, _ ->
            val newValue = input.text.toString().trim()
            if (newValue.isNotEmpty() && newValue !=
currentValue) {
                onSave(newValue)
            }
        }
        .setNegativeButton("Отмена", null)
        .show()
    }

    private fun updateProfileUI() {
        val sharedPrefs =
requireContext().getSharedPreferences("MessengerPrefs",
Context.MODE_PRIVATE)
        val nickname = sharedPrefs.getString("USER_NAME",
"unknown") ?: "unknown"
        val fullName = sharedPrefs.getString("USER_FIO", "Имя не
указано") ?: "Имя не указано"
        val email = sharedPrefs.getString("USER_EMAIL", "Почта
не указана") ?: "Почта не указана"
        val faculty = sharedPrefs.getString("USER_FACULTY",
"Студент") ?: "Студент"
        val group = sharedPrefs.getString("USER_GROUP", "") ?:
""
        val avatarUrl = sharedPrefs.getString("USER_AVATAR",
null)

        binding.tvProfileName.text = fullName
        binding.tvProfileNickname.text = "@$nickname"
        binding.tvProfileEmail.text = email
        binding.tvProfileFaculty.text = "Факультет: $faculty"
        binding.tvProfileGroup.text = "Группа: $group"

        binding.ivProfileAvatar.loadAvatarOrDefault(avatarUrl,
fullName)
    }

    private fun observeViewModel() {
        viewLifecycleOwner.lifecycleScope.launch {
viewLifecycleOwner.repeatOnLifecycle(Lifecycle.State.STARTED) {
            launch {
                viewModel.uploadState.collect { state ->
                    when (state) {
                        is
ProfileViewModel.UploadState.Loading -> {
binding.progressBarAvatar.visibility = View.VISIBLE

binding.btnChangeAvatar.isEnabled = false

```

```

        }
        is
ProfileViewModel.UploadState.Success -> {

binding.progressBarAvatar.visibility = View.GONE

binding.btnChangeAvatar.isEnabled = true

binding.ivProfileAvatar.loadAvatarOrDefault(state.newUrl,
binding.tvProfileName.text.toString())

        val sharedPrefs =
requireContext().getSharedPreferences("MessengerPrefs",
Context.MODE_PRIVATE)

sharedPrefs.edit().putString("USER_AVATAR",
state.newUrl).apply()

        Toast.makeText(requireContext(),
"Аватар обновлен!", Toast.LENGTH_SHORT).show()
        viewModel.resetState()
    }
    is
ProfileViewModel.UploadState.Error -> {

binding.progressBarAvatar.visibility = View.GONE

binding.btnChangeAvatar.isEnabled = true
        Toast.makeText(requireContext(),
state.message, Toast.LENGTH_SHORT).show()
        viewModel.resetState()
    }
    is ProfileViewModel.UploadState.Idle
-> {}

    }
}
}
launch {
    viewModel.updateInfoState.collect { state ->
        when (state) {
            is
ProfileViewModel.UpdateInfoState.Loading -> {

binding.progressBarAvatar.visibility = View.VISIBLE
        }
        is
ProfileViewModel.UpdateInfoState.Success -> {

binding.progressBarAvatar.visibility = View.GONE
        binding.tvProfileName.text =
state.fullName

```

```

binding.tvProfileNickname.text =
"@${state.username}"

binding.tvProfileEmail.text =
state.email

val sharedPrefs =
requireContext().getSharedPreferences("MessengerPrefs",
Context.MODE_PRIVATE)

sharedPrefs.edit()
    .putString("USER_FIO",
state.fullName)
    .putString("USER_NAME",
state.username)
    .putString("USER_EMAIL",
state.email)
    .apply()

Toast.makeText(requireContext(),
"Данные обновлены", Toast.LENGTH_SHORT).show()
viewModel.resetUpdateInfoState()
    }
    is
ProfileViewModel.UpdateInfoState.Error -> {
binding.progressBarAvatar.visibility = View.GONE
    Toast.makeText(requireContext(),
state.message, Toast.LENGTH_LONG).show()
    viewModel.resetUpdateInfoState()
    }
    is
ProfileViewModel.UpdateInfoState.Idle -> {}
    }
    }
    }
    }
    }

override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
}
}

```

Файл «ChatActivity.kt»

```

package com.example.tiltedgram.chat

import android.content.Context
import android.content.Intent
import android.os.Build
import android.os.Bundle

```

```

import android.provider.OpenableColumns
import android.view.Gravity
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.NotificationCompat
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.lifecycleScope
import androidx.lifecycle.repeatOnLifecycle
import androidx.recyclerview.widget.LinearLayoutManager
import coil.load
import com.example.tiltedgram.R
import com.example.tiltedgram.databinding.ActivityChatBinding
import com.example.tiltedgram.group.GroupProfileActivity
import com.example.tiltedgram.profile.ProfileActivity
import kotlinx.coroutines.launch
import androidx.core.widget.doAfterTextChanged
import android.app.NotificationManager
import com.example.tiltedgram.utils.loadAvatarOrDefault

```

```

class ChatActivity : AppCompatActivity() {

    private lateinit var binding: ActivityChatBinding
    private val viewModel: ChatViewModel by viewModels()

    private var currentUserId: Long = -1L
    private var receiverId: Long = -1L
    private var groupId: Long? = null
    private var isGroup: Boolean = false
    private var chatAdminId: Long = -1L
    private var currentChatName: String = ""
    private var currentAvatarUrl: String? = null
    private var currentMessagesLimit = 50

    private val UPLOAD_CHANNEL_ID = "upload_channel"
    private val UPLOAD_NOTIFICATION_ID = 1001
    private lateinit var notificationManager:
NotificationManager
    private lateinit var notificationBuilder:
NotificationCompat.Builder

    private var isInitialScrollDone = false
    private var forceScrollToBottom = false
    private lateinit var messageAdapter: MessageAdapter

    private var localSearchResults = listOf<Long>()

```

```

private var currentSearchIndex = -1

private var newMessagesCount = 0
private var lastProcessedMessageId: Long = -1L

private val pickFileLauncher =
registerForActivityResult(androidx.activity.result.contract.Acti
vityResultContracts.OpenDocument()) { uri ->
    if (uri != null) {
        Toast.makeText(this, "Отправка файла. Пожалуйста,
подождите...", Toast.LENGTH_SHORT).show()

        // file real name
        var fileName = "document"
        val cursor = contentResolver.query(uri, null, null,
null, null)
        cursor?.use {
            if (it.moveToFirst()) {
                val nameIndex =
it.getColumnIndex(OpenableColumns.DISPLAY_NAME)
                if (nameIndex != -1) fileName =
it.getString(nameIndex)
            }
        }

        // file type
        val mimeType = contentResolver.getType(uri) ?:
"application/octet-stream"

        val text = binding.etMessage.text.toString().trim()
        val rId = if (isGroup) null else receiverId
        val gId = if (isGroup) groupId else null

        viewModel.uploadAndSendMessage(currentUserId, rId,
gId, text, uri, fileName, mimeType, contentResolver)

        binding.etMessage.text.clear()
    }
}

override fun onCreate(savedInstanceState: Bundle?) {

window.requestFeature(android.view.Window.FEATURE_ACTIVITY_TRANS
ITIONS)

    val zAxisEnter =
com.google.android.material.transition.platform.MaterialSharedAx
is(com.google.android.material.transition.platform.MaterialShare
dAxis.Z, true)
    val zAxisReturn =
com.google.android.material.transition.platform.MaterialSharedAx

```

```

is(com.google.android.material.transition.platform.MaterialShare
dAxis.Z, false)
    window.enterTransition = zAxisEnter
    window.returnTransition = zAxisReturn
    window.sharedElementsUseOverlay = false

setExitSharedElementCallback(com.google.android.material.transit
ion.platform.MaterialContainerTransformSharedElementCallback())

    super.onCreate(savedInstanceState)
    binding = ActivityChatBinding.inflate(layoutInflater)
    setContentView(binding.root)

    notificationManager =
getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager

        // channel noty
        if (android.os.Build.VERSION.SDK_INT >=
android.os.Build.VERSION_CODES.O) {
            val channel = android.app.NotificationChannel(
                UPLOAD_CHANNEL_ID,
                "Отправка файлов",
                NotificationManager.IMPORTANCE_LOW
            )

notificationManager.createNotificationChannel(channel)
        }

        notificationBuilder = NotificationCompat.Builder(this,
UPLOAD_CHANNEL_ID)
            .setSmallIcon(android.R.drawable.stat_sys_upload)
            .setOngoing(true)
            .setOnlyAlertOnce(true)

        // padding
        ViewCompat.setOnApplyWindowInsetsListener(binding.root)
{ v, insets ->
            val topInset =
insets.getInsets(WindowInsetsCompat.Type.statusBars()).top
            val bottomInset =
insets.getInsets(WindowInsetsCompat.Type.navigationBars() or
WindowInsetsCompat.Type.ime()).bottom

            binding.chatAppBarLayout.setPadding(0, topInset, 0,
0)

            v.setPadding(0, 0, 0, bottomInset)
            insets
        }

        // get id

```

					ПГУ1.090301.37.001 ПЗ	Лист
						116
Изм.	Лист	№ докум.	Подпись	Дата		

```

        val sharedPrefs = getSharedPreferences("MessengerPrefs",
MODE_PRIVATE)
        currentUserId = sharedPrefs.getLong("USER_ID", -1L)
        val targetId = intent.getLongExtra("RECEIVER_ID", -1L)

        isGroup = intent.getBooleanExtra("IS_GROUP", false)
        receiverId = if (isGroup) -1L else targetId
        groupId = if (isGroup) targetId else null

        viewModel.startPolling(currentUserId, receiverId,
groupId, isGroup, currentMessagesLimit)

        // toolbar / Извлечение данных (безопасно для пушей и
адаптера)
        val receiverAvatar =
intent.getStringExtra("RECEIVER_AVATAR")
        val receiverNickname =
intent.getStringExtra("RECEIVER_NAME") ?: "Контакт"
        val receiverFio = intent.getStringExtra("RECEIVER_FIO")
?: "Контакт"
        val receiverEmail =
intent.getStringExtra("RECEIVER_EMAIL") ?: "Скрыта"
        val receiverFaculty =
intent.getStringExtra("RECEIVER_FACULTY") ?: "Не указан"
        val receiverGroup =
intent.getStringExtra("RECEIVER_GROUP") ?: "Не указана"
        chatAdminId = intent.getLongExtra("ADMIN_ID", -1L)

        currentChatName = receiverFio
        currentAvatarUrl = receiverAvatar

        binding.tvChatName.text = receiverFio

        if (isGroup) {
            binding.tvChatStatus.text = "Загрузка участников..."
        } else {
            binding.tvChatStatus.text = if
(!receiverGroup.isNullOrBlank() && receiverGroup != "null") {
                receiverGroup
            } else {
                "Личный чат"
            }
        }

        binding.ivChatAvatar.loadAvatarOrDefault(receiverAvatar,
receiverFio)
        binding.btnBack.setOnClickListener {
finishAfterTransition() }
        binding.btnAttach.setOnClickListener {
            pickFileLauncher.launch(arrayOf("*/*"))
        }

```

```

        // down messages
        val layoutManager = LinearLayoutManager(this)
        layoutManager.stackFromEnd = true
        binding.rvMessages.layoutManager = layoutManager

        val profileTransitionName =
            "profile_transform_$targetId"
        binding.llNormalToolbar.transitionName =
            profileTransitionName

        val openProfile = {
            val targetIntent = if (!isGroup) {
                Intent(this@ChatActivity,
                    ProfileActivity::class.java).apply {
                    putExtra("USER_ID", targetId)
                    putExtra("USER_NAME", receiverNickname)
                    putExtra("USER_FIO", currentChatName)
                    putExtra("USER_EMAIL", receiverEmail)
                    putExtra("USER_AVATAR", currentAvatarUrl)
                    putExtra("USER_FACULTY", receiverFaculty)
                    putExtra("USER_GROUP", receiverGroup)
                    putExtra("EXTRA_TRANSITION_NAME",
                        profileTransitionName)
                }
            } else {
                Intent(this@ChatActivity,
                    GroupProfileActivity::class.java).apply {
                    putExtra("GROUP_ID", targetId)
                    putExtra("GROUP_NAME", currentChatName)
                    putExtra("GROUP_AVATAR", currentAvatarUrl)
                    putExtra("ADMIN_ID", chatAdminId)
                    putExtra("EXTRA_TRANSITION_NAME",
                        profileTransitionName)
                }
            }

            val options =
                androidx.core.app.ActivityOptionsCompat.makeSceneTransitionAnimation(
                    this@ChatActivity,
                    binding.llNormalToolbar,
                    profileTransitionName
                )
            startActivity(targetIntent, options.toBundle())
        }

        binding.ivChatAvatar.setOnClickListener { openProfile() }
        binding.tvChatName.setOnClickListener { openProfile() }
    }

```

```

        binding.tvChatStatus.setOnClickListener { openProfile()
    }

    // send
    binding.btnSend.setOnClickListener {
        val text = binding.etMessage.text.toString().trim()
        if (text.isNotEmpty() && currentUserId != -1L &&
targetId != -1L) {
            val rId = if (isGroup) null else targetId
            val gId = if (isGroup) targetId else null
            viewModel.sendMessage(currentUserId, rId, gId,
text)

            binding.etMessage.text.clear()
        }
    }

    binding.btnSearchChat.setOnClickListener {
        val transition =
com.google.android.material.transition.MaterialSharedAxis(
com.google.android.material.transition.MaterialSharedAxis.Z,
        true
    )

        androidx.transition.TransitionManager.beginDelayedTransition(binding.chatAppBarLayout, transition)

        binding.llNormalToolbar.visibility = View.GONE
        binding.llSearchToolbar.visibility = View.VISIBLE
        binding.etChatSearch.requestFocus()

        val imm =
getSystemService(Context.INPUT_METHOD_SERVICE) as
android.view.inputmethod.InputMethodManager
        imm.showSoftInput(binding.etChatSearch,
android.view.inputmethod.InputMethodManager.SHOW_IMPLICIT)
    }

    binding.btnCloseSearch.setOnClickListener {
        closeSearch()
    }

    binding.etChatSearch.doAfterTextChanged { text ->
        val query = text.toString().trim()
        if (query.isEmpty() ||
!::messageAdapter.isInitialized) {
            localSearchResults = emptyList()
            currentSearchIndex = -1
            binding.tvSearchCount.text = ""
        }
    }

```

```

        if (::messageAdapter.isInitialized)
messageAdapter.clearTargetMessage()
        return@doAfterTextChanged
    }

    val allMessages = viewModel.messages.value
    localSearchResults = allMessages.filter {
it.content.contains(query, ignoreCase = true) }.map { it.id }

    if (localSearchResults.isNotEmpty()) {
        currentSearchIndex = localSearchResults.size - 1
        updateSearchUI()
    } else {
        currentSearchIndex = -1
        binding.tvSearchCount.text = "0/0"
        messageAdapter.clearTargetMessage()
    }
}

binding.btnSearchUp.setOnClickListener {
    if (currentSearchIndex > 0){
        currentSearchIndex--
        updateSearchUI()
    }
}

binding.btnSearchDown.setOnClickListener {
    if (currentSearchIndex < localSearchResults.size -
1) {
        currentSearchIndex++
        updateSearchUI()
    }
}

binding.cvScrollDownIndicator.setOnClickListener {
    if (::messageAdapter.isInitialized &&
messageAdapter.itemCount > 0) {

binding.rvMessages.scrollToPosition(messageAdapter.itemCount -
1)

        }
        newMessagesCount = 0
        binding.cvScrollDownIndicator.visibility = View.GONE
    }

    observeViewModel()

    binding.rvMessages.addOnScrollListener(object :
androidx.recyclerview.widget.RecyclerView.OnScrollListener() {
        private var isLoadingMore = false

```

```

        override fun onScrolled(recyclerView:
androidx.recyclerview.widget.RecyclerView, dx: Int, dy: Int) {
            val lm = recyclerView.layoutManager as
LinearLayoutManager
                val lastVisiblePosition =
lm.findLastVisibleItemPosition()
                val isAtBottom = lastVisiblePosition >=
(recyclerView.adapter?.itemCount ?: 0) - 2

com.example.tiltedgram.utils.ActiveChatManager.isAtBottom =
isAtBottom

                if (isAtBottom) {
                    newMessagesCount = 0
                    binding.cvScrollDownIndicator.visibility =
View.GONE
                }

                val firstVisiblePosition =
lm.findFirstVisibleItemPosition()

                if (firstVisiblePosition == 0 && dy < 0 &&
!isLoadingMore) {
                    val currentTotalItems =
messageAdapter.itemCount

                    if (currentTotalItems >=
currentMessagesLimit) {
                        isLoadingMore = true
                        currentMessagesLimit += 50
                        viewModel.startPolling(currentUserId,
receiverId, groupId, isGroup, currentMessagesLimit, forceRefresh
= true)

                        binding.rvMessages.postDelayed({
isLoadingMore = false }, 1000)
                    }
                }
            })
        }

        private fun updateSearchUI() {
            if (localSearchResults.isEmpty() || currentSearchIndex
!in localSearchResults.indices) return

            binding.tvSearchCount.text = "${currentSearchIndex + 1}
из ${localSearchResults.size}"
            val targetId = localSearchResults[currentSearchIndex]

            if (::messageAdapter.isInitialized) {

```

```

        messageAdapter.setTarget(targetId)
    }

    val targetPos = viewModel.messages.value.indexOfFirst {
it.id == targetId }
    if (targetPos != -1) {
        binding.rvMessages.post {
            val smoothScroller = object :
androidx.recyclerview.widget.LinearSmoothScroller(this) {
                override fun calculateDtToFit(viewStart:
Int, viewEnd: Int, boxStart: Int, boxEnd: Int, snapPreference:
Int): Int {
                    val boxCenter = boxStart + (boxEnd -
boxStart) / 2
                    val viewCenter = viewStart + (viewEnd -
viewStart) / 2
                    return boxCenter - viewCenter
                }

                override fun calculateTimeForScrolling(dx:
Int): Int {
                    return
super.calculateTimeForScrolling(dx) * 2
                }
            }
            smoothScroller.targetPosition = targetPos

binding.rvMessages.layoutManager?.startSmoothScroll(smoothScroll
er)

        }
    }

    private fun observeViewModel(){
        lifecycleScope.launch {
            repeatOnLifecycle(Lifecycle.State.STARTED){
                launch {
                    viewModel.isSending.collect { isSending ->
                        binding.btnSend.isEnabled = !isSending
                    }
                }
                launch {
                    viewModel.messageSentStatus.collect { status
->
                        when(status) {
                            true -> {
                                forceScrollToBottom = true
                                if
(::messageAdapter.isInitialized) {
messageAdapter.clearTargetMessage()

```

```

        }
        closeSearch()

viewModel.startPolling(currentUserId, receiverId, groupId,
    isGroup, currentMessagesLimit, forceRefresh = true)
        viewModel.resetSendStatus()
    }
    false -> {

Toast.makeText(this@ChatActivity, "Ошибка отправки сообщения",
    Toast.LENGTH_SHORT).show()

        viewModel.resetSendStatus()
    }
    null -> {}
}
}
}
launch {
    viewModel.messages.collect { messageList ->
        if (!::messageAdapter.initialized) {
            val targetMessageId =
intent.getLongExtra("SCROLL_TO_MESSAGE_ID", -1L)
            messageAdapter = MessageAdapter(
                messages = emptyList(),
                currentUserId = currentUserId,
                isGroup = isGroup,
                adminId = chatAdminId,
                targetMessageId =

targetMessageId,

                onEditClicked = { msgId, newText

->

                    viewModel.editMessage(msgId,

currentUserId, newText)

                },
                onDeleteClicked = { msgId ->

viewModel.deleteMessage(msgId, currentUserId)

                },
                onSenderAvatarClicked = {

senderId ->

                    lifecycleScope.launch {
                        try {
                            val res =
com.example.tiltedgram.data.NetworkModule.authApi.getUserInfo(se
nderId)

                            if (res.isSuccessful

&& res.body() != null) {

                                val user =
res.body()!!

```

```

                                                                    val
profileIntent = Intent(this@ChatActivity,
ProfileActivity::class.java).apply {

putExtra("USER_ID", user.id)

putExtra("USER_NAME", user.username)

putExtra("USER_FIO", user.fullName)

putExtra("USER_AVATAR", user.avatarUrl)

putExtra("USER_FACULTY", user.faculty)

putExtra("USER_GROUP", user.group)
                                                                    }

startActivity(profileIntent)
                                                                    }
                                                                    } catch (e: Exception) {
                                                                    e.printStackTrace()
                                                                    }
                                                                    }
                                                                    }
                                                                    )
                                                                    binding.rvMessages.adapter =
messageAdapter
                                                                    }

                                                                    if (messageList.isEmpty())
return@collect

                                                                    val lastMessage =
messageList.lastOrNull()
                                                                    if (lastMessage != null &&
lastProcessedMessageId != -1L && lastMessage.id >
lastProcessedMessageId) {
                                                                    val lm =
binding.rvMessages.layoutManager as LinearLayoutManager
                                                                    val lastVisiblePosition =
lm.findLastVisibleItemPosition()
                                                                    val isAtBottom = lastVisiblePosition
>= messageAdapter.itemCount - 2

                                                                    if (lastMessage.senderId !=
currentUserId && !isAtBottom && !forceScrollToBottom) {
                                                                    newMessagesCount++
                                                                    binding.tvNewMessagesCount.text
= newMessagesCount.toString()

                                                                    binding.cvScrollDownIndicator.visibility = View.VISIBLE

```

```

        }
    }
    if (lastMessage != null) {
        lastProcessedMessageId =
lastMessage.id
    }

    if (!isInitialScrollDone) {

messageAdapter.updateData(messageList)

        val targetMessageId =
intent.getLongExtra("SCROLL_TO_MESSAGE_ID", -1L)
        if (targetMessageId != -1L) {
            val targetIndex =
messageList.indexOfFirst { it.id == targetMessageId }
            if (targetIndex != -1) {

binding.rvMessages.postDelayed({
                                val layoutManager =
binding.rvMessages.layoutManager as LinearLayoutManager

layoutManager.scrollToPositionWithOffset(targetIndex, 300)
                                }, 150)
            } else {

binding.rvMessages.scrollToPosition(messageList.size - 1)
            }
            } else {

binding.rvMessages.scrollToPosition(messageList.size - 1)
            }
            isInitialScrollDone = true
        } else {
            val layoutManager =
binding.rvMessages.layoutManager as LinearLayoutManager
            val lastVisiblePosition =
layoutManager.findLastVisibleItemPosition()
            val isAtBottom = lastVisiblePosition
            >= messageAdapter.itemCount - 2

messageAdapter.updateData(messageList)

            if (isAtBottom ||
forceScrollToBottom) {

binding.rvMessages.scrollToPosition(messageList.size - 1)
                forceScrollToBottom = false
                newMessagesCount = 0
            }
        }
    }
}

```

```

binding.cvScrollDownIndicator.visibility = View.GONE
    }
    }
}
launch {
    viewModel.kickedEvent.collect { isKicked ->
        if (isKicked){
            Toast.makeText(this@ChatActivity,
"Вы были исключены из группы", Toast.LENGTH_LONG).show()
            finishAfterTransition()
        }
    }
}
launch {
    viewModel.headerState.collect { header ->
        if (header.name.isNotEmpty()) {
            binding.tvChatName.text =
header.name
            binding.tvChatStatus.text =
header.status

            if (currentAvatarUrl !=
header.avatarUrl || binding.ivChatAvatar.drawable == null) {
                currentAvatarUrl =
header.avatarUrl

binding.ivChatAvatar.loadAvatarOrDefault(header.avatarUrl,
header.name)
            }
        }
    }
}
launch {
    viewModel.uploadState.collect { state ->
        if (state.isUploading) {
            notificationBuilder
                .setContentTitle("Отправка:
${state.fileName}")

            .setContentText("${state.progress}%")
                .setProgress(100,
state.progress, false)

notificationManager.notify(UPLOAD_NOTIFICATION_ID,
notificationBuilder.build())
        } else {

notificationManager.cancel(UPLOAD_NOTIFICATION_ID)

```

```

    }
    }
    }
    }

    private fun closeSearch() {
        if (binding.llSearchToolbar.visibility == View.VISIBLE)
        {
            val transition =
com.google.android.material.transition.MaterialSharedAxis(
com.google.android.material.transition.MaterialSharedAxis.Z,
            false
        )

        androidx.transition.TransitionManager.beginDelayedTransition(bin
ding.chatAppBarLayout, transition)

            binding.llSearchToolbar.visibility = View.GONE
            binding.llNormalToolbar.visibility = View.VISIBLE
            binding.etChatSearch.text.clear()

            val imm =
getSystemService(Context.INPUT_METHOD_SERVICE) as
android.view.inputmethod.InputMethodManager

            imm.hideSoftInputFromWindow(binding.etChatSearch.windowToken, 0)

                if (::messageAdapter.isInitialized) {
                    messageAdapter.clearTargetMessage()
                }
            }

            override fun onResume() {
                super.onResume()
                val targetId = intent.getLongExtra("RECEIVER_ID", -1L)

com.example.tiltedgram.utils.ActiveChatManager.activeChatId =
targetId

com.example.tiltedgram.utils.ActiveChatManager.isAtBottom = true
                viewModel.startPolling(currentUserId, receiverId,
groupId, isGroup, currentMessagesLimit)
            }

            override fun onPause() {
                super.onPause()

```

					ПГУ1.090301.37.001 ПЗ	Лист
						127
Изм.	Лист	№ докум.	Подпись	Дата		

```

com.example.tiltedgram.utils.ActiveChatManager.activeChatId = -
1L
    }

    override fun onDestroy() {
        super.onDestroy()
        viewModel.stopPolling()
    }
}

```

Файл «ChatViewModel.kt»

```

package com.example.tiltedgram.chat

import android.content.ContentResolver
import android.net.Uri
import android.util.Log
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.tiltedgram.data.MessageDto
import com.example.tiltedgram.data.NetworkModule
import com.example.tiltedgram.data.SendMessageRequest
import com.example.tiltedgram.data.UriRequestBody
import kotlinx.coroutines.Job
import kotlinx.coroutines.async
import kotlinx.coroutines.coroutineScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.isActive
import kotlinx.coroutines.launch
import okhttp3.MediaType.Companion.toMediaTypeOrNull
import okhttp3.MultipartBody

data class ChatHeaderState(
    val name: String = "",
    val avatarUrl: String? = null,
    val status: String = ""
)

data class UploadState(val isUploading: Boolean, val progress:
Int, val fileName: String = "")
class ChatViewModel : ViewModel() {
    private val _uploadState =
MutableStateFlow(UploadState(false, 0))
    val uploadState = _uploadState.asStateFlow()
    private val _isSending = MutableStateFlow(false)
    val isSending: StateFlow<Boolean> = _isSending.asStateFlow()
    private val _messageSentStatus =
MutableStateFlow<Boolean?>(null)

```

```

        val messageSentStatus: StateFlow<Boolean?> =
        _messageSentStatus.asStateFlow()
        private val _messages =
MutableStateFlow<List<MessageDto>>(emptyList())
        val messages: StateFlow<List<MessageDto>> =
        _messages.asStateFlow()
        private val _kickedEvent = MutableStateFlow(false)
        val kickedEvent: StateFlow<Boolean> =
        _kickedEvent.asStateFlow()
        private val _headerState =
MutableStateFlow(ChatHeaderState())
        val headerState = _headerState.asStateFlow()
        private var pollingJob: Job? = null
        private var currentParams: String? = null

fun uploadAndSendMessage(
    senderId: Long, receiverId: Long?, groupId: Long?,
    content: String, fileUri: Uri, fileName: String,
    mimeType: String, contentResolver: ContentResolver
) {
    viewModelScope.launch {
        _isSending.value = true
        _uploadState.value = UploadState(true, 0, fileName)

        try {
            val body =
com.example.tiltedgram.data.UriRequestBody(contentResolver,
fileUri, mimeType.toMediaTypeOrNull()) { progress ->
                _uploadState.value = UploadState(true,
progress, fileName)
            }

            val part =
MultipartBody.Part.createFormData("file", fileName, body)
            val uploadResponse =
com.example.tiltedgram.data.NetworkModule.authApi.uploadAttachme
nt(part)

            if (uploadResponse.isSuccessful &&
uploadResponse.body() != null) {
                val attachmentInfo = uploadResponse.body()!!

                val request =
com.example.tiltedgram.data.SendMessageRequest(
                    senderId, receiverId, groupId, content,
                    attachmentInfo.url, attachmentInfo.name,
                    attachmentInfo.size, attachmentInfo.type
                )
                val sendResponse =
com.example.tiltedgram.data.NetworkModule.authApi.sendMessage(re
quest)

```

```

        _messageSentStatus.value =
sendResponse.isSuccessful
    } else {
        _messageSentStatus.value = false
    }
    } catch (e: Exception) {
        e.printStackTrace()
        _messageSentStatus.value = false
    } finally {
        _isSending.value = false
        _uploadState.value = UploadState(false, 0)
    }
}

}

fun sendMessage (senderId: Long, receiverId: Long?, groupId:
Long?, content: String){
    viewModelScope.launch {
        _isSending.value = true
        try {
            val request = SendMessageRequest(senderId,
receiverId, groupId, content)
            val response =
NetworkModule.authApi.sendMessage(request)

            if (response.isSuccessful) {
                _messageSentStatus.value = true
            } else {
                _messageSentStatus.value = false
            }
        } catch (e: Exception) {
            _messageSentStatus.value = false
        } finally {
            _isSending.value = false
        }
    }
}

fun resetSendStatus(){
    _messageSentStatus.value = null
}

fun editMessage(messageId: Long, userId: Long, newContent:
String) {
    viewModelScope.launch {
        try {
            val request =
com.example.tiltedgram.data.EditMessageRequest(userId,
newContent)

            val response =
NetworkModule.authApi.editMessage(messageId, request)
            if (!response.isSuccessful) {

```

```

        Log.e("MyLog", "Ошибка редактирования:
${response.code()}")
    }
    } catch (e: Exception) {
        e.printStackTrace()
    }
}

fun deleteMessage(messageId: Long, userId: Long) {
    viewModelScope.launch {
        try {
            val response =
NetworkModule.authApi.deleteMessage(messageId, userId)
            if (!response.isSuccessful) {
                Log.e("MyLog", "Ошибка удаления:
${response.code()}")
            }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}

fun startPolling(myId: Long, receiverId: Long, groupId:
Long?, isGroup: Boolean, limit: Int, forceRefresh: Boolean =
false) {
    val newParams = "$myId-$receiverId-$groupId-$isGroup-
$limit"

    if (!forceRefresh && pollingJob?.isActive == true &&
currentParams == newParams) {
        return
    }

    currentParams = newParams
    pollingJob?.cancel()
    pollingJob = viewModelScope.launch {
        while (isActive) {
            launch {
                try {
                    val msgRes = if (isGroup && groupId !=
null) {
NetworkModule.authApi.getGroupMessages(groupId, myId, limit)
                    } else {
NetworkModule.authApi.getChatHistory(myId, receiverId, limit)
                    }

                    if (msgRes.isSuccessful) {
                        @Suppress("UNCHECKED_CAST")

```

```

        _messages.value = (msgRes.body() as?
List<MessageDto>) ?: emptyList()
        } else if (msgRes.code() == 403 &&
isGroup) {
            _kickedEvent.value = true
            stopPolling()
        }
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    launch {
        try {
            val headerData = if (isGroup && groupId
!= null) {
                val res =
NetworkModule.authApi.getGroupInfo(groupId)
                if (res.isSuccessful) {
                    res.body()?.let {
ChatHeaderState(it.name, it.avatarUrl, "${it.memberCount}
участников") }

                    } else null
                } else {
                    val res =
NetworkModule.authApi.getUserInfo(receiverId)
                    if (res.isSuccessful) {
                        res.body()?.let {
ChatHeaderState(it.fullName, it.avatarUrl, it.group) }

                        } else null
                    }

                    if (headerData != null) {
                        _headerState.value = headerData
                    }
                } catch (e: Exception) {
                    e.printStackTrace()
                }
            }

            delay(3000)
        }
    }

    fun stopPolling() {
        pollingJob?.cancel()
        pollingJob = null
        currentParams = null
    }
}

```

Файл «MessageAdapter.kt»

```
package com.example.tiltedgram.chat

import android.app.DownloadManager
import android.content.ClipData
import android.content.Context
import android.content.Intent
import android.net.Uri
import android.os.Environment
import android.view.Gravity
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.webkit.MimeTypeMap
import android.widget.Toast
import androidx.core.content.ContextCompat
import androidx.core.content.FileProvider
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import coil.load
import com.example.tiltedgram.R
import com.example.tiltedgram.data.MessageDto
import com.example.tiltedgram.databinding.ItemMessageBinding
import com.example.tiltedgram.utils.loadAvatarOrDefault
import com.google.android.material.color.MaterialColors
import
com.google.android.material.dialog.MaterialAlertDialogBuilder
import kotlinx.coroutines.*
import java.io.File
import java.time.LocalDate
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter
import java.util.Locale

class MessageAdapter(
    private var messages: List<MessageDto>,
    private val currentUserId: Long,
    private val isGroup: Boolean,
    private val adminId: Long = -1L,
    private var targetMessageId: Long = -1L,
    private val onEditClicked: (messageId: Long, newContent:
String) -> Unit,
    private val onDeleteClicked: (messageId: Long) -> Unit,
    private val onSenderAvatarClicked: (Long) -> Unit
) : RecyclerView.Adapter<MessageAdapter.MessageViewHolder>() {

    private val downloadTrackingJobs = mutableMapOf<Long, Job>()

    class MessageViewHolder(val binding: ItemMessageBinding) :
RecyclerView.ViewHolder(binding.root) {
```

```

        var colorAnimator: android.animation.ValueAnimator? =
null
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): MessageViewHolder {
        val binding =
ItemMessageBinding.inflate(LayoutInflater.from(parent.context),
parent, false)
        return MessageViewHolder(binding)
    }

    private fun isFileDownloaded(fileName: String?,
expectedSize: Long?): Boolean {
        if (fileName.isNullOrEmpty() || expectedSize == null)
return false
        val downloadFolder =
Environment.getExternalStoragePublicDirectory(Environment.DIRECT
ORY_DOWNLOADS)
        val file = File(downloadFolder, fileName)
        return file.exists() && file.length() == expectedSize
    }

    private fun formatSize(bytes: Long): String {
        return if (bytes < 1024) "$bytes B"
        else if (bytes < 1024 * 1024) String.format(Locale.US,
"%1f KB", bytes / 1024.0)
        else String.format(Locale.US, "%1f MB", bytes / (1024.0
* 1024.0))
    }

    private fun setFileOpenClickListener(holder:
MessageViewHolder, message: MessageDto) {
        holder.binding.llAttachmentDoc.setOnClickListener { view
->
            val fileName = message.attachmentName ?: "file"
            val file =
File(Environment.getExternalStoragePublicDirectory(Environment.D
IRECTORY_DOWNLOADS), fileName)
            try {
                val uri =
FileProvider.getUriForFile(view.context,
"${view.context.packageName}.provider", file)
                val ext = file.extension.lowercase()
                val mimeType =
MimeTypeMap.getSingleton().getMimeTypeFromExtension(ext) ?:
"*/*"

                val intent = Intent(Intent.ACTION_VIEW).apply {
                    setDataAndType(uri, mimeType)
                }
            }
        }
    }

```

```

addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION or
Intent.FLAG_GRANT_WRITE_URI_PERMISSION)
        addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
        clipData = ClipData.newRawUri("", uri)
    }
    view.context.startActivity(intent)
} catch (e: Exception) {
    e.printStackTrace()
    Toast.makeText(view.context, "Нет приложения для
открытия", Toast.LENGTH_SHORT).show()
}
}

private fun trackDownloadProgress(context: Context, message:
MessageDto, holder: MessageViewHolder) {
    downloadTrackingJobs[message.id]?.cancel()

    downloadTrackingJobs[message.id] =
CoroutineScope(Dispatchers.Main).launch {
        val dm =
context.getSystemService(Context.DOWNLOAD_SERVICE) as
DownloadManager

        while (isActive) {
            var isDownloading = false
            var bytesDownloaded = 0L
            var bytesTotal = message.attachmentSize ?: 0L

            val query =
DownloadManager.Query().setFilterByStatus(DownloadManager.STATUS
_RUNNING or DownloadManager.STATUS_PENDING)
            val cursor = dm.query(query)

            if (cursor != null && cursor.moveToFirst()) {
                do {
                    val titleIdx =
cursor.getColumnIndex(DownloadManager.COLUMN_TITLE)
                    if (titleIdx >= 0 &&
cursor.getString(titleIdx) == message.attachmentName) {
                        isDownloading = true
                        val downIdx =
cursor.getColumnIndex(DownloadManager.COLUMN_BYTES_DOWNLOADED_SO
_FAR)
                        val totIdx =
cursor.getColumnIndex(DownloadManager.COLUMN_TOTAL_SIZE_BYTES)

                        if (downIdx >= 0) bytesDownloaded =
cursor.getLong(downIdx)

```

```

        if (totIdx >= 0 &&
cursor.getLong(totIdx) > 0) bytesTotal = cursor.getLong(totIdx)
        break
    }
    } while (cursor.moveToNext())
    cursor.close()
}

if (isDownloading) {
    //process
    holder.binding.ivDocIcon.visibility =
View.GONE
    holder.binding.pbDocLoading.visibility =
View.VISIBLE
    holder.binding.tvDocSize.text =
"${formatSize(bytesDownloaded)} / ${formatSize(bytesTotal)}"
    } else {
        if (isFileDownloaded(message.attachmentName,
message.attachmentSize)) {
            //file gone
            holder.binding.pbDocLoading.visibility =
View.GONE
            holder.binding.ivDocIcon.visibility =
View.VISIBLE

            holder.binding.ivDocIcon.setImageResource(R.drawable.ic_file)
            holder.binding.tvDocSize.text =
formatSize(message.attachmentSize ?: 0L)

            setFileOpenClickListener(holder,
message)

            cancel()
        } else {
            //not gone
            holder.binding.pbDocLoading.visibility =
View.GONE
            holder.binding.ivDocIcon.visibility =
View.VISIBLE

            holder.binding.ivDocIcon.setImageResource(R.drawable.ic_save_file)
            holder.binding.tvDocSize.text =
formatSize(message.attachmentSize ?: 0L)

            holder.binding.llAttachmentDoc.setOnClickListener { view ->
                val fixedUrl =
getActiveUrl(message.attachmentUrl)
                startDownload(view.context,
fixedUrl, message.attachmentName ?: "file") {

```

```

holder.binding.ivDocIcon.visibility = View.GONE

holder.binding.pbDocLoading.visibility = View.VISIBLE

trackDownloadProgress(view.context, message, holder)
    }
    }
    cancel()
}
}
    delay(500)
}
}

override fun onBindViewHolder(holder: MessageViewHolder,
position: Int) {
    val message = messages[position]
    holder.binding.tvMessageText.text = message.content
    holder.colorAnimator?.cancel()

    downloadTrackingJobs[message.id]?.cancel()

    val dateTime = try {
        LocalDateTime.parse(message.timestamp)
    } catch (e: Exception){
        LocalDateTime.now()
    }
    val timeFormatter = DateTimeFormatter.ofPattern("HH:mm")
    holder.binding.tvMessageTime.text =
dateTime.format(timeFormatter)

    val showDate: Boolean
    if (position == 0){
        showDate = true
    } else {
        val prevMessageDate =
LocalDateTime.parse(messages[position -
1].timestamp).toLocalDate()
        showDate = dateTime.toLocalDate() != prevMessageDate
    }
    if (showDate){
        holder.binding.tvDateHeader.visibility =
View.VISIBLE
        val isCurrentYear = dateTime.year ==
LocalDate.now().year
        val pattern = if (isCurrentYear) "d MMMM" else "d
MMMM yyyy"
        val dateFormatter =
DateTimeFormatter.ofPattern(pattern, Locale("ru"))

```

```

        holder.binding.tvDateHeader.text =
dateTime.format(dateFormatter)
    } else {
        holder.binding.tvDateHeader.visibility = View.GONE
    }

    val isTargetMessage = message.id == targetMessageId
    val context = holder.itemView.context

    val myColor = MaterialColors.getColor(context,
com.google.android.material.R.attr.colorPrimary,
android.graphics.Color.BLUE)
    val otherColor = MaterialColors.getColor(context,
com.google.android.material.R.attr.colorSurfaceVariant,
android.graphics.Color.GRAY)
    val highlightColor = MaterialColors.getColor(context,
com.google.android.material.R.attr.colorControlHighlight,
android.graphics.Color.LTGRAY)

    val originalColor = if (message.senderId ==
currentUserId) myColor else otherColor

    if (isTargetMessage) {
        holder.colorAnimator =
android.animation.ValueAnimator.ofObject(
            android.animation.ArgbEvaluator(),
            originalColor,
            highlightColor,
            originalColor
        ).apply {
            duration = 1400L
            addUpdateListener { animation ->
                val animatedColor = animation.animatedValue
as Int

holder.binding.cardMessage.setCardBackgroundColor(android.conten
t.res.ColorStateList.valueOf(animatedColor))
            }
            start()
        }
    } else {

holder.binding.cardMessage.setCardBackgroundColor(android.conten
t.res.ColorStateList.valueOf(originalColor))
    }

    if (message.senderId == currentUserId) {
        holder.binding.messageContainer.gravity =
Gravity.END
    }

```

```

holder.binding.tvMessageText.setTextColor(ContextCompat.getColor(
    context, android.R.color.white))

holder.binding.tvMessageTime.setTextColor(ContextCompat.getColor(
    context, android.R.color.white))
    holder.binding.tvMessageTime.alpha = 0.7f

holder.binding.tvDocName.setTextColor(ContextCompat.getColor(con
    text, android.R.color.white))

holder.binding.tvDocSize.setTextColor(ContextCompat.getColor(con
    text, android.R.color.white))
    holder.binding.tvDocSize.alpha = 0.7f
    holder.binding.ivSenderAvatar.visibility = View.GONE
    holder.binding.tvSenderName.visibility = View.GONE
} else {
    holder.binding.messageContainer.gravity =
Gravity.START

holder.binding.tvMessageText.setTextColor(ContextCompat.getColor(
    context, android.R.color.black))

holder.binding.tvMessageTime.setTextColor(ContextCompat.getColor(
    context, android.R.color.black))
    holder.binding.tvMessageTime.alpha = 0.5f

holder.binding.tvDocName.setTextColor(ContextCompat.getColor(con
    text, android.R.color.black))

holder.binding.tvDocSize.setTextColor(ContextCompat.getColor(con
    text, android.R.color.black))
    holder.binding.tvDocSize.alpha = 0.5f

    if (isGroup) {
        holder.binding.ivSenderAvatar.visibility =
View.VISIBLE
        holder.binding.tvSenderName.visibility =
View.VISIBLE
        holder.binding.tvSenderName.text =
message.senderName

holder.binding.ivSenderAvatar.loadAvatarOrDefault(message.sender
    AvatarUrl, message.senderName)

        holder.binding.ivSenderAvatar.setOnClickListener
{ onSenderAvatarClicked(message.senderId) }
        holder.binding.tvSenderName.setOnClickListener {
onSenderAvatarClicked(message.senderId) }

```

```

        } else {
            holder.binding.ivSenderAvatar.visibility =
View.GONE
            holder.binding.tvSenderName.visibility =
View.GONE
        }
    }

    if (!message.attachmentUrl.isNullOrBlank()) {
        if (message.attachmentType == "IMAGE"){
            holder.binding.flAttachmentImage.visibility =
View.VISIBLE
            holder.binding.llAttachmentDoc.visibility =
View.GONE

            val fixedImageUrl =
getActiveUrl(message.attachmentUrl)

            holder.binding.ivAttachmentImage.load(fixedImageUrl){
                crossfade(true)
                listener(
                    onStart =
{holder.binding.pbImageLoading.visibility = View.VISIBLE},
                    onSuccess = {_,_ ->
holder.binding.pbImageLoading.visibility = View.GONE},
                    onError = {_,_ ->
holder.binding.pbImageLoading.visibility = View.GONE}
                )
            }

            holder.binding.flAttachmentImage.setOnClickListener { view ->
                val fixedUrl =
getActiveUrl(message.attachmentUrl)
                val intent = Intent(Intent.ACTION_VIEW,
Uri.parse(fixedUrl))
                view.context.startActivity(intent)
            }

        } else {
            holder.binding.flAttachmentImage.visibility =
View.GONE
            holder.binding.llAttachmentDoc.visibility =
View.VISIBLE
            holder.binding.tvDocName.text =
message.attachmentName ?: "Файл"

            trackDownloadProgress(holder.itemView.context,
message, holder)
        }
    } else {

```

```

        holder.binding.flAttachmentImage.visibility =
View.GONE
        holder.binding.llAttachmentDoc.visibility =
View.GONE
    }

    if (message.content.isBlank()) {
        holder.binding.tvMessageText.visibility = View.GONE
    } else {
        holder.binding.tvMessageText.visibility =
View.VISIBLE
    }
    val menuOptions = mutableListOf<String>()

    if (!message.attachmentUrl.isNullOrBlank()) {
        menuOptions.add("Сохранить")
    }

    if (message.senderId == currentUserId) {
        if (message.attachmentUrl.isNullOrBlank()) {
            menuOptions.add("Редактировать")
        }
        menuOptions.add("Удалить")
    } else if (isGroup && adminId == currentUserId) {
        menuOptions.add("Удалить")
    }

    if (menuOptions.isEmpty()) {

holder.binding.cardMessage.setOnLongClickListener(null)

holder.binding.flAttachmentImage.setOnLongClickListener(null)
    } else {
        val longClickListener = View.OnLongClickListener {
view ->
            val context = view.context

            val isMyMessage = message.senderId ==
currentUserId
            val baseColor = if (isMyMessage) {
                myColor
            } else {
                otherColor
            }

holder.binding.cardMessage.setCardBackgroundColor(android.conten
t.res.ColorStateList.valueOf(highlightColor))

            val popup =
androidx.appcompat.widget.PopupMenu(context, view,
Gravity.BOTTOM or Gravity.END)

```

```

        menuOptions.forEach { option ->
            popup.menu.add(option)
        }
        popup.setOnDismissListener {

holder.binding.cardMessage.setCardBackgroundColor(android.conten
t.res.ColorStateList.valueOf(baseColor))
        }

        popup.setOnMenuItemClickListener { menuItem ->
            when (menuItem.title) {
                "Сохранить" -> {
                    val fixedUrl =
getActiveUrl(message.attachmentUrl)
                    startDownload(context, fixedUrl,
message.attachmentName ?: "file") {}
                    true
                }
                "Редактировать" -> {
                    val input =
com.google.android.material.textfield.TextInputEditText(context)
                    .apply {
                        setText(message.content)

setSelection(message.content.length)
                    }
                    val container =
android.widget.FrameLayout(context)
                    val params =
android.widget.FrameLayout.LayoutParams(
ViewGroup.LayoutParams.MATCH_PARENT,
ViewGroup.LayoutParams.WRAP_CONTENT
                    )
                    params.setMargins(64, 24, 64, 0)
                    container.addView(input, params)

                    MaterialAlertDialogBuilder(context)
                        .setTitle("Редактировать
сообщение")
                        .setView(container)
                        .setPositiveButton("Сохранить")
{ _, _ ->
                    val newText =
input.text.toString().trim()
                    if (newText.isNotEmpty() &&
newText != message.content) {
                        onEditClicked(message.id, newText)
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    .setNegativeButton("Отмена",
null)
        .show()
        true
    }
    "Удалить" -> {
        MaterialAlertDialogBuilder(context)
            .setTitle("Удалить сообщение?")
            .setMessage("Вы уверены, что
хотите удалить это сообщение?")
            .setPositiveButton("Удалить") {
_, _ ->
                onDeleteClicked(message.id)
            }
            .setNegativeButton("Отмена",
null)
                .show()
                true
            }
        else -> false
    }
}

if (android.os.Build.VERSION.SDK_INT >=
android.os.Build.VERSION_CODES.Q) {
    popup.setForceShowIcon(true)
}

popup.show()
true
}

holder.binding.cardMessage.setOnLongClickListener(longClickListe
ner)

holder.binding.flAttachmentImage.setOnLongClickListener(longClic
kListener)
}

private fun startDownload(context: Context, url: String,
fileName: String, onConfirm: () -> Unit) {
    if (url.isBlank()) return
    val fixedUrl = getActiveUrl(url)
    MaterialAlertDialogBuilder(context)
        .setTitle("Загрузка")
        .setMessage("Сохранить «$fileName» на устройство?")
        .setPositiveButton("Да") {_, _ ->

```

```

        onConfirm()
        val request =
DownloadManager.Request(Uri.parse(fixedUrl))
            .setTitle(fileName)
            .setDescription("Загрузка файла...")

        .setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE_NOTIFY_COMPLETED)

        .setDestinationInExternalPublicDir(Environment.DIRECTORY_DOWNLOADS, fileName)

        val downloadManager =
context.getSystemService(Context.DOWNLOAD_SERVICE) as
DownloadManager
        downloadManager.enqueue(request)
        Toast.makeText(context, "Загрузка началась",
Toast.LENGTH_SHORT).show()
    }
    .setNegativeButton("Отмена", null)
    .show()
}

override fun getItemCount() = messages.size

override fun onViewRecycled(holder: MessageViewHolder) {
    super.onViewRecycled(holder)
    val position = holder.adapterPosition
    if (position != RecyclerView.NO_POSITION && position <
messages.size) {
        val messageId = messages[position].id
        downloadTrackingJobs[messageId]?.cancel()
    }
}

fun updateData(newMessages: List<MessageDto>) {
    val diffCallback = object : DiffUtil.Callback() {
        override fun getOldListSize() = messages.size
        override fun getNewListSize() = newMessages.size

        override fun areItemsTheSame(oldPos: Int, newPos:
Int): Boolean {
            return messages[oldPos].id ==
newMessages[newPos].id
        }

        override fun areContentsTheSame(oldPos: Int, newPos:
Int): Boolean {
            return messages[oldPos] == newMessages[newPos]
        }
    }
}

```

```

        val diffResult = DiffUtil.calculateDiff(diffCallback)
        messages = newMessages
        diffResult.dispatchUpdatesTo(this)
    }
    fun clearTargetMessage() {
        if (targetMessageId != -1L) {
            val oldId = targetMessageId
            targetMessageId = -1L
            val pos = messages.indexOfFirst { it.id == oldId }
            if (pos != -1) notifyItemChanged(pos)
        }
    }

    fun setTarget(id: Long) {
        val oldId = targetMessageId
        targetMessageId = id
        val oldPos = messages.indexOfFirst { it.id == oldId }
        val newPos = messages.indexOfFirst { it.id == id }

        if (oldPos != -1) notifyItemChanged(oldPos)
        if (newPos != -1) notifyItemChanged(newPos)
    }

    private fun getActiveUrl(originalUrl: String?): String {
        if (originalUrl.isNullOrEmpty()) return ""
        val activeHost =
            "${com.example.tiltedgram.data.NetworkModule.currentHost}:${com.
            example.tiltedgram.data.NetworkModule.currentPort}"
        return originalUrl
            .replace("192.168.1.63:8080", activeHost)
            //.replace("193.68.89.59:8081", activeHost)
            .replace("2.26.93.132:8081", activeHost)
    }
}

```

Файл «AuthApi.kt»

```

package com.example.tiltedgram.data
import com.example.tiltedgram.data.RegisterRequest
import okhttp3.MultipartBody
import retrofit2.Response
import retrofit2.http.Body
import retrofit2.http.DELETE
import retrofit2.http.POST
import retrofit2.http.GET
import retrofit2.http.HTTP
import retrofit2.http.Header
import retrofit2.http.Multipart
import retrofit2.http.Part
import retrofit2.http.Path
import retrofit2.http.PUT
import retrofit2.http.Query

```

```

//to server login
data class LoginRequest(
    val username: String,
    val password: String
)
//from server login
data class LoginResponse(
    val message: String,
    val token: String,
    val userId: Long,
    val username: String,
    val fullName: String,
    val email: String,
    val faculty: String,
    val studyGroup: String,
    val avatarUrl: String?
)
//user's chats
data class ChatUserResponse(
    val id: Long,
    val username: String,
    val fullName: String,
    val faculty: String,
    val group: String,
    val avatarUrl: String?
)
//request 2send
data class SendMessageRequest(
    val senderId: Long,
    val receiverId: Long?,
    val groupId: Long? = null,
    val content: String,
    val attachmentUrl: String? = null,
    val attachmentName: String? = null,
    val attachmentSize: Long? = null,
    val attachmentType: String? = null
)
//message
data class MessageDto(
    val id: Long,
    val senderId: Long,
    val receiverId: Long?,
    val groupId: Long? = null,
    val content: String,
    val timestamp: String,
    val senderName: String,
    val senderAvatarUrl: String?,
    val attachmentUrl: String? = null,
    val attachmentName: String? = null,
    val attachmentSize: Long? = null,
    val attachmentType: String? = null
)

```

```

)
//pfp
data class AvatarResponse(
    val message: String,
    val avatarUrl: String
)
//profile settings
data class UpdateProfileInfoRequest(
    val fullName: String,
    val username: String,
    val email: String
)
data class UpdateProfileInfoResponse(
    val message: String,
    val fullName: String,
    val username: String,
    val email: String
)
data class CreateGroupRequest(
    val name: String,
    val memberIds: List<Long>,
    val creatorId: Long
)
data class GroupResponse(
    val id: Long,
    val name: String,
    val avatarUrl: String?,
    val memberCount: Int,
    val adminId: Long
)
data class KickMemberRequest(
    val adminId: Long,
    val userIdToKick: Long
)
data class UpdateGroupRequest(
    val adminId: Long,
    val newName: String?
)
data class DialogResponse(
    val id: Long,
    val name: String,
    val username: String?,
    val faculty: String?,
    val group: String?,
    val email: String?,
    val avatarUrl: String?,
    val isGroup: Boolean,
    val lastMessage: String,
    val timestamp: String,
    val adminId: Long?,
    val messageId: Long? = null

```

```

)
data class AttachmentResponse(
    val url: String,
    val name: String,
    val size: Long,
    val type: String
)
data class EditMessageRequest(
    val userId: Long,
    val newContent: String
)
interface AuthApi {
    @POST("/api/auth/register")
    suspend fun register(@Body request: RegisterRequest):
Response<Any>
    @POST("/api/auth/login")
    suspend fun login(@Body request: LoginRequest):
Response<LoginResponse>
    @GET("/api/auth/users/{myId}/chats")
    suspend fun getActiveChats(@Path("myId") userId: Long):
Response<List<ChatUserResponse>>
    @POST("/api/auth/message/send")
    suspend fun sendMessage(@Body request: SendMessageRequest):
Response<MessageDto>
    @GET("api/auth/users")
    suspend fun getAllUsers(): Response<List<ChatUserResponse>>
    @GET("/api/auth/message/{myId}/{otherId}")
    suspend fun getChatHistory(
        @Path("myId") myId: Long,
        @Path("otherId") otherId: Long,
        @Query("size") size: Int
    ): Response<List<MessageDto>>
    @Multipart
    @POST("/api/auth/users/{userId}/avatar/upload")
    suspend fun uploadAvatarFile(
        @Path("userId") userId: Long,
        @Part file: MultipartBody.Part
    ): Response<AvatarResponse>
    @PUT("/api/auth/users/{userId}/profile")
    suspend fun updateProfileInfo(
        @Path("userId") userId: Long,
        @Body request: UpdateProfileInfoRequest
    ): Response<UpdateProfileInfoResponse>
    @POST("/api/auth/groups")
    suspend fun createGroup(@Body request: CreateGroupRequest):
Response<GroupResponse>
    @GET("/api/auth/users/{userId}/groups")
    suspend fun getUserGroups(@Path("userId") userId: Long):
Response<List<GroupResponse>>
    @GET("/api/auth/groups/{groupId}/messages")
    suspend fun getGroupMessages(

```

```

        @Path("groupId") groupId: Long,
        @Query("userId") userId: Long,
        @Query("size") size: Int
    ): Response<List<MessageDto>>
    @GET("/api/auth/groups/{groupId}/members")
    suspend fun getGroupMembers(@Path("groupId") groupId: Long):
Response<List<ChatUserResponse>>
        @DELETE("/api/auth/groups/{groupId}/leave/{userId}")
        suspend fun leaveGroup(@Path("groupId") groupId: Long,
@Path("userId") userId: Long): Response<Any>
        @PUT("/api/auth/groups/{groupId}")
        suspend fun updateGroup(@Path("groupId") groupId: Long,
@Body request: UpdateGroupRequest): Response<Any>
        @HTTP(method = "DELETE", path =
"/api/auth/groups/{groupId}/kick", hasBody = true)
        suspend fun kickMember(@Path("groupId") groupId: Long, @Body
request: KickMemberRequest): Response<Any>
        @GET("/api/auth/users/{userId}/info")
        suspend fun getUserInfo(@Path("userId") userId: Long):
Response<ChatUserResponse>
        @GET("/api/auth/groups/{groupId}/info")
        suspend fun getGroupInfo(@Path("groupId") groupId: Long):
Response<GroupResponse>
        @Multipart
        @POST("/api/auth/groups/{groupId}/avatar/upload")
        suspend fun uploadGroupAvatar(
            @Path("groupId") groupId: Long,
            @Query("adminId") adminId: Long,
            @Part file: MultipartBody.Part
        ): Response<Any>
        @GET("/api/auth/users/{userId}/dialogs")
        suspend fun getDialogs(@Path("userId") userId: Long) :
Response<List<DialogResponse>>
        @Multipart
        @POST("/api/auth/message/upload_attachment")
        suspend fun uploadAttachment(
            @Part file: MultipartBody.Part
        ): Response<AttachmentResponse>
        @PUT("/api/auth/message/{messageId}")
        suspend fun editMessage(
            @Path("messageId") messageId: Long,
            @Body request: EditMessageRequest
        ): Response<Any>
        @DELETE("/api/auth/message/{messageId}")
        suspend fun deleteMessage(
            @Path("messageId") messageId: Long,
            @Query("userId") userId: Long
        ): Response<Any>
        @GET("/api/auth/users/{userId}/search")
        suspend fun searchGlobalDialogs(
            @Path("userId") userId: Long,

```

```

        @Query("query") query: String
    ): Response<List<DialogResponse>>
    @POST("/api/auth/users/{userId}/fcm-token")
    suspend fun updateFcmToken(
        @Header("Authorization") token: String,
        @Path("userId") userId: Long,
        @Body tokenMap: Map<String, String>
    ): Response<Any>
    @GET("/api/auth/groups/auto-id")
    suspend fun getAutoGroupId(
        @Query("name") name: String,
        @Query("userId") userId: Long
    ): Response<Map<String, Long>>
}

```

Файл «NotificationActionReceiver.kt»

```

package com.example.tiltedgram.services

import android.app.NotificationManager
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import androidx.core.app.RemoteInput
import com.example.tiltedgram.data.NetworkModule
import com.example.tiltedgram.data.SendMessageRequest
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import androidx.core.app.NotificationCompat
import com.example.tiltedgram.R

class NotificationActionReceiver : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        val manager =
            context.getSystemService(Context.NOTIFICATION_SERVICE) as
            NotificationManager
        val notificationId =
            intent.getIntExtra("NOTIFICATION_ID", -1)

        if (intent.action ==
            "com.example.tiltedgram.ACTION_MARK_READ") {
            if (notificationId != -1) {
                manager.cancel(notificationId)
            }
            return
        }

        if (intent.action ==
            "com.example.tiltedgram.ACTION_REPLY") {
            val replyText =
                RemoteInput.getResultsFromIntent(intent)

```

```

        ?.getCharSequence("KEY_TEXT_REPLY")?.toString()
        val chatId = intent.getLongExtra("RECEIVER_ID", -1L)
        val isGroup = intent.getBooleanExtra("IS_GROUP",
false)

        if (!replyText.isNullOrEmpty() && chatId != -1L) {
            val sharedPrefs =
context.getSharedPreferences("MessengerPrefs",
Context.MODE_PRIVATE)
            val currentUserId =
sharedPrefs.getLong("USER_ID", -1L)

            if (currentUserId != -1L) {
                val pendingResult = goAsync()

                CoroutineScope(Dispatchers.IO).launch {
                    try {
                        val rId = if (isGroup) null else
chatId
                        val gId = if (isGroup) chatId else
null

                        NetworkModule.authApi.sendMessage(
                            SendMessageRequest(
                                senderId = currentUserId,
                                receiverId = rId,
                                groupId = gId,
                                content = replyText
                            )
                        )

                        val repliedNotification =
NotificationCompat.Builder(context, "chat_messages_channel")
                            .setSmallIcon(R.drawable.logo)
                            .setContentText("Отправлено")
                            .build()
                        manager.notify(notificationId,
repliedNotification)

                        manager.cancel(notificationId)

                    } catch (e: Exception) {
                        e.printStackTrace()
                    } finally {
                        pendingResult.finish()
                    }
                }
            }
        }
    }
}

```

```

    }
}

```

Файл «GroupWizardActivity.kt»

```

package com.example.tiltedgram.group

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.example.tiltedgram.R
import
com.example.tiltedgram.databinding.ActivityGroupWizardBinding
import com.google.android.material.transition.MaterialSharedAxis

class GroupWizardActivity : AppCompatActivity() {

    lateinit var binding: ActivityGroupWizardBinding

    override fun onCreate(savedInstanceState: Bundle?) {

        window.requestFeature(android.view.Window.FEATURE_ACTIVITY_TRANS
ITIONS)
        window.enterTransition =
com.google.android.material.transition.platform.MaterialSharedAx
is(

com.google.android.material.transition.platform.MaterialSharedAx
is.Z, true
        )
        window.returnTransition =
com.google.android.material.transition.platform.MaterialSharedAx
is(

com.google.android.material.transition.platform.MaterialSharedAx
is.Z, false
        )

        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding =
ActivityGroupWizardBinding.inflate(layoutInflater)
        setContentView(binding.root)

        ViewCompat.setOnApplyWindowInsetsListener(binding.appBarLayout)
        { v, insets ->
            val statusBarHeight =
insets.getInsets(WindowInsetsCompat.Type.statusBars()).top
            v.setPadding(0, statusBarHeight, 0, 0)
        }
    }
}

```

					ПГУ1.090301.37.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		152

```

        insets
    }

    binding.btnBack.setOnClickListener {
        if (supportFragmentManager.backStackEntryCount > 0)
        {
            supportFragmentManager.popBackStack()
            updateToolbar("Создать группу", "Выберите
участников")
        } else {
            finishAfterTransition()
        }
    }

    if (savedInstanceState == null) {
        supportFragmentManager.beginTransaction()
            .replace(R.id.fragmentContainer,
SelectMembersFragment())
            .commit()
    }
}

fun updateToolbar(title: String, subtitle: String = "") {
    binding.tvWizardTitle.text = title
    binding.tvWizardSubtitle.text = subtitle
    binding.tvWizardSubtitle.visibility = if
(subtitle.isEmpty()) android.view.View.GONE else
android.view.View.VISIBLE
}

fun navigateToNextStep(selectedIds: LongArray) {
    val nextFragment = CreateGroupInfoFragment().apply {
        arguments = Bundle().apply {
putLongArray("SELECTED_MEMBER_IDS", selectedIds) }
        enterTransition =
MaterialSharedAxis(MaterialSharedAxis.X, true)
        returnTransition =
MaterialSharedAxis(MaterialSharedAxis.X, false)
    }

    val currentFragment =
supportFragmentManager.findFragmentById(R.id.fragmentContainer)
    currentFragment?.exitTransition =
MaterialSharedAxis(MaterialSharedAxis.X, true)
    currentFragment?.reenterTransition =
MaterialSharedAxis(MaterialSharedAxis.X, false)

    supportFragmentManager.beginTransaction()
        .replace(R.id.fragmentContainer, nextFragment)
        .addToBackStack(null)
        .commit()
}

```

```

        updateToolbar("Новая группа", "")
    }
}

```

Файл «activity_chat.xml»

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="?android:attr/colorBackground"
    android:animateLayoutChanges="true">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/chatAppBarLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?attr/colorOnPrimaryFixedVariant"
        app:elevation="4dp">

        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize">

            <LinearLayout
                android:id="@+id/llNormalToolbar"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="horizontal"
                android:gravity="center_vertical"
                android:paddingHorizontal="4dp">

                <ImageButton
                    android:id="@+id/btnBack"
                    android:layout_width="48dp"
                    android:layout_height="48dp"
                    android:src="@drawable/ic_arrow_back"

                    android:background="?attr/selectableItemBackgroundBorderless"
                    app:tint="@android:color/white" />

            <com.google.android.material.imageview.ShapeableImageView
                android:id="@+id/ivChatAvatar"
                android:layout_width="42dp"
                android:layout_height="42dp"
                android:layout_marginStart="4dp"
                android:scaleType="centerCrop"

```

```

app:shapeAppearanceOverlay="@style/CircleImageView" />

<LinearLayout
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:orientation="vertical"
    android:layout_marginStart="12dp">

    <TextView
        android:id="@+id/tvChatName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@android:color/white"
        android:textSize="18sp"
        android:textStyle="bold"
        android:maxLines="1"
        android:ellipsize="end"/>

    <TextView
        android:id="@+id/tvChatStatus"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#B3FFFFFF"
        android:textSize="13sp"
        android:maxLines="1"/>
</LinearLayout>

<ImageButton
    android:id="@+id/btnSearchChat"
    android:layout_width="48dp"
    android:layout_height="48dp"
    android:src="@drawable/ic_search"

    android:background="?attr/selectableItemBackgroundBorderless"
    app:tint="@android:color/white" />
</LinearLayout>

<LinearLayout
    android:id="@+id/llSearchToolbar"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:gravity="center_vertical"
    android:paddingHorizontal="4dp"
    android:visibility="gone">

    <ImageButton
        android:id="@+id/btnCloseSearch"
        android:layout_width="48dp"

```

```

        android:layout_height="48dp"
        android:src="@drawable/ic_arrow_back"

android:background="?attr/selectableItemBackgroundBorderless"
        app:tint="@android:color/white" />

<EditText
    android:id="@+id/etChatSearch"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:background="@null"
    android:hint="Поиск..."
    android:textColorHint="#B3FFFFFF"
    android:textColor="@android:color/white"
    android:inputType="text"
    android:maxLines="1"
    android:imeOptions="actionSearch"/>

<TextView
    android:id="@+id/tvSearchCount"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@android:color/white"
    android:textSize="14sp"
    android:layout_marginEnd="8dp"/>

<ImageButton
    android:id="@+id/btnSearchUp"
    android:layout_width="40dp"
    android:layout_height="40dp"
    android:src="@drawable/ic_arrow_up"

android:background="?attr/selectableItemBackgroundBorderless"
        app:tint="@android:color/white" />

<ImageButton
    android:id="@+id/btnSearchDown"
    android:layout_width="40dp"
    android:layout_height="40dp"
    android:src="@drawable/ic_arrow_down"

android:background="?attr/selectableItemBackgroundBorderless"
        app:tint="@android:color/white" />
</LinearLayout>

</FrameLayout>
</com.google.android.material.appbar.AppBarLayout>

<FrameLayout
    android:layout_width="match_parent"

```

```

        android:layout_height="0dp"
        android:layout_weight="1">

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/rvMessages"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:clipToPadding="false"
    android:paddingVertical="8dp" />

<com.google.android.material.card.MaterialCardView
    android:id="@+id/cvScrollDownIndicator"
    android:layout_width="wrap_content"
    android:layout_height="42dp"
    android:layout_gravity="bottom|center_horizontal"
    android:layout_marginBottom="16dp"
    android:visibility="gone"
    app:cardCornerRadius="21dp"
    app:cardElevation="6dp"
    app:cardBackgroundColor="?attr/colorPrimary">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        android:gravity="center"
        android:paddingHorizontal="16dp">

        <ImageView
            android:layout_width="18dp"
            android:layout_height="18dp"
            android:src="@drawable/ic_arrow_down"
            app:tint="@android:color/white" />

        <TextView
            android:id="@+id/tvNewMessagesCount"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:textColor="@android:color/white"
            android:textSize="14sp"
            android:textStyle="bold"
            android:text="1" />
    </LinearLayout>
</com.google.android.material.card.MaterialCardView>
</FrameLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?android:attr/colorBackground"

```

```

        android:paddingHorizontal="8dp"
        android:paddingVertical="8dp">

        <com.google.android.material.card.MaterialCardView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:minHeight="50dp"
            app:cardCornerRadius="25dp"
            app:cardElevation="0dp"
            app:cardBackgroundColor="?attr/colorSurfaceVariant">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="horizontal"
                android:gravity="bottom"
                android:paddingHorizontal="4dp"
                android:paddingVertical="4dp">

                <ImageButton
                    android:id="@+id/btnAttach"
                    android:layout_width="42dp"
                    android:layout_height="42dp"
                    android:src="@drawable/ic_attach_file"

                    android:background="?attr/selectableItemBackgroundBorderless"
                    app:tint="?attr/colorOutline" />

                <EditText
                    android:id="@+id/etMessage"
                    android:layout_width="0dp"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:background="@null"
                    android:hint="Message"
                    android:textColorHint="?attr/colorOutline"
                    android:textColor="?attr/colorOnSurface"
                    android:textSize="16sp"
                    android:maxLines="5"
                    android:layout_gravity="center_vertical"
                    android:paddingHorizontal="8dp"
                    android:paddingVertical="10dp"

                    android:inputType="textMultiLine|textCapSentences" />

                <ImageButton
                    android:id="@+id/btnSend"
                    android:layout_width="42dp"
                    android:layout_height="42dp"
                    android:src="@drawable/ic_send"

```

```

        android:background="?attr/selectableItemBackgroundBorderless"
        app:tint="?attr/colorPrimary" />

```

```

    </LinearLayout>
</com.google.android.material.card.MaterialCardView>

```

```

</LinearLayout>
</LinearLayout>

```

Файл «activity_login.xml»

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:fitsSystemWindows="true"
    android:animateLayoutChanges="true">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="16dp"
        android:gravity="center">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Tiltedgram"
            android:textSize="32sp"
            android:fontFamily="monospace"
            android:textStyle="bold"
            android:layout_marginBottom="50dp"/>

        <EditText
            android:id="@+id/etUsername"
            android:layout_width="match_parent"
            android:layout_height="50dp"
            android:hint="Никнейм"
            android:inputType="text"
            android:layout_marginBottom="8dp"/>

        <EditText
            android:id="@+id/etPassword"
            android:layout_width="match_parent"
            android:layout_height="50dp"
            android:hint="Пароль"
            android:inputType="textPassword"
            android:layout_marginBottom="16dp"/>

```

```

<TextView
    android:id="@+id/tvError"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@android:color/holo_red_dark"
    android:visibility="gone"
    android:layout_marginBottom="8dp" />

<ProgressBar
    android:id="@+id/progressBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:visibility="gone"/>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button
        android:id="@+id/btnGoToRegister"
        android:layout_width="0dp"
        android:layout_height="65dp"
        android:layout_weight="1"

        android:text="Регистрация"
        style="?attr/materialButtonOutlinedStyle"
        android:layout_marginEnd="8dp" />

    <Button
        android:id="@+id/btnLogin"
        android:layout_width="0dp"
        android:layout_height="65dp"
        android:layout_weight="1"
        android:text="Войти"
        android:layout_marginStart="8dp"
        />

</LinearLayout>
</LinearLayout>
</LinearLayout>
Файл «activity_main.xml»

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="?android:attr/colorBackground">

```

```

<androidx.fragment.app.FragmentContainerView
    android:id="@+id/nav_host_fragment"

android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_above="@id/bottomNav"
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph" />

<include
    android:id="@+id/bottomNav"
    layout="@layout/bottom_navigation"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    />

</RelativeLayout>

```

Файл «activity_registration.xml»

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:fitsSystemWindows="true"
    android:animateLayoutChanges="true">

    <ImageButton
        android:id="@+id/btnBack"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:layout_gravity="start"

        android:background="?attr/selectableItemBackgroundBorderless"
        android:src="?attr/homeAsUpIndicator"
        android:scaleType="fitCenter"
        android:padding="10dp"
        android:contentDescription="Назад"
        android:layout_marginTop="15dp"
        android:layout_marginLeft="15dp"/>

    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="16dp"

```

```

        android:gravity="center">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Создать аккаунт"
            android:textSize="24sp"
            android:textStyle="bold"
            android:layout_marginBottom="32dp"/>

        <EditText
            android:id="@+id/etFullName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Имя и Фамилия"
            android:inputType="textPersonName"
            android:layout_marginBottom="8dp"/>

        <EditText
            android:id="@+id/etUsername"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Никнейм"
            android:inputType="text"
            android:layout_marginBottom="8dp"/>

        <EditText
            android:id="@+id/etEmail"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Email"
            android:inputType="textEmailAddress"
            android:layout_marginBottom="8dp"/>

        <EditText
            android:id="@+id/etPassword"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Пароль"
            android:inputType="textPassword"
            android:layout_marginBottom="8dp"/>

        <com.google.android.material.textfield.TextInputLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Выберите факультет"

            style="@style/Widget.MaterialComponents.TextInputLayout.Outlined
            Box.ExposedDropdownMenu"
            android:layout_marginBottom="8dp">

```

```

        <AutoCompleteTextView
            android:id="@+id/actvFaculty"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="none"/>
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Выберите группу"

style="@style/Widget.MaterialComponents.TextInputLayout.Outlined
Box.ExposedDropdownMenu"
        android:layout_marginBottom="16dp">

        <AutoCompleteTextView
            android:id="@+id/actvGroup"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="none"/>
    </com.google.android.material.textfield.TextInputLayout>

    <TextView
        android:id="@+id/tvError"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@android:color/holo_red_dark"
        android:visibility="gone"
        android:layout_marginBottom="8dp"/>

    <ProgressBar
        android:id="@+id/progressBar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"/>

    <Button
        android:id="@+id/btnRegister"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Зарегистрироваться"/>
    </LinearLayout>
</LinearLayout>

```

Файл «item_chat.xml»

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"

```

Изм.	Лист	№ докум.	Подпись	Дата

```

xmlns:app="http://schemas.android.com/apk/res-auto"
android:orientation="horizontal"
android:padding="16dp"
android:gravity="center_vertical"
android:background="?attr/selectableItemBackground"
android:animateLayoutChanges="true">

<com.google.android.material.imageview.ShapeableImageView
    android:id="@+id/ivAvatar"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:scaleType="centerCrop"
    android:src="@drawable/ic_launcher_background"
    app:shapeAppearanceOverlay="@style/CircleImageView" />

<LinearLayout
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:orientation="vertical"
    android:layout_marginStart="16dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="center_vertical">

        <TextView
            android:id="@+id/tvChatName"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Имя контакта"
            android:textSize="18sp"
            android:textStyle="bold"
            android:maxLines="1"
            android:ellipsize="end"
            android:textColor="?attr/colorOnSurface"/>

        <TextView
            android:id="@+id/tvChatTime"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="00:00"
            android:textSize="12sp"
            android:layout_marginStart="8dp"
            android:textColor="?attr/colorOutline"
            android:visibility="gone"/>
    </LinearLayout>

```

```

        <TextView
            android:id="@+id/tvLastMessage"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Последнее сообщение..."
            android:layout_marginTop="2dp"
            android:maxLines="1"
            android:ellipsize="end"
            android:textColor="?attr/colorOnSurfaceVariant"/>
    </LinearLayout>
</LinearLayout>

```

Файл «AuthController.kt»

```

package com.tiledgram.messenger_server

import org.springframework.http.ResponseEntity
import
org.springframework.security.crypto.password.PasswordEncoder
import org.springframework.web.bind.annotation.*
import org.springframework.web.multipart.MultipartFile
import java.io.File
import org.springframework.data.domain.PageRequest

@RestController
@RequestMapping("/api/auth")
class AuthController(
    private val userRepository: UserRepository,
    private val messageRepository: MessageRepository,
    private val groupRepository: GroupRepository,
    private val passwordEncoder: PasswordEncoder,
    private val jwtUtil: JwtUtil,
    private val fcmService: FcmService
) {

    data class RegisterRequest(
        val username: String,
        val fullName: String,
        val email: String,
        val password: String,
        val faculty: String,
        val group: String
    )

    @PostMapping("/register")
    fun register(@RequestBody request: RegisterRequest):
    ResponseEntity<Any> {

        val nameRegex = Regex("[a-zA-Za-яA-ЯёЁ\\s\\-]{2,50}$")
        if (!nameRegex.matches(request.fullName)) {
            return ResponseEntity.badRequest()
                .body(mapOf("error" to "ФИО может содержать

```

```

        только буквы, пробелы и дефисы (от 2 до 50 символов)))
    }

    val usernameRegex = Regex("^[a-zA-Z0-9_]{3,20}$")
    if (!usernameRegex.matches(request.username)) {
        return ResponseEntity.badRequest()
            .body(mapOf("error" to "Никнейм должен быть от 3
до 20 символов (только латиница, цифры и '_'"))
    }

    val emailRegex = Regex("^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-
]+\\.[a-z]+$")
    if (!emailRegex.matches(request.email)) {
        return ResponseEntity.badRequest()
            .body(mapOf("error" to "Некорректный формат
email"))
    }

    if
(userRepository.existsByUsernameOrEmail(request.username,
request.email)) {
        return ResponseEntity.badRequest()
            .body(mapOf("error" to "Пользователь с таким
email или именем уже существует"))
    }

    val newUser = User(
        username = request.username,
        fullName = request.fullName,
        email = request.email,
        passwordHash =
passwordEncoder.encode(request.password) ?: "",
        faculty = request.faculty,
        studyGroup = request.group
    )

    userRepository.save(newUser)
    handleAutoGroupJoin(newUser, request.group)
    return ResponseEntity.ok(mapOf("message" to "Регистрация
успешна!"))
}

data class LoginRequest(
    val username: String,
    val password: String
)

@PostMapping("/login")
fun login(@RequestBody request: LoginRequest):
ResponseEntity<Any> {
    val user =

```

```

userRepository.findByUsername(request.username)
    if (user == null) {
        return
        ResponseEntity.badRequest().body(mapOf("error" to "Неверное имя
пользователя или пароль"))
    }
    if (!passwordEncoder.matches(request.password,
user.passwordHash ?: "")) {
        return
        ResponseEntity.badRequest().body(mapOf("error" to "Неверное имя
пользователя или пароль"))
    }
    val token = jwtUtil.generateToken(user.username,
user.id)
    return ResponseEntity.ok(
        mapOf(
            "message" to "Вход успешно выполнен!",
            "token" to token,
            "userId" to user.id,
            "username" to user.username,
            "fullName" to user.fullName,
            "email" to user.email,
            "faculty" to user.faculty,
            "studyGroup" to user.studyGroup,
            "avatarUrl" to user.avatarUrl
        )
    )
}

//answer class
data class UserResponse(
    val id: Long,
    val username: String,
    val fullName: String,
    val faculty: String,
    val group: String,
    val avatarUrl: String?
)

//all users
@GetMapping("/users")
fun getAllUsers(): ResponseEntity<List<UserResponse>> {
    val allUsers = userRepository.findAll().map { user ->
        UserResponse(
            id = user.id,
            username = user.username,
            fullName = user.fullName,
            faculty = user.faculty,
            group = user.studyGroup,
            avatarUrl = user.avatarUrl
        )
    }
}

```

```

    }
    return ResponseEntity.ok(allUsers)
}

@GetMapping("/users/{myId}/chats")
fun getActiveChats(@PathVariable myId: Long):
ResponseEntity<List<UserResponse>> {
    val activeUsers =
userRepository.findActiveChatUsers(myId).map { user ->
        UserResponse(user.id, user.username, user.fullName,
user.faculty, user.studyGroup, user.avatarUrl)
    }
    return ResponseEntity.ok(activeUsers)
}

//data transfer object
data class SendMessageRequest(
    val senderId: Long,
    val receiverId: Long?,
    val groupId: Long? = null,
    val content: String?,
    val attachmentUrl: String? = null,
    val attachmentName: String? = null,
    val attachmentSize: Long? = null,
    val attachmentType: String? = null
)

@PostMapping("/message/send")
fun sendMessage(@RequestBody request: SendMessageRequest):
ResponseEntity<Message> {
    //message object
    val newMessage = Message(
        senderId = request.senderId,
        receiverId = request.receiverId,
        groupId = request.groupId,
        content = request.content ?: "",
        attachmentUrl = request.attachmentUrl,
        attachmentName = request.attachmentName,
        attachmentSize = request.attachmentSize,
        attachmentType = request.attachmentType
    )
    //save 2SQL by repository
    val savedMessage = messageRepository.save(newMessage)

    val pushBody = if (request.content.isNullOrBlank() &&
!request.attachmentType.isNullOrBlank()) {
        if (request.attachmentType == "IMAGE") "🖼️
Изображение" else "📁 Файл"
    } else {
        request.content ?: ""
    }
}

```

```

    }

    if (request.receiverId != null) {
        val sender =
userRepository.findById(request.senderId).get()
        fcmService.sendNotification(
            userId = request.receiverId,
            title = sender.fullName,
            body = pushBody,
            chatId = request.senderId,
            isGroup = false,
            avatarUrl = sender.avatarUrl,
            senderId = sender.id,
            senderAvatarUrl = sender.avatarUrl
        )
    } else if (request.groupId != null) {
        val group =
userRepository.findById(request.groupId).get()
        val sender =
userRepository.findById(request.senderId).get()

        group.members.forEach { member ->
            if (member.id != request.senderId) {
                fcmService.sendNotification(
                    userId = member.id,
                    title = group.name,
                    body = "${sender.fullName}: $pushBody",
                    chatId = group.id,
                    isGroup = true,
                    avatarUrl = group.avatarUrl,
                    senderId = sender.id,
                    senderAvatarUrl = sender.avatarUrl
                )
            }
        }
    }

    return ResponseEntity.ok(savedMessage)
}

//chat history
@GetMapping("/message/{myId}/{otherId}")
fun getMessages(
    @PathVariable myId: Long,
    @PathVariable otherId: Long,
    @RequestParam(defaultValue = "50") size: Int
): ResponseEntity<List<MessageResponse>> {
    val pageable = PageRequest.of(0, size)
    val history = messageRepository.findChatHistory(myId,
otherId, pageable).reversed()

```

```

        val senderIds = history.map { it.senderId }.distinct()
        val sendersMap =
userRepository.findAllById(senderIds).associateBy { it.id }

        val enrichedHistory = history.map { msg ->
            val sender = sendersMap[msg.senderId]
            MessageResponse(
                id = msg.id,
                senderId = msg.senderId,
                receiverId = msg.receiverId,
                groupId = msg.groupId,
                content = msg.content,
                timestamp = msg.timestamp.toString(),
                senderName = sender?.fullName ?: "Неизвестный",
                senderAvatarUrl = sender?.avatarUrl,
                attachmentUrl = msg.attachmentUrl,
                attachmentName = msg.attachmentName,
                attachmentSize = msg.attachmentSize,
                attachmentType = msg.attachmentType
            )
        }
        return ResponseEntity.ok(enrichedHistory)
    }

    @PostMapping("/users/{userId}/avatar/upload")
    fun updateAvatar(@PathVariable userId: Long,
@RequestParam("file") file: MultipartFile): ResponseEntity<Any>
    {
        val user = userRepository.findById(userId).orElse(null)
        ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Пользователь
не найден"))

        //create uploads
        val uploadDir = File("uploads").absoluteFile
        if (!uploadDir.exists()) {
            uploadDir.mkdirs()
        }

        //save file
        val fileName = "avatar_${user.id}.jpg"
        val filePath = File(uploadDir, fileName)
        file.transferTo(filePath)

        //link
        val requestHeader =
org.springframework.web.context.request.RequestContextHolder.cur
rentRequestAttributes() as
org.springframework.web.context.request.ServletRequestAttributes
        //val host = requestHeader.request.getHeader("Host") ?:
"193.68.89.59:8081"

```

```

        val host = requestHeader.request.getHeader("Host") ?:
"2.26.93.132:8081"

        val publicUrl =
"http://$host/uploads/$fileName?t=${System.currentTimeMillis()}"

        //update db
        user.avatarUrl = publicUrl
        userRepository.save(user)

        return ResponseEntity.ok(mapOf("message" to "Аватар
успешно загружен", "avatarUrl" to publicUrl))
    }

    data class UpdateProfileInfoRequest(
        val fullName: String,
        val username: String,
        val email: String
    )

    @PutMapping("/users/{userId}/profile")
    fun updateProfile(
        @PathVariable userId: Long,
        @RequestBody request: UpdateProfileInfoRequest
    ): ResponseEntity<Any> {
        val nameRegex = Regex("[a-zA-Za-яA-ЯёЁ\\s\\-]{2,50}$")
        if (!nameRegex.matches(request.fullName)) {
            return
ResponseEntity.badRequest().body(mapOf("error" to "ФИО может
содержать только буквы, пробелы и дефисы"))
        }
        val usernameRegex = Regex("[a-zA-Z0-9_]{3,20}$")
        if (!usernameRegex.matches(request.username)) {
            return ResponseEntity.badRequest()
                .body(mapOf("error" to "Никнейм должен быть от 3
до 20 символов (только латиница, цифры и '_'"))
        }
        val emailRegex = Regex("[A-Za-z0-9+_.-]+@[A-Za-z0-9.-
]+\\.([a-z]+)$")
        if (!emailRegex.matches(request.email)) {
            return ResponseEntity.badRequest()
                .body(mapOf("error" to "Некорректный формат
email"))
        }

        val user = userRepository.findById(userId).orElse(null)
        ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Пользователь
не найден"))

        if (user.username != request.username) {

```

```

        val existingUser =
userRepository.findByUsername(request.username)
        if (existingUser != null && existingUser.id !=
userId) {
            return
ResponseEntity.badRequest().body(mapOf("error" to "Этот никнейм
уже занят"))
        }
    }
    if (user.email != request.email) {
        val existingEmail =
userRepository.findByEmail(request.email)
        if (existingEmail != null && existingEmail.id !=
userId) {
            return
ResponseEntity.badRequest().body(mapOf("error" to "Эта почта уже
занята"))
        }
    }
    user.fullName = request.fullName
    user.username = request.username
    user.email = request.email
    userRepository.save(user)

    return ResponseEntity.ok(
        mapOf(
            "message" to "Профиль успешно обновлен",
            "fullName" to user.fullName,
            "username" to user.username,
            "email" to user.email
        )
    )
}

data class CreateGroupRequest(
    val name: String,
    val memberIds: List<Long>,
    val creatorId: Long
)

data class GroupResponse(
    val id: Long,
    val name: String,
    val avatarUrl: String?,
    val memberCount: Int,
    val adminId: Long
)

@PostMapping("/groups")
fun createGroup(@RequestBody request: CreateGroupRequest):
ResponseEntity<Any> {

```

```

        if (request.name.isBlank() ||
request.memberIds.isEmpty()) {
            return
ResponseEntity.badRequest().body(mapOf("error" to "Название и
участники обязательны"))
        }

        // Указываем админа при создании!
val group = GroupChat(
    name = request.name,
    adminId = request.creatorId,
    isAcademic = false
)

val members =
userRepository.findAllById(request.memberIds)
group.members.addAll(members)
val savedGroup = groupRepository.save(group)

return ResponseEntity.ok(
    GroupResponse(
        id = savedGroup.id,
        name = savedGroup.name,
        avatarUrl = savedGroup.avatarUrl,
        memberCount = savedGroup.members.size,
        adminId = savedGroup.adminId
    )
)
}

@GetMapping("/users/{userId}/groups")
fun getUserGroups(@PathVariable userId: Long):
ResponseEntity<List<GroupResponse>> {
    val groups =
groupRepository.findGroupsByUserId(userId).map { group ->
    GroupResponse(
        id = group.id,
        name = group.name,
        avatarUrl = group.avatarUrl,
        memberCount = group.members.size,
        adminId = group.adminId
    )
}
    return ResponseEntity.ok(groups)
}

@GetMapping("/groups/{groupId}/messages")
fun getGroupMessages(
    @PathVariable groupId: Long,
    @RequestParam userId: Long,
    @RequestParam(defaultValue = "50") size: Int

```

```

        ): ResponseEntity<Any> {
            val group =
groupRepository.findById(groupId).orElse(null)
            ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Группа не
найденa"))

            val isMember = group.members.any { it.id == userId }
            if (!isMember) {
                return ResponseEntity.status(403).body(mapOf("error"
to "Вы были исключены из группы"))
            }

            val pageable = PageRequest.of(0, size)
            val history = messageRepository.findByGroupId(groupId,
pageable).reversed()

            val senderIds = history.map { it.senderId }.distinct()
            val sendersMap =
userRepository.findAllById(senderIds).associateBy { it.id }

            val enrichedHistory = history.map { msg ->
                val sender = sendersMap[msg.senderId]
                MessageResponse(
                    id = msg.id,
                    senderId = msg.senderId,
                    receiverId = msg.receiverId,
                    groupId = msg.groupId,
                    content = msg.content,
                    timestamp = msg.timestamp.toString(),
                    senderName = sender?.fullName ?: "Неизвестный",
                    senderAvatarUrl = sender?.avatarUrl,
                    attachmentUrl = msg.attachmentUrl,
                    attachmentName = msg.attachmentName,
                    attachmentSize = msg.attachmentSize,
                    attachmentType = msg.attachmentType
                )
            }
            return ResponseEntity.ok(enrichedHistory)
        }

data class MessageResponse(
    val id: Long,
    val senderId: Long,
    val receiverId: Long?,
    val groupId: Long?,
    val content: String,
    val timestamp: String,
    val senderName: String,
    val senderAvatarUrl: String?,
    val attachmentUrl: String?,

```

```

        val attachmentName: String?,
        val attachmentSize: Long?,
        val attachmentType: String?
    )

    data class UpdateGroupRequest(
        val adminId: Long,
        val newName: String?
    )

    data class KickMemberRequest(
        val adminId: Long,
        val userIdToKick: Long
    )

    @DeleteMapping("/groups/{groupId}/leave/{userId}")
    fun leaveGroup(@PathVariable groupId: Long, @PathVariable
userId: Long): ResponseEntity<Any> {
        val group =
groupRepository.findById(groupId).orElse(null)
        ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Группа не
найдена"))

        val user = userRepository.findById(userId).orElse(null)
        ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Пользователь
не найден"))

        if (group.members.contains(user)) {
            group.members.remove(user)

            if (group.adminId == userId) {
                if (group.members.isNotEmpty()) {
                    group.adminId = group.members.first().id
                } else {
                    val groupMessages =
messageRepository.findAllByGroupId(group.id)

                    groupMessages.forEach { msg ->
                        if (!msg.attachmentName.isNullOrBlank())
{
                            try {
                                val file =
File("uploads/attachments", msg.attachmentName)
                                if (file.exists()) file.delete()
                            } catch (e: Exception) {
                                e.printStackTrace()
                            }
                        }
                    }
                }
            }
        }
    }

```

```

        messageRepository.deleteAll(groupMessages)

        groupRepository.delete(group)
        return ResponseEntity.ok(mapOf("message" to
"Группа и её история успешно удалены"))
    }
}
groupRepository.save(group)
}
return ResponseEntity.ok(mapOf("message" to "Вы успешно
вышли из группы"))
}

@PutMapping("/groups/{groupId}")
fun updateGroup(@PathVariable groupId: Long, @RequestBody
request: UpdateGroupRequest): ResponseEntity<Any> {
    val group =
groupRepository.findById(groupId).orElse(null)
    ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Группа не
найдена"))

    if (group.adminId != request.adminId) {
        return ResponseEntity.status(403).body(mapOf("error"
to "Только создатель может изменять группу"))
    }
    if (!request.newName.isNullOrBlank()) {
        group.name = request.newName
    }
    groupRepository.save(group)
    return ResponseEntity.ok(mapOf("message" to "Группа
успешно обновлена"))
}

>DeleteMapping("/groups/{groupId}/kick")
fun kickMember(@PathVariable groupId: Long, @RequestBody
request: KickMemberRequest): ResponseEntity<Any> {
    val group =
groupRepository.findById(groupId).orElse(null)
    ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Группа не
найдена"))

    if (group.adminId != request.adminId) {
        return ResponseEntity.status(403).body(mapOf("error"
to "Только создатель может удалять участников"))
    }
    if (request.adminId == request.userIdToKick) {
        return
ResponseEntity.badRequest().body(mapOf("error" to "Нельзя
выгнать самого себя"))
    }
}

```

```

    }
    val userToKick =
userRepository.findById(request.userIdToKick).orElse(null)
        ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Пользователь
не найден"))

    group.members.remove(userToKick)
    groupRepository.save(group)

    return ResponseEntity.ok(mapOf("message" to
"Пользователь удален"))
}

@GetMapping("/groups/{groupId}/members")
fun getGroupMembers(@PathVariable groupId: Long):
ResponseEntity<List<UserResponse>> {
    val group =
groupRepository.findById(groupId).orElse(null)
        ?: return ResponseEntity.badRequest().build()

    val membersList = group.members.map { user ->
        UserResponse(
            id = user.id,
            username = user.username,
            fullName = user.fullName,
            faculty = user.faculty,
            group = user.studyGroup,
            avatarUrl = user.avatarUrl
        )
    }
    return ResponseEntity.ok(membersList)
}

@PostMapping("/groups/{groupId}/avatar/upload")
fun updateGroupAvatar(
    @PathVariable groupId: Long,
    @RequestParam("adminId") adminId: Long,
    @RequestParam("file") file: MultipartFile
): ResponseEntity<Any> {
    val group =
groupRepository.findById(groupId).orElse(null)
        ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Группа не
найдена"))

    if (group.adminId != adminId) {
        return ResponseEntity.status(403).body(mapOf("error"
to "Только создатель может изменять группу"))
    }
    val uploadDir = File("uploads").absoluteFile

```

```

        if (!uploadDir.exists()) uploadDir.mkdirs()

        val fileName = "group_${group.id}.jpg"
        val filePath = File(uploadDir, fileName)
        file.transferTo(filePath)

        val requestHeader =
org.springframework.web.context.request.RequestContextHolder.cur
rentRequestAttributes() as
org.springframework.web.context.request.ServletRequestAttributes
        //val host = requestHeader.request.getHeader("Host") ?:
"193.68.89.59:8081"
        val host = requestHeader.request.getHeader("Host") ?:
"2.26.93.132:8081"

        val publicUrl =
"http://$host/uploads/$fileName?t=${System.currentTimeMillis()}"

        group.avatarUrl = publicUrl
        groupRepository.save(group)
        return ResponseEntity.ok(mapOf("message" to "Аватар
обновлен", "avatarUrl" to publicUrl))
    }
    @GetMapping("/users/{userId}/info")
    fun getUserInfo(@PathVariable userId: Long):
ResponseEntity<UserResponse> {
        val user = userRepository.findById(userId).orElse(null)
        ?: return ResponseEntity.badRequest().build()

        return ResponseEntity.ok(UserResponse(
            user.id, user.username, user.fullName, user.faculty,
            user.studyGroup, user.avatarUrl
        ))
    }
    @GetMapping("/groups/{groupId}/info")
    fun getGroupInfo(@PathVariable groupId: Long):
ResponseEntity<GroupResponse> {
        val group =
groupRepository.findById(groupId).orElse(null)
        ?: return ResponseEntity.badRequest().build()

        return ResponseEntity.ok(GroupResponse(
            group.id, group.name, group.avatarUrl,
            group.members.size, group.adminId
        ))
    }
    data class DialogResponse(
        val id: Long,
        val name: String,
        val username: String?,
        val faculty: String?,

```

```

        val group: String?,
        val email: String?,
        val avatarUrl: String?,
        val isGroup: Boolean,
        val lastMessage: String,
        val timestamp: String,
        val adminId: Long? = null,
        val messageId: Long? = null
    )
    @GetMapping("/users/{userId}/dialogs")
    fun getDialogs(@PathVariable userId: Long):
    ResponseEntity<List<DialogResponse>> {
        val dialogs = mutableListOf<DialogResponse>()

        //ls
        val activeUsers =
        userRepository.findActiveChatUsers(userId)
        for (user in activeUsers) {
            val lastMsg =
            messageRepository.findChatHistory(userId, user.id,
            PageRequest.of(0, 1)).firstOrNull()

            val msgContent = if (lastMsg != null) {
                if (!lastMsg.attachmentUrl.isNullOrBlank()) {
                    val label = if (lastMsg.attachmentType ==
                    "IMAGE") "Изображение" else "Файл"
                    if (lastMsg.content.isNotBlank()) "$label:
                    ${lastMsg.content}" else label
                } else lastMsg.content
            } else "Нажмите, чтобы начать переписку"

            dialogs.add(
                DialogResponse(
                    id = user.id,
                    name = user.fullName,
                    username = user.username,
                    faculty = user.faculty,
                    group = user.studyGroup,
                    email = user.email,
                    avatarUrl = user.avatarUrl,
                    isGroup = false,
                    lastMessage = msgContent,
                    timestamp = lastMsg?.timestamp?.toString()

?: "",

                    adminId = null,
                    messageId = lastMsg?.id
                )
            )
        }

        //groups

```

```

        val groups = groupRepository.findGroupsByUserId(userId)
        for (group in groups) {
            if (group.isAcademic) continue
            val lastMsg =
messageRepository.findByGroupId(group.id, PageRequest.of(0,
1)).firstOrNull()

            val lastMessageText = if (lastMsg != null) {
                //sender
                val prefix = if (lastMsg.senderId == userId) {
                    "Вы: "
                } else {
                    val sender =
userRepository.findById(lastMsg.senderId).orElse(null)
                    val shortName = sender?.fullName?.split("
")?.get(0) ?: "Кто-то"
                    "$shortName: "
                }
                //file/txt
                val msgContent = if
(!lastMsg.attachmentUrl.isNullOrBlank()) {
                    val fileLabel = if (lastMsg.attachmentType
== "IMAGE") "Изображение" else "Файл"
                    if (lastMsg.content.isNotBlank())
"$fileLabel, ${lastMsg.content}" else fileLabel
                } else {
                    lastMsg.content
                }
                "$prefix$msgContent"
            } else {
                "${group.members.size} участников"
            }

            dialogs.add(
                DialogResponse(
                    id = group.id,
                    name = group.name,
                    username = null,
                    faculty = null,
                    group = null,
                    email = null,
                    avatarUrl = group.avatarUrl,
                    isGroup = true,
                    lastMessage = lastMessageText,
                    timestamp = lastMsg?.timestamp?.toString()

?: "",

                    adminId = group.adminId,
                    messageId = lastMsg?.id
                )
            )
        }
    }

```

```

        val sortedDialogs = dialogs.sortedByDescending {
it.timestamp }
        return ResponseEntity.ok(sortedDialogs)
    }
    @GetMapping("/users/{userId}/search")
    fun searchGlobalDialogs(@PathVariable userId: Long,
@RequestParam query: String):
ResponseEntity<List<DialogResponse>> {
        if (query.isBlank()) return
ResponseEntity.ok(emptyList())

        val allMessages = messageRepository.findAll().filter {
            it.content.contains(query, ignoreCase = true)
        }

        val results = mutableListOf<DialogResponse>()

        for (msg in allMessages) {
            val gId = msg.groupId
            val rId = msg.receiverId

            if (gId != null) {
                val group =
groupRepository.findById(gId).orElse(null) ?: continue
                if (group.members.any { it.id == userId }) {

                    val prefix = if (msg.senderId == userId) {
                        "Вы: "
                    } else {
                        val sender =
userRepository.findById(msg.senderId).orElse(null)
                        val shortName =
sender?.fullName?.split(" ")?[0] ?: "Кто-то"
                        "$shortName: "
                    }

                    val msgContent = if
(!msg.attachmentUrl.isNullOrBlank()) {
                        val fileLabel = if (msg.attachmentType
== "IMAGE") "Изображение" else "Файл"
                        if (msg.content.isNotBlank())
"$fileLabel, ${msg.content}" else fileLabel
                    } else {
                        msg.content
                    }

                    results.add(
                        DialogResponse(
                            id = group.id,
                            name = group.name,

```

```

        username = null, faculty = null,
group = null, email = null,
        avatarUrl = group.avatarUrl,
        isGroup = true,
        lastMessage = "$prefix$msgContent",
        timestamp =
msg.timestamp.toString(),
        adminId = group.adminId,
        messageId = msg.id
    )
    )
    }
    } else {
        if (msg.senderId == userId || rId == userId) {
            val otherUserId = if (msg.senderId ==
userId) rId else msg.senderId
            if (otherUserId == null) continue

            val otherUser =
userRepository.findById(otherUserId).orElse(null) ?: continue
            val prefix = if (msg.senderId == userId)
"Вы: " else ""

            val msgContent = if
(!msg.attachmentUrl.isNullOrBlank()) {
                val fileLabel = if (msg.attachmentType
== "IMAGE") "Изображение" else "Файл"
                if (msg.content.isNotBlank())
"$fileLabel, ${msg.content}" else fileLabel
            } else {
                msg.content
            }

            results.add(
                DialogResponse(
                    id = otherUser.id,
                    name = otherUser.fullName,
                    username = otherUser.username,
                    faculty = otherUser.faculty,
                    group = otherUser.studyGroup,
                    email = otherUser.email,
                    avatarUrl = otherUser.avatarUrl,
                    isGroup = false,
                    lastMessage = "$prefix$msgContent",
                    timestamp =
msg.timestamp.toString(),
                    adminId = null,
                    messageId = msg.id
                )
            )
        }
    }
}

```

```

    }
}
return ResponseEntity.ok(results.sortedByDescending {
it.timestamp })
}
@PostMapping("/message/upload_attachment")
fun uploadAttachment(@RequestParam("file") file:
MultipartFile): ResponseEntity<Any> {
    try {
        //folder
        val uploadDir =
File("uploads/attachments").absoluteFile
        if (!uploadDir.exists()) uploadDir.mkdirs()

        //name
        val originalName = file.originalFilename ?:
"file_${System.currentTimeMillis()}"
        val extension = originalName.substringAfterLast('.',
        "")

        val safeFileName =
"attach_${System.currentTimeMillis()}.${extension}"

        //save
        val filePath = File(uploadDir, safeFileName)
        file.transferTo(filePath)

        //link
        val requestHeader =
org.springframework.web.context.request.RequestContextHolder.cur
rentRequestAttributes() as
org.springframework.web.context.request.ServletRequestAttributes
        //val host = requestHeader.request.getHeader("Host")
        ?: "193.68.89.59:8081"
        val host = requestHeader.request.getHeader("Host")
        ?: "2.26.93.132:8081"

        val publicUrl =
"http://$host/uploads/attachments/$safeFileName"

        //type
        val isImage = extension.lowercase() in listOf("jpg",
"jpeg", "png", "gif", "webp")
        val type = if (isImage) "IMAGE" else "DOCUMENT"

        //2user
        return ResponseEntity.ok(mapOf(
            "url" to publicUrl,
            "name" to originalName,
            "size" to file.size, // Размер в байтах
            "type" to type
        ))
    }
}

```

```

        } catch (e: Exception) {
            e.printStackTrace()
            return ResponseEntity.status(500).body(mapOf("error"
to "Ошибка загрузки файла"))
        }
    }

    data class EditMessageRequest(
        val userId: Long,
        val newContent: String
    )

    @PutMapping("/message/{messageId}")
    fun editMessage(
        @PathVariable messageId: Long,
        @RequestBody request: EditMessageRequest
    ): ResponseEntity<Any> {
        val msg =
messageRepository.findById(messageId).orElse(null)
        ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Сообщение не
найден"))

        if (msg.senderId != request.userId) {
            return ResponseEntity.status(403).body(mapOf("error"
to "Нельзя редактировать чужие сообщения"))
        }

        if (!msg.attachmentUrl.isNullOrBlank()) {
            return
ResponseEntity.badRequest().body(mapOf("error" to "Сообщения с
вложениями нельзя редактировать, только удалять"))
        }

        msg.content = request.newContent
        messageRepository.save(msg)

        return ResponseEntity.ok(mapOf("message" to "Сообщение
успешно обновлено"))
    }

    @DeleteMapping("/message/{messageId}")
    fun deleteMessage(
        @PathVariable messageId: Long,
        @RequestParam userId: Long
    ): ResponseEntity<Any> {
        val msg =
messageRepository.findById(messageId).orElse(null)
        ?: return
ResponseEntity.badRequest().body(mapOf("error" to "Сообщение не
найден"))
    }

```

```

        if (msg.senderId != userId) {
            if (msg.groupId != null) {
                val group =
groupRepository.findById(msg.groupId!!).orElse(null)
                if (group == null || group.adminId != userId) {
                    return
ResponseEntity.status(403).body(mapOf("error" to "Нельзя удалять
чужие сообщения"))
                }
            } else {
                return
ResponseEntity.status(403).body(mapOf("error" to "Нельзя удалять
чужие сообщения"))
            }
        }

        if (!msg.attachmentName.isNullOrBlank()) {
            try {
                val file = File("uploads/attachments",
msg.attachmentName)
                if (file.exists()) file.delete()
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }

        messageRepository.delete(msg)
        return ResponseEntity.ok(mapOf("message" to "Сообщение
удалено"))
    }

    @PostMapping("/users/{userId}/fcm-token")
    fun updateFcmToken(@PathVariable userId: Long, @RequestBody
tokenDto: Map<String, String>): ResponseEntity<Any> {
        val token = tokenDto["token"]
        if (token.isNullOrBlank()) return
ResponseEntity.badRequest().build()
        val user = userRepository.findById(userId).orElse(null)
        ?: return ResponseEntity.notFound().build()
        user.fcmToken = token
        userRepository.save(user)
        return ResponseEntity.ok(mapOf("message" to "Токен
обновлен"))
    }

    @GetMapping("/groups/auto-id")
    fun getAutoGroupId(@RequestParam name: String, @RequestParam
userId: Long): ResponseEntity<Any> {
        if (name.isBlank()) return
ResponseEntity.badRequest().body(mapOf("error" to "Название
группы пустое"))
        val autoGroupName = "Беседа $name"

```

```

        var group =
groupRepository.findByNameAndIsAcademic(autoGroupName, true)
        val user = userRepository.findById(userId).orElse(null)

        if (group == null) {
            group = GroupChat(name = autoGroupName, adminId = -
1L, avatarUrl = "psu_logo", isAcademic = true)
            if (user != null) group.members.add(user)
            group = groupRepository.save(group)
        } else if (user != null &&
!group.members.contains(user)) {
            group.members.add(user)
            group = groupRepository.save(group)
        }
        return ResponseEntity.ok(mapOf("groupId" to group.id))
    }

    private fun handleAutoGroupJoin(user: User, studyGroupName:
String) {
        if (studyGroupName.isBlank()) return
        val autoGroupName = "Беседа $studyGroupName"
        var group =
groupRepository.findByNameAndIsAcademic(autoGroupName, true)
        if (group == null) {
            group = GroupChat(name = autoGroupName, adminId = -
1L, avatarUrl = "psu_logo", isAcademic = true)
        }
        group.members.add(user)
        groupRepository.save(group)
    }

    private fun handleAutoGroupLeave(user: User,
oldStudyGroupName: String) {
        if (oldStudyGroupName.isBlank()) return
        val autoGroupName = "Беседа $oldStudyGroupName"
        val group =
groupRepository.findByNameAndIsAcademic(autoGroupName, true) ?:
return
        group.members.remove(user)
        if (group.members.isEmpty()) {
            val groupMessages =
messageRepository.findAllByGroupId(group.id)
            messageRepository.deleteAll(groupMessages)
            groupRepository.delete(group)
        } else {
            groupRepository.save(group)
        }
    }
}

```

Файл «JwtUtil.kt»

					ПГУ1.090301.37.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		186

```

package com.tiledgram.messenger_server

import io.jsonwebtoken.Claims
import io.jsonwebtoken.Jwts
import io.jsonwebtoken.SignatureAlgorithm
import io.jsonwebtoken.security.Keys
import org.springframework.stereotype.Component
import java.nio.charset.StandardCharsets
import java.util.*
import javax.crypto.SecretKey

@Component
class JwtUtil {
    private val secretString =
        "TiltedGramSuperSecretKeyForJwtTokensMustBeAtLeast32BytesLong!"
    private val secretKey: SecretKey =
        Keys.hmacShaKeyFor(secretString.toByteArray(StandardCharsets.UTF_8))
    private val expirationTime: Long = 2678400000L

    fun generateToken(username: String, userId: Long): String {
        return Jwts.builder()
            .setSubject(username)
            .claim("userId", userId)
            .setIssuedAt(Date(System.currentTimeMillis()))
            .setExpiration(Date(System.currentTimeMillis() +
expirationTime))
            .signWith(secretKey)
            .compact()
    }

    fun extractUserId(token: String): Long {
        return extractAllClaims(token).get("userId",
Integer::class.java).toLong()
    }

    fun extractUsername(token: String): String {
        return extractAllClaims(token).subject
    }

    fun validateToken(token: String): Boolean {
        return try {
            val claims = extractAllClaims(token)
            !claims.expiration.before(Date())
        } catch (e: Exception) {
            false
        }
    }

    private fun extractAllClaims(token: String): Claims {
        return Jwts.parserBuilder()
            .setSigningKey(secretKey)
            .build()
            .parseClaimsJws(token)
            .body
    }
}

```

					ПГУ1.090301.37.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		187

```

    }
}

```

Файл «User.kt»

```

package com.tiledgram.messenger_server

import jakarta.persistence.*

@Entity
@Table(name = "users")
data class User(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long = 0,

    @Column(unique = true, nullable = false)
    var username: String,

    @Column(nullable = false)
    var fullName: String,

    @Column(unique = true, nullable = false)
    var email: String,

    @Column(nullable = false)
    val passwordHash: String,

    @Column(nullable = false)
    var faculty: String,

    @Column(nullable = false)
    var studyGroup: String,

    @Column(name = "avatar_url", length = 1000)
    var avatarUrl: String? = null,

    @Column(name = "fcm_token")
    var fcmToken: String? = null
)

```

Приложение В

(Обязательно)

Мобильное приложение «Мессенджер для студентов ПГУ» для ОС Android

Презентация

					ПГУ1.090301.37.001 ПЗ	Лист
						189
Изм.	Лист	№ докум.	Подпись	Дата		

Android - приложение «Мессенджер для студентов ПГУ»

Автор работы: студент группы 22ВВП2 Корнилов В.М.

Научный руководитель: доцент кафедры ВТ Кузнецов А.В.

Пенза, 2026

1

Актуальность проекта

В эпоху цифровизации образования использование популярных универсальных мессенджеров для обучения создает избыточный «информационный шум». Учебные чаты неизбежно смешиваются с развлекательным контентом, рекламой и личными переписками, что приводит к снижению концентрации студентов и регулярной потере важных академических уведомлений.

Разработка специализированного студенческого мессенджера решает эту проблему, предоставляя изолированную и безопасную среду, где коммуникация привязана к структуре вуза (факультетам и группам). Это позволяет оперативно обмениваться учебными материалами, эффективно координировать командную работу и минимизировать когнитивную нагрузку на учащихся за счет отсутствия стороннего контента.

2

Цели работы

Цель работы - разработать мобильное приложение-мессенджер для Android устройств, предназначенное для организации учебной коммуникации студентов.

В приложении должно быть реализовано:

- регистрация и авторизация, с возможностью выбора факультета и группы;
- создание и редактирование личного профиля студента;
- обмен текстовыми сообщениями в рамках личных и групповых бесед;
- прикрепление и передача файлов, их просмотр и загрузка;
- настройка и получение push-уведомлений о новых сообщениях;
- поиск необходимой информации по истории переписки;
- интуитивно понятный пользовательский интерфейс;

3

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.37.001 ПЗ

Лист

190

Обзор аналогов

Telegram



Достоинства:

- Широкие возможности модерации
- Удобная работа с файлами
- Кроссплатформенность

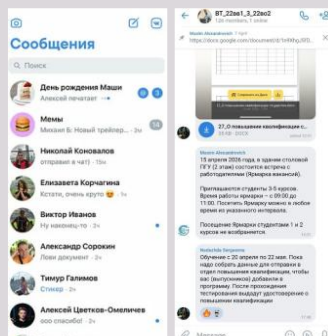
Недостатки:

- Переизбыток информации
- Нестабильность работы
- Наличие платной подписки

4

Обзор аналогов

VK



Достоинства:

- Привычная среда
- Интеграция с сервисами
- Прозрачность контактов

Недостатки:

- Перегруженность
- Работа с файлами

5

Обзор аналогов

MAX



Достоинства:

- Многофункциональность
- Работа с файлами
- Единая цифровая экосистема

Недостатки:

- Перегруженность
- Репутация

6

Изм.	Лист	№ докум.	Подпись	Дата

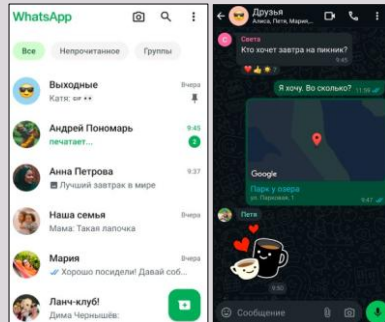
ПГУ1.090301.37.001 ПЗ

Лист

191

Обзор аналогов

WhatsApp



Достоинства:

- Простота использования
- Синхронизация с контактами
- Полностью бесплатный

Недостатки:

- Ограничения на передачу файлов
- Нестабильность работы

7

Стек технологий



8

UML-диаграмма вариантов использования



9

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.37.001 ПЗ

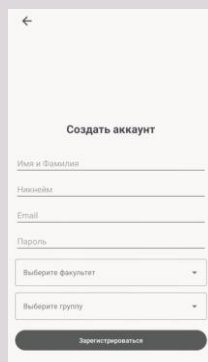
Лист

192

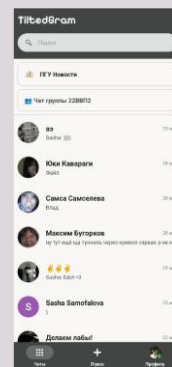
Интерфейс приложения



Экран авторизации



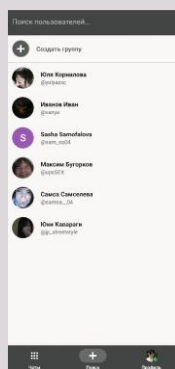
Экран регистрации



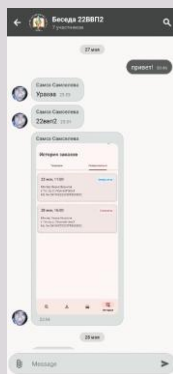
Главный экран

10

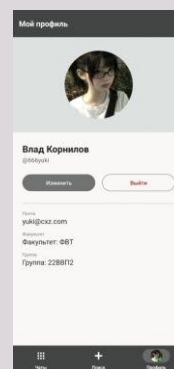
Интерфейс приложения



Поиск контактов



Экран переписки

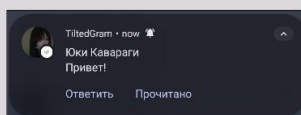


Просмотр профиля

11

Тестирование

Разработанный комплекс успешно прошёл полный цикл интеграционного и системного тестирования, охвативший 12 ключевых тест-кейсов. В ходе испытаний была проверена корректность клиентской валидации ввода, обработка пустых полей при авторизации, а также общая стабильность и скорость выполнения сетевых REST-запросов к API бэкенда через библиотеку Retrofit.



Успешно доставленное уведомление

12

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.37.001 ПЗ

Лист

193

Заключение

Результатом выполнения данной выпускной квалификационной работы можно считать успешно разработанный клиент-серверный комплекс, полностью соответствующее заданию.

Разработанное приложение является готовым самостоятельным инструментом, готовым к эксплуатации.

В перспективе планируется: интеграция мессенджера с ЭИОС, добавление модулей голосовой связи и внедрение сквозного (end-to-end) шифрования чатов.

13

Спасибо за внимание!

14

					<i>ПГУ1.090301.37.001 ПЗ</i>	Лист
						194
Изм.	Лист	№ докум.	Подпись	Дата		