

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»

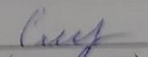
БАКАЛАВРСКАЯ РАБОТА

на тему

КОРПОРАТИВНЫЙ МЕССЕНДЖЕР ПОД ОС

ANDROID

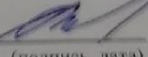
Студент

 Субботкин Максим Владимирович
(подпись, дата) 29.05.2023.

Руководитель

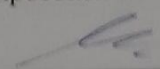
 Юрова Ольга Викторовна
(подпись, дата) 1.06.2023.

Нормоконтролёр

 Бычков Андрей Станиславович
(подпись, дата) 1.06.2023.

Работа допущена к защите (протокол зас. кафедры от 02.06.23 г. № 14)

Заведующий кафедрой


(подпись)

М.А. Митрохин

(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

14.06.2023, № 9)

Секретарь ГЭК


(подпись)

Е.И. Гурий

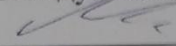
(фамилия, инициалы)

Пенза 2023

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«26» 04 2023 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Субботкин Максим Владимирович гр. 19ВВП2 факультета вычислительной техники

2. Тема работы «Корпоративный мессенджер под ОС Android»

Тема утверждена приказом ПГУ №458/о от "26" апреля 2023г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Юрова О. В

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

Назначение разработки – корпоративный мессенджер

Разработать программное обеспечение для общения пользователей в пределах компании под управлением ОС Android. Клиентская часть программного обеспечения должна предоставлять пользователю возможность обмениваться текстовыми и голосовыми сообщениями, файлами, создавать чаты и группы чатов, менять настройки пользователя.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области, определить архитектуру приложения, выбрать инструменты для разработки, реализовать приложение.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10 %	26.04.23 - 29.04.23	<i>ф.о.в.</i>
2.	Выбор архитектуры приложения и технологий, применяемых при проектировании и разработке	15 %	30.04.23 - 04.05.23	<i>ф.о.в.</i>
3.	Разработка разметки	20 %	05.05.23 - 09.05.23	<i>ф.о.в.</i>
4.	Разработка логики приложения	25 %	10.05.23 - 21.05.23	<i>ф.о.в.</i>
5.	Тестирование и отладка	15 %	22.05.23 - 24.05.23	<i>ф.о.в.</i>
6.	Выполнение задания по БЖД	5 %	25.05.23 - 27.05.23	<i>ф.о.в.</i>
7.	Оформление выпускной работы	10 %	28.05.23 - 31.05.23	<i>ф.о.в.</i>

Дата выдачи задания "26" апреля 2023г.

Руководитель бакалаврской работы 26.04.23 *ф.о.в.* Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент 26.04.23 *С.м.в.* Субботкин М.В.
(подпись, дата)

Работу к защите допустить
Декан факультета *Л.П.* 06.06.2023 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 125 л., 16 рис., 4 источ., 3 прил.
ANDROID, БАЗА ДАННЫХ, KOTLIN, FIREBASE, API

Цель работы – разработать мобильное приложение, позволяющее пользователям обмениваться различными видами сообщений.

В ходе разработки необходимо реализовать следующие возможности для пользователя: обмениваться текстовыми и голосовыми сообщениями, файлами, создавать чаты и группы чатов, менять настройки пользователя.

В процессе разработки проводился анализ актуальности проекта, аналогов, способов разработки мобильного ПО.

В результате проведенной работы создано приложение, соответствующее поставленной задаче.

				ПГУ1.090301.33.001 ПЗ				
		№ докум.	Подпись	Корпоративный мессенджер под ОС Android Пояснительная записка		Лит.	Лист	Листов
Разраб.	Субботкин М.В.							
Провер.	Юрова О.В.						2	125
Н. Контр.	Бычков А.С.							
Утверд.	Митрохин М.А.							
						ФВТ, каф. ВТ гр. 19ВВП2		

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Корпоративный мессенджер под управление ОС Android	7
1.1 Анализ предметной области	7
1.1.1 Обзор аналогов	8
1.1.2 Выбор инструментов разработки	9
1.1.3 Постановка задачи.....	11
1.2 Архитектура приложения.....	11
1.3 Реализация приложения	14
1.3.1 Реализация аутентификации пользователя	14
1.3.2 Реализация настроек пользователя	17
1.3.3 Реализация списка контактов	20
1.3.4 Реализация одиночного чата.....	25
1.3.5 Реализация групповых чатов	33
1.3.6 Реализация главного окна приложения	38
1.4 Руководство пользователя.....	45
2 Безопасность жизнедеятельности	50
2.1 Потенциально опасные факторы при работе с ПЭВМ	50
2.2 Классификация помещения	51
2.3 Санитарно-гигиеническая характеристика	51
2.4 Расчет освещенности рабочего места	52
Заключение.....	55

Список использованных источников	56
Приложение А.....	57
Приложение Б	62
Приложение В.....	119

Перечень сокращений и обозначений

БД – база данных

ОС – операционная система

ПК – персональный компьютер

ПЭВМ – персональная электронно-вычислительная машина

ЭВМ – электронно-вычислительная машина

API (Application Programming Interface) – программный интерфейс приложения

IDE (Integrated Development Enviroment) – Интегрированная среда разработки

MVC (Model-View-Controller) – «Модель-Представление-Контроллер» или
«Модель-Вид-Контроллер»

UML (Unified Modeling Language) – унифицированный язык моделирования.

XML (Extensible Markup Language) – расширяемый язык разметки

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		5

Введение

Современный мир требует от предприятий и компаний быстрой и эффективной коммуникации между сотрудниками. В связи с этим, все большую популярность набирают корпоративные мессенджеры, которые позволяют ускорить и улучшить внутреннюю коммуникацию в организации.

Существует множество инструментов, которые предназначены для обеспечения внутренней коммуникации между сотрудниками, такие как электронная почта, голосовая и видеосвязь, а также корпоративные социальные сети. Но наиболее популярным и удобным способом обмена сообщениями являются мессенджеры.

Целью данной дипломной работы является разработка собственного корпоративного мессенджера, который будет отвечать всем требованиям современных компаний и их сотрудников. В рамках работы будет проведен анализ существующих решений на рынке, определены их достоинства и недостатки, а также сформулированы требования к создаваемому мессенджеру. Далее будет разработана архитектура приложения, реализованы функциональные возможности и проведено тестирование разработанного мессенджера.

Результатом данной работы будет создание функционального и удобного в использовании корпоративного мессенджера, который поможет улучшить коммуникацию между сотрудниками компании и повысить эффективность их работы.

1 Корпоративный мессенджер под управление ОС Android

1.1 Анализ предметной области

Предметная область разработки корпоративного мессенджера относится к области информационных технологий и включает в себя разработку программного обеспечения для внутренней коммуникации между сотрудниками компаний различных масштабов и отраслей.

Корпоративные мессенджеры являются эффективным инструментом для организации коммуникации и совместной работы сотрудников, работающих в разных офисах, городах или даже странах. Они позволяют сократить время ответа на вопросы, быстро решать проблемы, улучшить процессы совместной работы и повысить эффективность командной работы.

В предметную область разработки корпоративного мессенджера входят следующие аспекты:

- функциональность. Корпоративный мессенджер должен предоставлять все необходимые функции для эффективной коммуникации между сотрудниками, такие как возможность обмена текстовыми сообщениями, файлами, голосовыми звонками, а также удобную организацию рабочих групп и чатов.
- безопасность. Корпоративный мессенджер должен обеспечивать высокий уровень безопасности передачи данных, а также гарантировать конфиденциальность сообщений и документов.
- доступность и мобильность.
- удобство использования. Корпоративный мессенджер должен быть интуитивно понятен и удобен в использовании, чтобы максимально упростить взаимодействие между сотрудниками и не требовать дополнительного времени и обучения для освоения.

1.1.1 Обзор аналогов

На сегодняшний день существует большое количество мобильных корпоративных мессенджеров, которые предлагают широкий спектр функций для обмена сообщениями, голосовой и видеосвязи, а также интеграцию с другими системами.

Одним из наиболее популярных мессенджеров является *Slack*. Он предоставляет возможность создания каналов для общения сотрудников внутри компании, интеграцию с другими сервисами и платформами, и поддерживает наличие мобильных приложений для удобства использования на смартфонах.

К минусам *Slack* можно отнести низкую производительность, большое потребление оперативной памяти, отсутствие русского языка, ограниченность функционала в бесплатной версии.

К плюсам *Slack* можно отнести простоту и удобство использования, высокую степень интеграции с другими сервисами, возможность участия неограниченного числа пользователей.

Другим известным мессенджером является *Microsoft Teams*, который предлагает широкий набор функций для коммуникации и совместной работы сотрудников, таких как голосовая и видеосвязь, обмен файлами, интеграция с *Microsoft Office* и другими приложениями.

К минусам *Microsoft Teams* можно отнести работу с ошибками, не самый простой функционал и сложный интерфейс.

К плюсам *Microsoft Teams* можно отнести доступ к веб-версиям *Word*, *Excel*, *PowerPoint*, видеозвонки.

VK Teams — это корпоративный мессенджер, разработанный командой социальной сети ВКонтакте для коммуникации внутри организаций и команд.

К минусам *VK Teams* можно отнести большое количество ошибок и частые обновления, разрывы соединения, проблемы с отправкой сообщений.

К плюсам *VK Teams* можно отнести шифрование чатов и звонков, наличие чат-ботов, наличие бесплатного тарифа.

1.1.2 Выбор инструментов разработки

На данный момент в мире информационных технологий существует огромное количество различных средств разработки приложений на *Android*.

На сегодняшний день самыми востребованными средами являются:

- *IntelliJ Idea*;
- *Android Studio*.

Каждая *IDE* обладает уникальными характеристиками. О них ниже.

IntelliJ Idea разработала российская компания *JetBrains*. Подобно *Eclipse*, эта среда позволяет создавать приложения и программы на нескольких программных языках. Также *IntelliJ Idea* не нагружает ПК.

Сильными сторонами *IntelliJ Idea* являются:

- более оперативная отладка значений;
- присутствие автозаполнителя методов;
- понятный и лаконичный интерфейс;
- подходит для тех, кто программирует на *Kotlin*.

Самый большой недостаток *IntelliJ Idea* – её модель распространения, которая является платной и распространяется по лицензии.

Android Studio – интегрированная среда разработки для работы с платформой *Android*, разработанная *Google*. Она представляет собой набор интегрированных инструментов для эффективной разработки, отладки и тестирования программ, позволяя создать проект любой сложности, используя стандартные шаблоны *Activity* и наборы все возможных элементов, а также позволяет эмулировать устройство на базе ОС *Android* (от планшета до мобильного телефона). Это достаточно удобно, так как можно увидеть, как будет выглядеть программа на разных устройствах. Самым большим преимуществом *Android Studio* является её распространение. Почти все

Android-разработчики используют *Android Studio*. Благодаря этому можно без труда найти интересующую по данной среде информацию в интернете.

Единственным недостатком *Android Studio* является её нагрузка на ПК в процессе разработки и запуска эмуляторов.

Однако достоинств у *Android Studio* гораздо больше:

- наличие *XML* редактора;
- поддержка всех последних технологий, которые можно использовать при разработке приложения для *Android*
- возможность эмуляции различных устройств;
- поддержка рендера средствами *GPU*;
- обширная база примеров проектирования (*Samples Browser*).

Проанализировав достоинства и недостатки представленных сред разработки, было принято решение использовать *Android Studio*.

Поскольку основной задачей приложения является сбор и хранение информации, было необходимо найти удобную и безопасную БД для её хранения. *Firebase* отвечает всем этим требованиям.

Особенности *Firebase* :

Он предоставляет облачный сервис, поэтому никаких настроек не требуется;

Данные хранятся в собственном формате *JSON*, поэтому пользователи могут видеть, что они сохранили;

Данные в безопасности, потому что *Firebase* требует 2048-битное шифрование *SSL* для всех передач данных;

Он прекрасно интегрируется с фреймворками. Так что это позволяет создать приложение в очень короткое время;

Хранение файлов при поддержке *Google Cloud Storage*.

Наличие *Realtime Database*, что позволяет оперативно обновлять данные пользователей при их изменении

Он предоставляет различные методы аутентификации пользователя

В итоге *Firebase* имеет в своем распоряжении все нужные инструменты и функции, которые позволяют создать мощные приложение для *Android*.

1.1.3 Постановка задачи

Задачей данного дипломного проекта является создание мобильного приложения под управлением ОС *Android*. Приложение должно предоставлять пользователям возможность обмениваться текстовыми сообщениями, голосовыми сообщениями, файлами, создавать чаты и группы чатов, менять настройки пользователя.

1.2 Архитектура приложения

Выбор хорошей архитектуры приложения невероятно важная вещь, поскольку правильно созданная архитектура при необходимости позволяет вносить изменения с минимальными затратами ресурсов.

При выборе архитектуры приложения было решено воспользоваться *MVC* архитектурой, её суть заключается в разделении данных и логики приложения на 3 уровня (Модель-Вид-Контроллер) [1].

Разработанная архитектура будущего приложения представлена на рисунке 1.

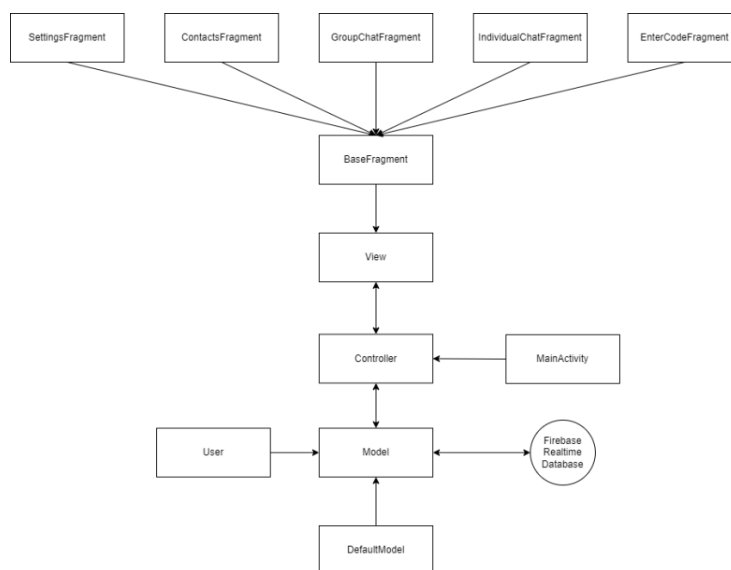


Рисунок 1 – Архитектура приложения

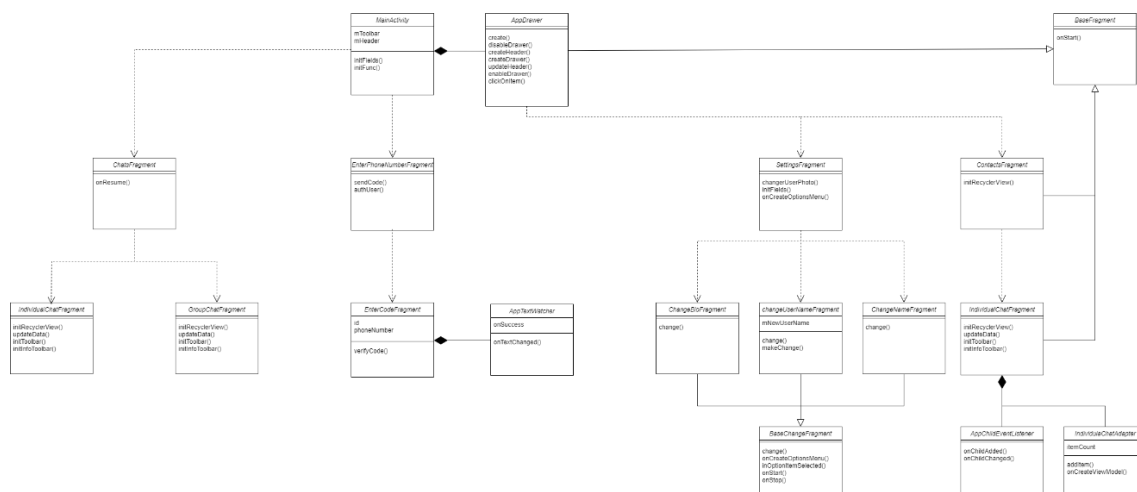


Рисунок 2 - Uml-диаграмма классов

Разбираем данную диаграмму:

- *MainActivity* – класс, который создает единственную *Activity* в приложении, привязывает *Toolbar* и выдвигающееся меню, создает подключение с *Firebase*;
- *EnterPhoneFragment* – класс, который отвечает за ввод телефона пользователя и передачу его в БД для последующего подтверждения;
- *EnterCodeFragment* – класс, который отвечает за вход пользователя в приложение посредством ввода проверочного кода, который пришел на мобильное устройство;
- *AppTextWatcher* – вспомогательный класс для *EnterCodeFragment*, который отвечает за проверку правильности ввода и перехода и авторизацию;
- *AppDrawer* – вспомогательный класс, отвечающий за инициализацию полей выдвижного меню и информации о пользователе;
- *SettingsFragment* – класс, создающий меню изменения настроек, который использует *ChangeBioFragment* для изменения биографии пользователя, *ChangeUserNameFragment* для изменения имени пользователя

и *ChangeNameFragment* для изменения настоящего имени пользователя, все три класса наследуются от *BaseChangeFragment*;

– *ContactsFragment* – класс, создающий фрагмент, в котором пользователь может просматривать людей из списка своих контактов, которые пользуются приложением, внутри используется *IndividualChatFragment*, который создает с пользователем из контактов чат для общения, он использует *AppChildEventListener* и *IndividualChatAdapter* как вспомогательные классы для отображения и загрузки сообщений[4];

– *GroupChatFragment* – класс, отвечающий за создание фрагмента групповых чатов.

Для того, чтобы использовать приложение, пользователь должен авторизоваться, затем он сможет пользоваться всем функционалом приложения, который представлен на рисунке 3.

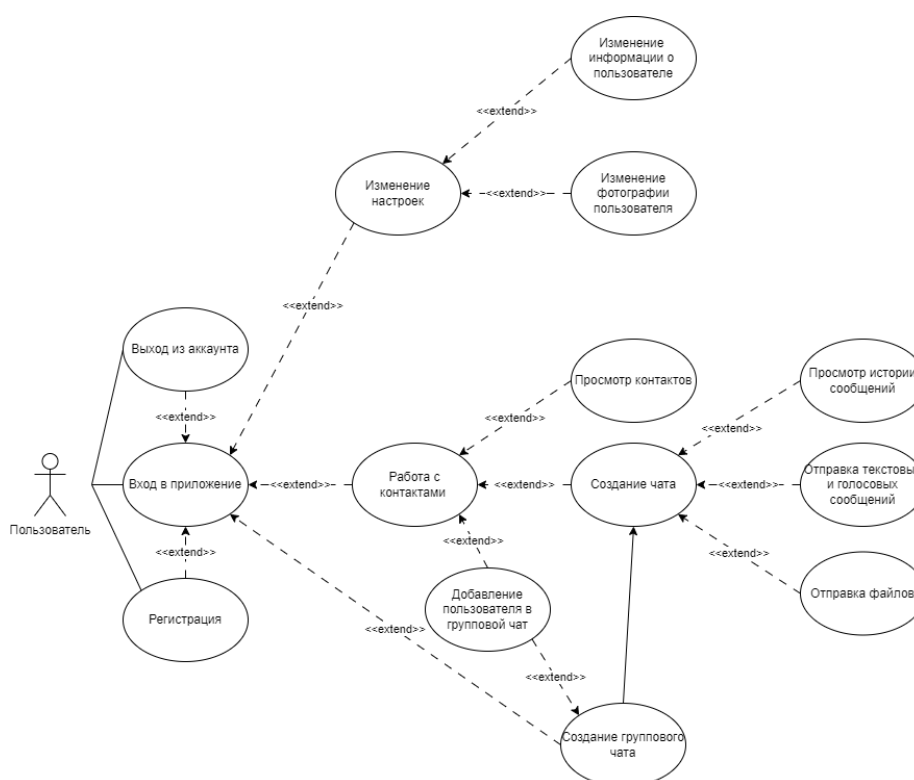


Рисунок 3 - Uml-диаграмма вариантов использования

1.3 Реализация приложения

1.3.1 Реализация аутентификации пользователя

Была создана система аутентификации пользователей на основе *Firebase*.

Аутентификация *Firebase* — это функция аутентификации пользователей, предоставляемая *Firebase* в качестве ее серверных служб. Она создана на основе токенов, это обеспечивает легкую интеграцию с большинством платформ. Еще одна замечательная особенность здесь заключается в том, что эта аутентификация легко интегрируется с API от *Firebase*, что делает возможным запустить эту функцию очень быстро и легко.

Аутентификация использует метод *PhoneAuthProvider()*, который принимает на вход номер телефона и код, которые берутся из полей *EditText*. Для того, чтобы не передать пустую строку в этот метод была сделана проверка на заполнение [2].

После того как пользователь введет 6 символов, которые пришли ему в сообщении, произойдет проверка и, если код правильный, пользователь перейдет в основной фрагмент приложения, данный фрагмент представлен на рисунке 4.



Рисунок 4 - Ввод кода подтверждения

Код фрагмента для авторизации пользователей представлен ниже:

```
private fun sendCode() {
    if (register_input_phone.text.toString().isEmpty()) {
        showToast("Введите номер телефона")
    } else {
        authUser()
    }
}

private fun authUser() {
    mPhoneNumber = register_input_phone.text.toString()

    PhoneAuthProvider.verifyPhoneNumber(
        PhoneAuthOptions.newBuilder()
            .setPhoneNumber(mPhoneNumber)
            .setTimeout(60, TimeUnit.SECONDS)
            .setActivity(ACTIVITY)
            .setCallbacks(mCallback)
            .build()
    )
}

override fun onStart() {
    super.onStart()
    mCallback = object :
PhoneAuthProvider.OnVerificationStateChangedCallbacks() {
        override fun onVerificationCompleted(credential:
PhoneAuthCredential) {

AUTH.signInWithCredential(credential).addOnCompleteListener() {
            if (it.isSuccessful){
                showToast("Добро пожаловать")
                restartActivity()
            }
            else
                showToast(it.exception?.message.toString())
        }
    }

    override fun onVerificationFailed(p0: FirebaseException)
{
        showToast(p0.message.toString())
    }

    override fun onCodeSent(id: String, token:
PhoneAuthProvider.ForceResendingToken) {
        switchFragment(EnterCodeFragment(mPhoneNumber , id))
    }
}
}
```

```

        register_enter_phone_btn_next.setOnClickListener {

            sendCode()

        }

    }

    private fun verifyCode() {
        val code = register_input_code.text.toString()
        val credential = PhoneAuthProvider.getCredential(id, code)

        AUTH.signInWithCredential(credential).addOnCompleteListener() {
            if (it.isSuccessful) {
                val uid = AUTH.currentUser?.uid.toString()
                var dataMap = mutableMapOf<String, Any>()
                dataMap[CHILD_ID] = uid
                dataMap[CHILD_PHONE] = phoneNumber
                dataMap[CHILD_USERNAME] = uid

                REF_DATABASE_ROOT.child(NODE_PHONES).child(phoneNumber).setValue
                (uid)

                    .addOnSuccessListener {

                REF_DATABASE_ROOT.child(NODE_USERS).child(uid).updateChildren(da
                taMap)

                    .addOnSuccessListener {
                        showToast("Добро пожаловать")
                        restartActivity()
                    }

            }

            } else
                showToast(it.exception?.message.toString())
        }
        showToast("OK")
    }

}

override fun onStart() {
    super.onStart()
    ACTIVITY.title = phoneNumber
    register_input_code.addTextChangedListener(AppTextWatcher {

        val string = register_input_code.text.toString()
        if (string.length == 6) {
            verifyCode()
        }

    })
}

```

1.3.2 Реализация настроек пользователя

В окне настроек пользователь может поменять своё фото, номер телефона, своё полное имя, имя пользователя, и написать о себе, также пользователь имеет возможность выйти из приложения.

За заполнение информации о пользователе отвечает метод *initFields*, который загружает информацию о пользователе из БД и заполняет поля настроек.

На каждый элемент установлен слушатель нажатий, при нажатии на который выполняется функция *switchFragment*, в параметрах которого указывается фрагмент, на который нужно перейти для изменения нужного поля, после изменения информация сохранится в БД.

Для изменения фотографии пользователя отвечает метод *changeUserPhoto*, который использует библиотеку *CropImage*, при нажатии на кнопку изменения фотографии пользователь может выбрать изображение из проводника или сделать снимок с камеры, после этого фотография сохраняется в Firebase Storage, а в БД сохраняется *URL* на эту фотографию [2]. После этого вызывается метод *updateHeader*, который отвечает за то, чтобы после изменения данных пользователя обновился также и *Header* в выдвижном меню.

Функции *onCreateOptionsMenu* и *onOptionsItemSelected* отвечают за обработку нажатий по кнопке меню, которая находится в верхнем правом углу приложения. Фрагмент настроек представлен на рисунке 5.

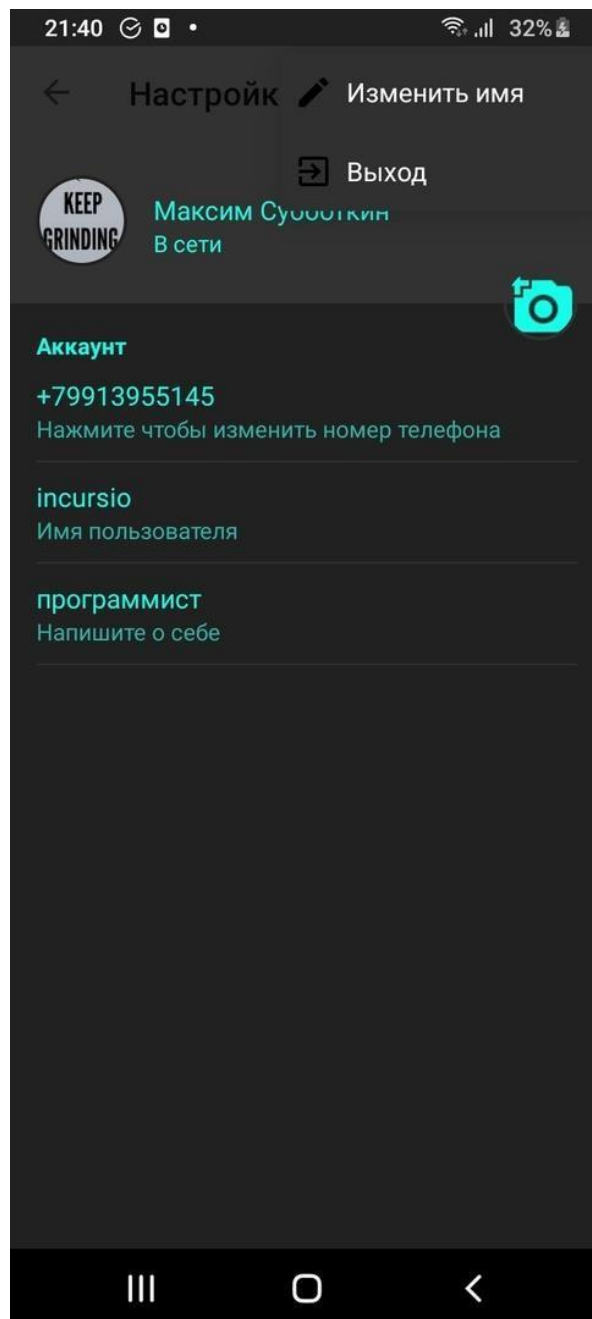


Рисунок 5- Фрагмент настроек

Код реализации фрагмента настроек представлен ниже:

```
class SettingsFragment :
BaseFragment(R.layout.fragment_settings) {

    override fun onResume() {
        super.onResume()
        ACTIVITY.title = "Настройки"
        setHasOptionsMenu(true)
        initFields()
    }
}
```

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		18

```

private fun initFields() {

    settings_bio.text = USER.bio
    settings_full_name.text = USER.fullname
    settings_phone_number.text = USER.phone
    settings_status.text = USER.state
    settings_username.text = USER.username
    settings_btn_change_username.setOnClickListener {
switchFragment(ChangeUsernameFragment()) }
    settings_btn_change_bio.setOnClickListener {
switchFragment(ChangeBioFragment()) }
    settings_change_photo.setOnClickListener {
changeUserPhoto()}
    settings_user_photo.downloadAndSetImage(USER.photoUrl)
}

private fun changeUserPhoto() {
    CropImage.activity()
        .setGuidelines(CropImageView.Guidelines.ON)
        .setMultiTouchEnabled(true)
        .setAspectRatio(1, 1)
        .setRequestedSize(250, 250,
CropImageView.RequestSizeOptions.RESIZE_INSIDE)
        .setMinCropWindowSize(0, 0)
        .setCropShape(CropImageView.CropShape.OVAL)
        .start(ACTIVITY, this)

}

    override fun onCreateOptionsMenu(menu: Menu, inflater:
MenuInflater) {

activity?.menuInflater?.inflate(R.menu.settings_action_menu,
menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean
{
    when (item.itemId) {
        R.id.settings_menu_exit -> {
            States.updateState(States.OFFLINE)
            AUTH.signOut()
            restartActivity()
        }
        R.id.settings_menu_change_name -> {
            switchFragment(ChangeNameFragment())
        }
    }
    return true
}
}

```

```

        override fun onActivityResult(requestCode: Int, resultCode:
Int, data: Intent?) {
            super.onActivityResult(requestCode, resultCode, data)
            if (requestCode ==
CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE && resultCode ==
RESULT_OK && data != null) {
                val uri = CropImage.getActivityResult(data).uri
                val path =
REF_STORAGE_ROOT.child(FOLDER_PROFILE_IMAGE).child(UID)

                putFileToStorage(uri, path) {
                    getUrlStorage(path) {
                        putUrlIntoDatabase(it) {

settings_user_photo.downloadAndSetImage(it)
                            showToast("Данные обновлены")
                            USER.photoUrl = it
                            ACTIVITY.mAppDrawer.updateHeader()

                        }
                    }
                }
            }
        }

        override fun onPause() {
            super.onPause()
        }
    }
}

```

1.3.3 Реализация списка контактов

При открытии приложения контактов приложение запрашивает разрешение на считывание контактов телефона и проверяет разрешение с помощью функции *checkPermission*, которая представлена ниже.

```

fun checkPermission(permission: String): Boolean {
    return if (Build.VERSION.SDK_INT >= 23 &&
ContextCompat.checkSelfPermission(
        ACTIVITY,
        permission
    ) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(ACTIVITY,
arrayOf(permission), PERMISSION_REQUEST)
        false
    } else true
}

```

Данная функция является универсальной и применяется в моём приложении для проверки всех разрешений, которые становятся необходимы по мере использования приложения. Также в файле `AndroidManifest` нужно прописать соответствующее разрешение для считывания контактов. Считывание контактов происходит в отдельном потоке и не мешает работе приложения.

За отображение контактов отвечает метод `initRecyclerView`, который работает с `XML` файлом и создает `Adapter` и `Holder` для отображения. Приложение проверяет телефоны, считанные с телефонной книги и телефоны, которые хранятся в БД и в зависимости от этого отображаются контакты во фрагменте, за это отвечает функция `initContacts` и `updatePhonesDatabase`, которые представлены ниже.

```
fun initContacts() {
    if (checkPermission(READ_CONTACTS)) {
        var contactsArray = arrayListOf<DefaultModel>()
        val cursor = ACTIVITY.contentResolver.query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
            null,
            null,
            null,
            null
        )
        cursor?.let {
            while (it.moveToNext()) {
                val fullName =

it.getString(it.getColumnIndexOrThrow(ContactsContract.Contacts.
DISPLAY_NAME))
                val phone =

it.getString(it.getColumnIndexOrThrow(ContactsContract.CommonDat
aKinds.Phone.NUMBER))
                val newModel = DefaultModel()
                newModel.fullname = fullName
                newModel.phone = phone.replace(Regex("[\\s,-]"),
                "")
                contactsArray.add(newModel)
            }
        }

        cursor?.close()
        updatePhonesDatabase(contactsArray)
    }
}
```

```

    }
}

fun updatePhonesDatabase(contactsArray: ArrayList<DefaultModel>)
{
    if (AUTH.currentUser != null) {

REF_DATABASE_ROOT.child(NODE_PHONES).addListenerForSingleValueEv
ent(AppValueEventListener {
    it.children.forEach { dataSnapshot ->
        contactsArray.forEach { contact ->
            if (dataSnapshot.key == contact.phone) {

REF_DATABASE_ROOT.child(NODE_PHONES_CONTACTS).child(UID)

.child(dataSnapshot.value.toString()).child(CHILD_ID)

.setValue(dataSnapshot.value.toString())
                .addOnFailureListener {
                    showToast("Error")
                }

REF_DATABASE_ROOT.child(NODE_PHONES_CONTACTS).child(UID)

.child(dataSnapshot.value.toString()).child(CHILD_FULLNAME)
                .setValue(contact.fullname)
                .addOnFailureListener {
                    showToast("Error")
                }

            }
        }
    }
})
}
}

```

В итоге пользователь видит контакты людей, которые зарегистрировались в приложении и находятся у него в списке контактов , рисунок 6.

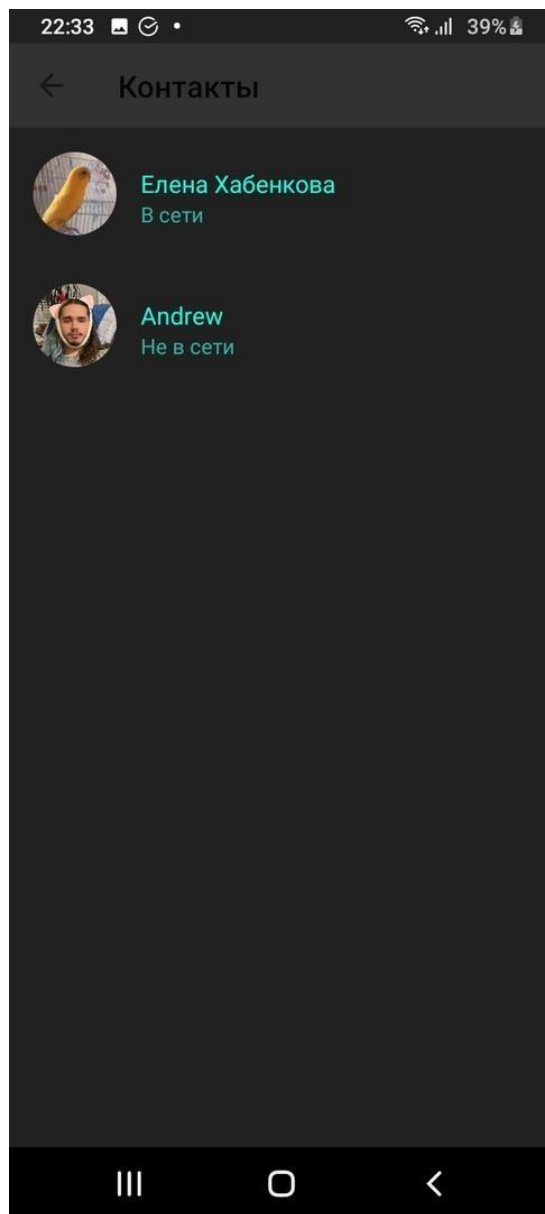


Рисунок 6 - Список контактов

Код фрагмента контактов представлен ниже:

```
class ContactsFragment :
    BaseFragment(R.layout.fragment_contacts) {

    private lateinit var mRecyclerView : RecyclerView
    private lateinit var mAdapterer :
        FirebaseRecyclerAdapter<DefaultModel, ContactsHolder>
    private lateinit var mRefToContacts : DatabaseReference
    private lateinit var mRefToUsers : DatabaseReference
    private lateinit var mRefToUserListener :
        AppValueEventListener
    private var mapOfListeners = HashMap<DatabaseReference ,
        AppValueEventListener>()
```

```

        override fun onResume() {

super.onResume()

        ACTIVITY.title = "Контакты"

initRecyclerView()
    }

    private fun initRecyclerView() {
        mRecyclerView = contacts_recycler_view
        mRefToContacts =
REF_DATABASE_ROOT.child(NODE_PHONES_CONTACTS).child(UID)

        val options =
FirebaseRecyclerOptions.Builder<DefaultModel>()
            .setQuery(mRefToContacts, DefaultModel::class.java)
            .build()
        mAdapter = object :
FirebaseRecyclerViewAdapter<DefaultModel, ContactsHolder>(options){
            override fun onCreateViewHolder(parent: ViewGroup,
viewType: Int): ContactsHolder {
                val view =
LayoutInflater.from(parent.context).inflate(R.layout.contact_ite
m , parent,false)
                return ContactsHolder(view)
            }

            override fun onBindViewHolder(
                holder: ContactsHolder,
                position: Int,
                model: DefaultModel
            ) {
                mRefToUsers =
REF_DATABASE_ROOT.child(NODE_USERS).child(model.id)
                mRefToUserListener = AppValueEventListener {
                    val contact = it.getDefaultModel()

                    if (contact.fullname.isEmpty()){
                        holder.name.text = model.fullname
                    }
                    else holder.name.text = contact.fullname

                    holder.name.text = contact.fullname
                    holder.status.text = contact.state

                    holder.photo.downloadAndSetImage(contact.photoUrl)
                    holder.itemView.setOnClickListener {

switchFragment(IndividualChatFragment(model))
                }
            }
        }
    }

```

```

        mRefToUsers.addValueEventListener(mRefToUserListener)
        mapOfListeners[mRefToUsers] = mRefToUserListener

    }

}

mRecyclerView.adapter = mAdapter
mAdapter.startListening()
}

class ContactsHolder(view :
View):RecyclerView.ViewHolder(view){
    val name : TextView = view.contact_fullname
    val status : TextView = view.contact_status
    val photo : CircleImageView = view.contact_photo
}

override fun onPause() {
    super.onPause()
    mAdapter.stopListening()
    mapOfListeners.forEach {
        it.key.removeEventListener(it.value)
    }
}
}

```

1.3.4 Реализация одиночного чата

При нажатии на пользователя в списках контактов с ним открывается одиночный чат, в котором пользователь имеет возможность обмениваться текстовыми сообщениями, изображениями, файлами и голосовыми сообщениями, удалять и очищать чат. После отправки первого сообщения чат автоматически попадёт на главный фрагмент как пользователю, так и собеседнику.

Метод *initFields* отвечает за начальную инициализацию всех полей чата, таких как данные о собеседнике и подготовка всех слушателей.

Методы *attach*, *attachImage* и *attachImage* отвечают за отправку изображений и файлов. После отправки сообщения оно сохраняется в БД, для отображения сообщения используется *RecyclerView + View* и *Holder* для

отображения в нём различных видов сообщений, сам *RecyclerView* был оптимизирован таким образом, чтобы загружать из базы данных только определенное количество сообщений, которое будет видно пользователю, а уже при прокрутке вверх загружать из БД остальные сообщения.

Для создания *Holder* для различных видов сообщений используются классы *FileMessageHolder*, *ImageMessageHolder*, *TextMessageHolder*, *VoiceMessageHolder*, также общий интерфейс *MessageHolder* и фабрика *HolderFactory*, для создания *View* используются классы *ViewFileMessage*, *ViewImageMessage*, *ViewTextMessage*, *ViewVoiceMessage*, также общий интерфейс *MessageView* и фабрика *ViewFactory* [3].

Методы *onOptionsItemSelected* и *onOptionsItemSelected* отвечают за обработку нажатий по элементам меню, которое находится в верхнем правом углу чата, пример чата представлен на рисунке 7.

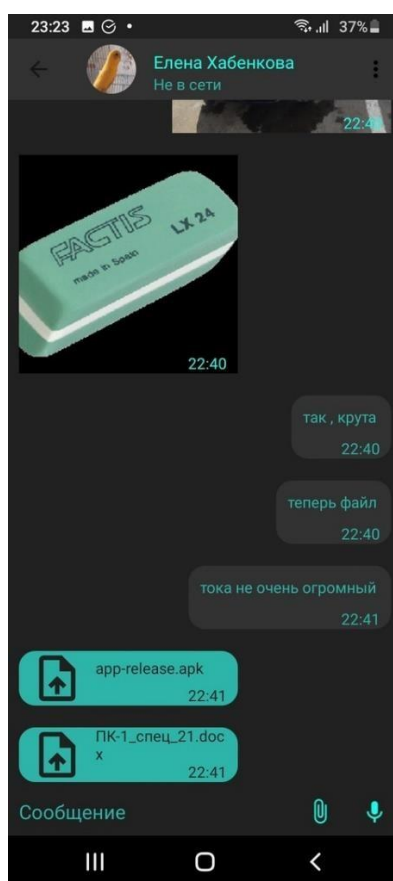


Рисунок 7 - Фрагмент чата

Код фрагмента одиночного чата представлен ниже:

```
class IndividualChatFragment(private val contact: DefaultModel)
:
    BaseFragment(R.layout.fragment_individual_chat) {

    private lateinit var mListenerInfoToolbar:
AppValueEventListener
    private lateinit var mReceivingUser: User
    private lateinit var mToolBarInfo: View
    private lateinit var mRefToUser: DatabaseReference
    private lateinit var mRefToMessages: DatabaseReference
    private lateinit var mAdapter: IndividualChatAdapter
    private lateinit var mRecyclerView: RecyclerView
    private lateinit var mMessagesListener:
AppChildEventListener
    private var mCountMessagesToLoad = 10
    private var mIsUserScrolling = false
    private var mSmoothScrollToPosition = true
    private lateinit var mSwipeRefreshLayout: SwipeRefreshLayout
    private lateinit var layoutManager: LinearLayoutManager
    private lateinit var mAppVoiceRecorder: AppVoiceRecorder
    private lateinit var mBottomSheetBehavior:
BottomSheetBehavior<*>

    override fun onResume() {
        super.onResume()
        initFields()
        initToolbar()
        initRecyclerView()
    }

    private fun initFields() {
        setHasOptionsMenu(true)
        mBottomSheetBehavior =
BottomSheetBehavior.from(bottom_sheet_choice)
        mBottomSheetBehavior.state =
BottomSheetBehavior.STATE_HIDDEN
        mAppVoiceRecorder = AppVoiceRecorder()
        mSwipeRefreshLayout = chats_swipe_refresh
        layoutManager = LinearLayoutManager(this.context)

        chats_input_message.addTextChangedListener(AppTextWatcher {
            val input = chats_input_message.text.toString()
            if(input.isEmpty() || input == "Идет запись"){
                chats_btn_send.visibility = View.GONE
                chats_btn_attch.visibility = View.VISIBLE
                chats_btn_voice_message.visibility =
View.VISIBLE
            }
        })
    }
}
```

```

    }else {

chats_btn_send.visibility = View.VISIBLE
        chats_btn_attch.visibility = View.GONE
        chats_btn_voice_message.visibility = View.GONE
    }

    })

    chats_btn_attch.setOnClickListener {attach()}
    CoroutineScope(Dispatchers.IO).launch {
        chats_btn_voice_message.setOnTouchListener { v,
event ->

            if (checkPermission(RECORD_AUDIO)) {
                if (event.action == MotionEvent.ACTION_DOWN)
{
                    chats_input_message.setText("Идет
запись")

                    val messageKey =
getMessageKey(contact.id)

mAppVoiceRecorder.startRecording(messageKey)

                } else if (event.action ==
MotionEvent.ACTION_UP) {
                    chats_input_message.setText("")
                    mAppVoiceRecorder.stopRecording{ file ,
messageKey ->

uploadFileToStorage(Uri.fromFile(file) , messageKey , contact.id
, TYPE_MESSAGE_VOICE)

                    mSmoothScrollToPosition = true

                }

            }

        }

    }

    true

}

}

private fun attach() {
    mBottomSheetBehavior.state =
BottomSheetBehavior.STATE_EXPANDED
    attach_file_btn.setOnClickListener {
        attachFile()
    }
    attach_image_btn.setOnClickListener {
        attachImage()
    }
}

```

```

private fun attachFile() {

    val intent = Intent(Intent.ACTION_GET_CONTENT)
    intent.type = "*/*"
    startActivityForResult(intent ,
    CHOOSE_FILE_REQUEST_CODE)

}

private fun attachImage() {
    CropImage.activity()
        .setGuidelines(CropImageView.Guidelines.ON)
        .setMultiTouchEnabled(true)
        .setAspectRatio(1, 1)
        .setRequestedSize(250, 250,
CropImageView.RequestSizeOptions.RESIZE_INSIDE)
        .setMinCropWindowSize(0, 0)
        .start(ACTIVITY, this)
}

private fun initRecyclerView() {
    mRecyclerView = chats_recycler_view
    mAdapter = IndividualChatAdapter()
    mRefToMessages =
REF_DATABASE_ROOT.child(NODE_MESSAGES).child(UID).child(contact.
id)

    mRecyclerView.adapter = mAdapter
    mRecyclerView.setHasFixedSize(true)
    mRecyclerView.isNestedScrollingEnabled = false

    mRecyclerView.layoutManager = layoutManager

    mMessagesListener = AppChildEventListener {

        val message = it.getDefaultModel()

        if (mSmoothScrollToPosition) {

mAdapter.addItemToBottom(ViewFactory.getView(message)) {

mRecyclerView.smoothScrollToPosition(mAdapter.itemCount)
        }
        } else {

mAdapter.addItemToTop(ViewFactory.getView(message)) {
            mSwipeRefreshLayout.isRefreshing = false
        }
        }

    }
}

```

```

mRefToMessages.limitToLast(mCountMessagesToLoad).addChildEventListener(mMessagesListener)

        mRecyclerView.addOnScrollListener(object :
RecyclerView.OnScrollListener() {
            override fun onScrolled(recyclerView: RecyclerView,
dx: Int, dy: Int) {
                super.onScrolled(recyclerView, dx, dy)
                if (mIsUserScrolling && dy < 0 &&
mLayoutManager.findFirstVisibleItemPosition() <= 3) {
                    updateData()
                }
            }

            override fun onScrollStateChanged(recyclerView:
RecyclerView, newState: Int) {
                super.onScrollStateChanged(recyclerView,
newState)

                if (newState ==
AbsListView.OnScrollListener.SCROLL_STATE_TOUCH_SCROLL) {
                    mIsUserScrolling = true
                }
            }
        })

        mSwipeRefreshLayout.setOnRefreshListener {
            updateData()
        }

    }

    private fun updateData() {
        mSmoothScrollToPosition = false
        mIsUserScrolling = false
        mCountMessagesToLoad += 10
        mRefToMessages.removeEventListener(mMessagesListener)

mRefToMessages.limitToLast(mCountMessagesToLoad).addChildEventListener(mMessagesListener)

    }

    private fun initToolbar() {
        mToolBarInfo = ACTIVITY.mToolbar.toolbar_chats
        mToolBarInfo.visibility = View.VISIBLE
        mListenerInfoToolbar = AppValueEventListener {
            mReceivingUser = it.getUserModel()
            initInfoToolbar()
        }
    }

```

```

        mRefToUser =
REF_DATABASE_ROOT.child(NODE_USERS).child(contact.id)
        mRefToUser.addValueEventListener(mListenerInfoToolbar)
        chats_btn_send.setOnClickListener {
            mSmoothScrollToPosition = true
            val message = chats_input_message.text.toString()
            if (message.isEmpty()) {
                showToast("Введите сообщение")
            } else {
                sendMessage(message, contact.id, TYPE_TEXT) {
                    saveToMainList(contact.id , TYPE_CHAT)
                    chats_input_message.setText("")
                }
            }
        }
    }
}

private fun initInfoToolbar() {
    if (mReceivingUser.fullname.isEmpty()) {
        mToolBarInfo.toolbar_chats_fullname.text =
contact.fullname
    } else mToolBarInfo.toolbar_chats_fullname.text =
mReceivingUser.fullname

mToolBarInfo.toolbar_chats_image.downloadAndSetImage(mReceivingU
ser.photoUrl)

    mToolBarInfo.toolbar_chats_status.text =
mReceivingUser.state
}

override fun onPause() {
    super.onPause()
    mToolBarInfo.visibility = View.GONE
    mRefToUser.removeEventListener(mListenerInfoToolbar)
    mRefToMessages.removeEventListener(mMessagesListener)
}

override fun onActivityResult(requestCode: Int, resultCode:
Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if(data != null) {
        when (requestCode) {
            CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE -> {
                val uri =
CropImage.getActivityResult(data).uri
                val messageKey = getMessageKey(contact.id)
                uploadFileToStorage(uri, messageKey,

```

```

contact.id, TYPE_MESSAGE_IMAGE)

mSmoothScrollToPosition = true
    }
    CHOOSE_FILE_REQUEST_CODE -> {

        val uri = data.data
        val messageKey = getMessageKey(contact.id)
        val filename = getFileNameFromUri(uri!!)
        uploadFileToStorage(uri, messageKey,
contact.id, TYPE_MESSAGE_FILE , filename)
        mSmoothScrollToPosition = true
    }
    }

}

}

override fun onDestroyView() {
    super.onDestroyView()
    mAppVoiceRecorder.releaseRecorder()
    mAdapter.destroy()
}

override fun onCreateOptionsMenu(menu: Menu, inflater:
MenuInflater) {

activity?.menuInflater?.inflate(R.menu.individual_chat_action_me
nu, menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean
    {
        when (item.itemId) {
            R.id.menu_clear_chat -> clearChat(contact.id){
                showToast("Чат очищен")
                switchFragment(ChatsFragment())
            }
            R.id.menu_delete_chat -> deleteChat(contact.id){
                showToast("Чат удален")
                switchFragment(ChatsFragment())
            }
        }
        return true
    }

}

}

```

1.3.5 Реализация групповых чатов

За добавление пользователей в групповой чат отвечает класс *AddContactsFragment*. Его метод *initRecyclerView* отвечает за отображение пользователей в списке участников, которых можно добавить в чат, после нажатия на кнопку продолжения список выбранных участников передается во фрагмент *CreateGroupFragment*, добавление представлено на рисунке 8.

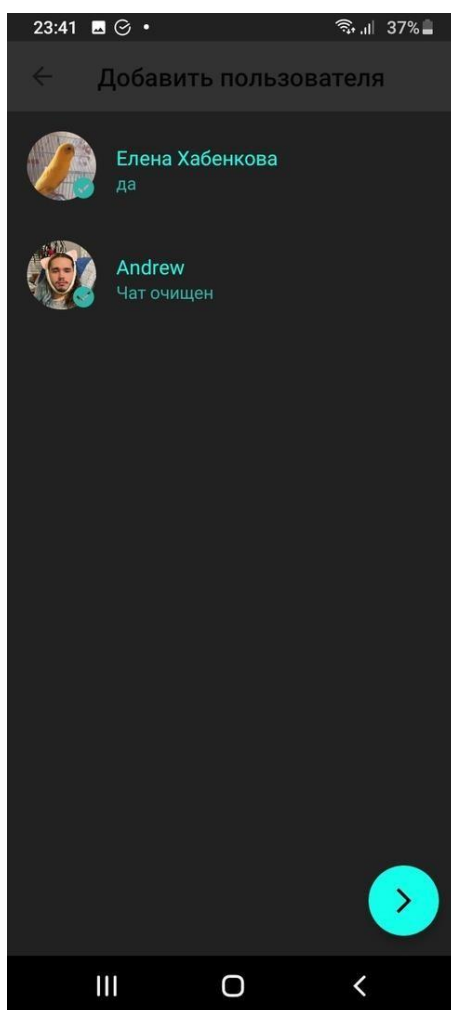


Рисунок 8 - Добавление пользователей в групповой чат

Код для добавления участников в групповой чат представлен ниже:

```
class AddContactsFragment :  
BaseFragment(R.layout.fragment_add_contacts) {  
    private lateinit var mRecyclerView: RecyclerView  
    private lateinit var mAdapter: AddContactsAdapter  
    private val mRefContactsList =  
REF_DATABASE_ROOT.child(NODE_PHONES_CONTACTS).child(UID)  
  
    private val mRefUsers = REF_DATABASE_ROOT.child(NODE_USERS)
```

```

private val mRefMessages =
REF_DATABASE_ROOT.child(NODE_MESSAGES).child(UID)
    private var mListItems = listOf<DefaultModel>()
    override fun onResume() {
        contactsList.clear()
        super.onResume()
        ACTIVITY.title = "Добавить пользователя"
        closeKeyboard()
        initRecyclerView()
        add_contacts_btn_next.setOnClickListener {
            if (contactsList.isEmpty()) {
                showToast("Добавьте участников")
            } else {

switchFragment(CreateGroupFragment(contactsList))
            }
        }

        private fun initRecyclerView() {
            mRecyclerView = add_contacts_recycler_view
            mAdapter = AddContactsAdapter()

mRefContactsList.addListenerForSingleValueEvent(AppValueEventListener { dataSnapshot ->
            mListItems = dataSnapshot.children.map {
it.getDefaultModel() }
            mListItems.forEach { model ->

                mRefUsers.child(model.id)

.addListenersForSingleValueEvent(AppValueEventListener {
dataSnapshot1 ->
                    val newModel =
dataSnapshot1.getDefaultModel()

mRefMessages.child(model.id).limitToLast(1)

.addListenersForSingleValueEvent(AppValueEventListener {
dataSnapshot2 ->
                        val tempList =
dataSnapshot2.children.map { it.getDefaultModel() }
                        if (tempList.isEmpty()) {
                            newModel.lastMessage = "Чат
очищен"
                        } else {
                            newModel.lastMessage =

tempList[0].messageText

```

```

        }

        if (newModel.fullname.isEmpty())
        {
            newModel.fullname =
newModel.phone
        }

mAdapter.updateItemsList(newModel)
        })
    }
})

mRecyclerView.adapter = mAdapter
}

companion object {
    val contactsList = mutableListOf<DefaultModel>()
}

}

```

За создание группового чата отвечает класс *CreateGroupFragment*, при создании чата пользователь может изменить название чата и его изображение, после создания группового чата он появляется у всех пользователей на главном экране.

Метод *initRecyclerView* отвечает за отображение контактов при создании группового чата, метод *addPhoto* за изменение фото чата, само название чата изменяется при помощи слушателя на элементе *EditText*, при этом имеется проверка на то, чтобы название чата не было пустым. Групповой чат имеет такой же функционал, как и одиночный, создание группового чата представлено на рисунке 9.

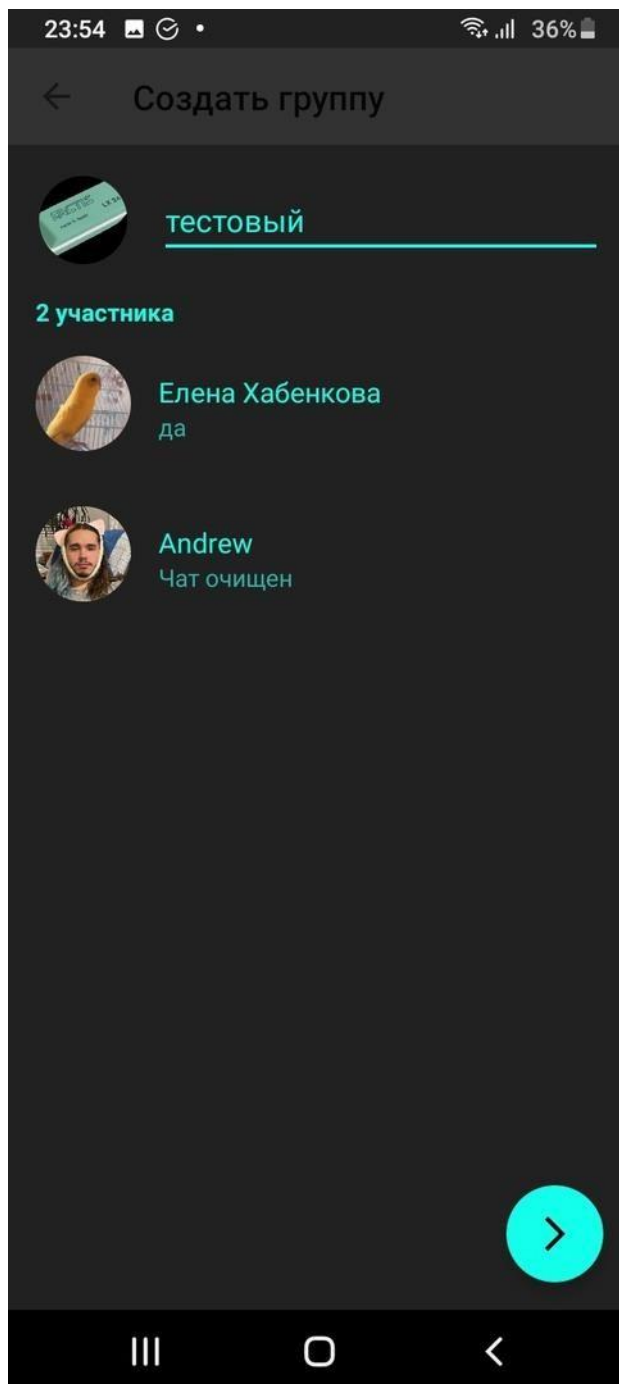


Рисунок 9 - Создание группового чата

Код для создания группового чата представлен ниже:

```
class CreateGroupFragment(var contactsList: List<DefaultModel>)
:
    BaseFragment(R.layout.fragment_create_group) {

    private lateinit var mRecyclerView: RecyclerView
    private lateinit var mAdapterter: AddContactsAdapter
    private var mUri = Uri.EMPTY
```

```

        override fun onResume() {

super.onResume()
        ACTIVITY.title = "Создать группу"
        closeKeyboard()
        initRecyclerView()
        create_group_photo.setOnClickListener {
            addPhoto()
        }
        create_group_btn_complete.setOnClickListener {
            val groupName =
create_group_input_name.text.toString()
            if(groupName.isEmpty()) {
                showToast("Введите имя группы")
            } else {

                createGroupInDatabase(groupName , mUri ,
contactsList) {
                    switchFragment (ChatsFragment ())
                }
            }
        }
        create_group_input_name.requestFocus()
        create_group_counts.text = getPlurals (contactsList.size)
    }

private fun addPhoto() {
    CropImage.activity()
        .setGuidelines (CropImageView.Guidelines.ON)
        .setMultiTouchEnabled (true)
        .setAspectRatio (1, 1)
        .setRequestedSize (250, 250,
CropImageView.RequestSizeOptions.RESIZE_INSIDE)
        .setMinCropWindowSize (0, 0)
        .setCropShape (CropImageView.CropShape.OVAL)
        .start (ACTIVITY, this)
    }

private fun initRecyclerView() {
    mRecyclerView = create_group_recycler_view
    mAdapter = AddContactsAdapter()
    mRecyclerView.adapter = mAdapter
    contactsList.forEach { mAdapter.updateItemsList(it) }

}

override fun onActivityResult (requestCode: Int, resultCode:
Int, data: Intent?) {

```

```

super.onActivityResult(requestCode, resultCode, data)
    if (requestCode ==

CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE && resultCode ==
Activity.RESULT_OK && data != null) {
    mUri = CropImage.getActivityResult(data).uri
    create_group_photo.setImageURI(mUri)

    }
}
}

```

1.3.6 Реализация главного окна приложения

Главное окно приложения представляет из себя список одиночных чатов и групп, а также выдвижное меню, из которого можно попасть в любой фрагмент приложения, который были рассмотрены выше. За инициализацию выдвижного меню отвечает класс *AppDrawer*, метод *createHeader* создает верхнюю часть выдвижного меню, метод *createDrawer* создает элементы выдвижного меню и устанавливает на них слушатель нажатий, выдвижное меню представлено на рисунке 10.

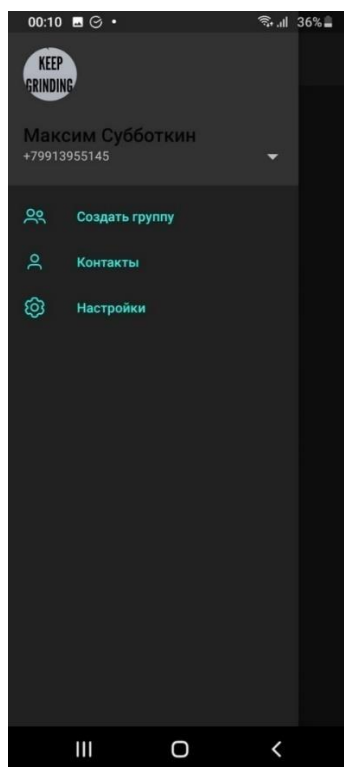


Рисунок 10 - Выдвижное меню

Код создания выдвижного меню представлен ниже:

```
class AppDrawer {

    private lateinit var mDrawer: Drawer
    private lateinit var mHeader: AccountHeader
    private lateinit var mDrawerLayout: DrawerLayout
    private lateinit var mCurrentProfile: ProfileDrawerItem

    fun create() {
        initImageLoader()
        createHeader()
        createDrawer()
        mDrawerLayout = mDrawer.drawerLayout
    }

    fun disableDrawer() {

        mDrawer.actionBarDrawerToggle?.isDrawerIndicatorEnabled
= false

        ACTIVITY.supportActionBar?.setDisplayHomeAsUpEnabled(true)

        mDrawerLayout.setDrawerLockMode(DrawerLayout.LOCK_MODE_LOCKED_CLOSED)
        ACTIVITY.mToolbar.setNavigationOnClickListener {
            ACTIVITY.supportFragmentManager.popBackStack()
        }

    }

    fun enableDrawer() {

        ACTIVITY.supportActionBar?.setDisplayHomeAsUpEnabled(false)
        mDrawer.actionBarDrawerToggle?.isDrawerIndicatorEnabled
= true

        mDrawerLayout.setDrawerLockMode(DrawerLayout.LOCK_MODE_UNLOCKED)
        ACTIVITY.mToolbar.setNavigationOnClickListener {
            mDrawer.openDrawer()
        }

    }

    private fun createDrawer() {
        mDrawer = DrawerBuilder()
            .withActivity(ACTIVITY)
            .withToolbar(ACTIVITY.mToolbar)
            .withActionBarDrawerToggle(true)
            .withSelectedItem(-1)

            .withAccountHeader(mHeader)
```

```

        .addDrawerItems(

PrimaryDrawerItem().withIdentifier(100)
                    .withIconTintingEnabled(true)
                    .withName("Создать группу")
                    .withSelectable(false)
                    .withIcon(R.drawable.ic_menu_create_groups),
PrimaryDrawerItem().withIdentifier(103)
                    .withIconTintingEnabled(true)
                    .withName("Контакты")
                    .withSelectable(false)
                    .withIcon(R.drawable.ic_menu_contacts),
PrimaryDrawerItem().withIdentifier(106)
                    .withIconTintingEnabled(true)
                    .withName("Настройки")
                    .withSelectable(false)
                    .withIcon(R.drawable.ic_menu_settings)
                ).withOnDrawerItemClickListener(object :
Drawer.OnDrawerItemClickListener {
                override fun onItemClick(
                    view: View?,
                    position: Int,
                    drawerItem: IDrawerItem<*>
                ): Boolean {
                    clickOnItem(position)
                    return false
                }
            }).build()

    }

    private fun clickOnItem(position: Int) {
        when (position) {
            1-> switchFragment(AddContactsFragment())
            3 -> switchFragment(SettingsFragment())
            2 -> switchFragment(ContactsFragment())
        }
    }

    private fun createHeader() {
        mCurrentProfile = ProfileDrawerItem()
            .withName(USER.fullname)
            .withEmail(USER.phone)
            .withIcon(USER.photoUrl)
            .withIdentifier(200)
        mHeader = AccountHeaderBuilder()
            .withActivity(ACTIVITY)
            .withHeaderBackground(R.drawable.header)
            .addProfiles(

mCurrentProfile

```

```

        ).build()
    }

    fun updateHeader() {
        mCurrentProfile
            .withName (USER.fullname)
            .withEmail (USER.phone)
            .withIcon (USER.photoUrl)

        mHeader.updateProfile (mCurrentProfile)
    }

    private fun initImageLoader() {
        DrawerImageLoader.init(object :
AbstractDrawerImageLoader() {
            override fun set(imageView: ImageView, uri: Uri,
placeholder: Drawable) {
                imageView.downloadAndSetImage(uri.toString())
            }
        })
    }
}

```

За отображение чатов на главном окне приложения отвечает класс *ChatsFragment* и вспомогательный для него класс *ChatsAdapter*, метод *showGroup* определяет, как будет отображаться групповой чат, а метод *showChat* определяет, как будет отображаться одиночный чат. При нажатии на чат пользователь перейдет в него, главное окно приложения представлено на рисунке 11.

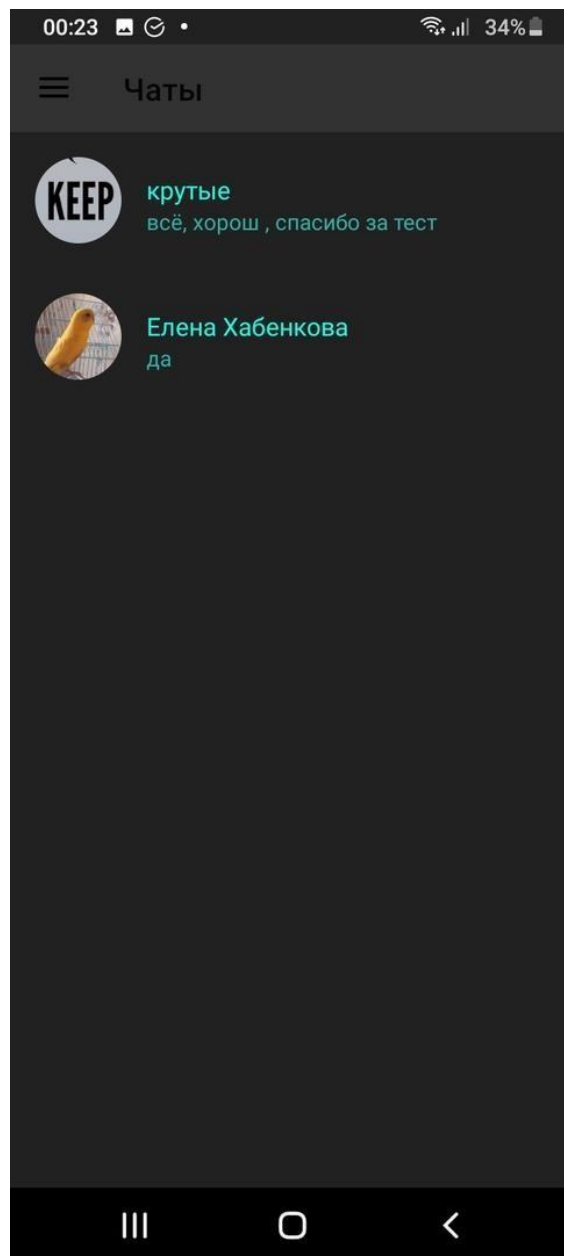


Рисунок 11 - Главное окно приложения

Код главного окна приложения представлен ниже:

```
class ChatsFragment : Fragment(R.layout.fragment_chats) {

    private lateinit var mRecyclerView : RecyclerView
    private lateinit var mAdapterter : ChatsAdapter
    private val mRefMainList =
    REF_DATABASE_ROOT.child(NODE_MAIN_LIST).child(UID)
    private val mRefUsers = REF_DATABASE_ROOT.child(NODE_USERS)
    private val mRefMessages =
    REF_DATABASE_ROOT.child(NODE_MESSAGES).child(UID)
    private var mListItems = listOf<DefaultModel>()
```

```

        override fun onResume() {
            super.onResume()
            ACTIVITY.title = "Чаты"
            ACTIVITY.mAppDrawer.enableDrawer()
            closeKeyboard()
            initRecyclerView()
        }

        private fun initRecyclerView() {
            mRecyclerView = main_list_recycler_view
            mAdapter = ChatsAdapter()

mRefMainList.addListenerForSingleValueEvent(AppValueEventListener { dataSnapshot ->
            mListItems = dataSnapshot.children.map {
it.getDefaultModel() }
            mListItems.forEach { model->

                when(model.messageType) {
                    TYPE_CHAT -> showChat(model)
                    TYPE_GROUP -> showGroup(model)
                }

            }

        })

        mRecyclerView.adapter = mAdapter

    }

    private fun showGroup(model: DefaultModel) {
REF_DATABASE_ROOT.child(NODE_GROUPS).child(model.id).addListener
ForSingleValueEvent(AppValueEventListener { dataSnapshot1->
        val newModel = dataSnapshot1.getDefaultModel()

REF_DATABASE_ROOT.child(NODE_GROUPS).child(model.id).child(NODE_
MESSAGES)

        .limitToLast(1).addListenerForSingleValueEvent(AppValueEventList
ener { dataSnapshot2->
            val tempList = dataSnapshot2.children.map {
it.getDefaultModel() }
            if(tempList.isEmpty()) {
                newModel.lastMessage = "Чат очищен"
            } else {
                newModel.lastMessage =
tempList[0].messageText
            }
            newModel.messageType = TYPE_GROUP
            mAdapter.updateItemsList(newModel)
    }

```

```

        })
    })
}

private fun showChat(model: DefaultModel) {

mRefUsers.child(model.id).addListenerForSingleValueEvent(AppValue
eEventListener { dataSnapshot1->
    val newModel = dataSnapshot1.getDefaultModel()

mRefMessages.child(model.id).limitToLast(1).addListenerForSingle
ValueEvent(AppValueEventListener { dataSnapshot2->
    val tempList = dataSnapshot2.children.map {
it.getDefaultModel() }
    if(tempList.isEmpty()) {
        newModel.lastMessage = "Чат очищен"
    } else {
        newModel.lastMessage =
tempList[0].messageText
    }

    if(newModel.fullname.isEmpty()) {
        newModel.fullname = newModel.phone
    }
    newModel.messageType = TYPE_CHAT
    mAdapter.updateItemsList(newModel)
    })
})
}

}

```

1.4 Руководство пользователя

При запуске приложения пользователю нужно будет ввести номер телефона и после этого код подтверждения, после этого он попадет на главный экран приложения.

Если пользователь уже вошел в приложение, то при его запуске он будет сразу попадет на главный экран в свой аккаунт, рисунки 12-13.

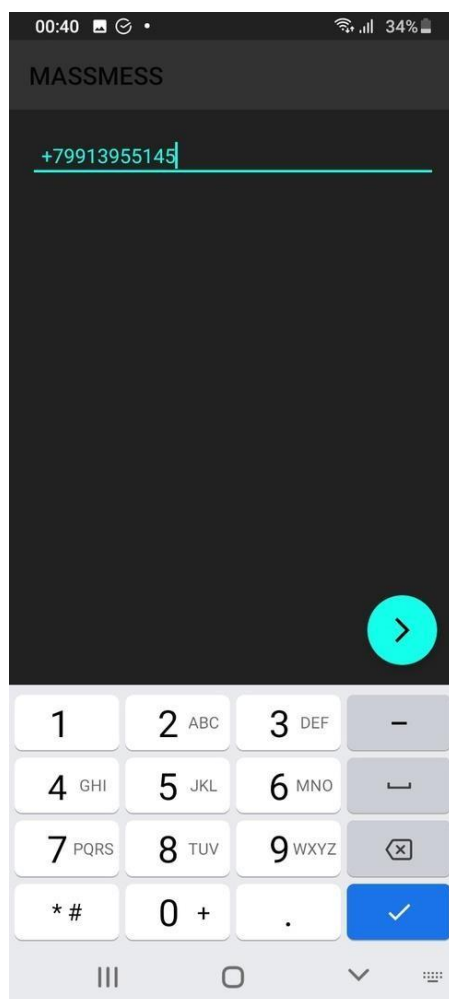


Рисунок 12 - Окно ввода номера телефона

						Лист
						45
Изм.	Лист	№ докум	Подпись	Дата		

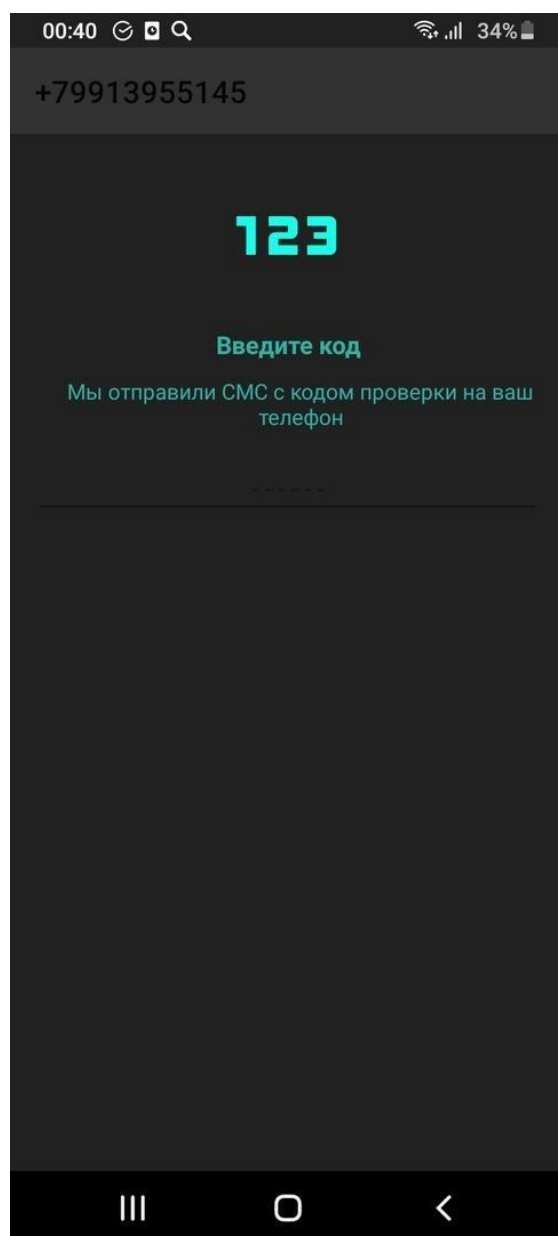


Рисунок 13 - Окно ввода проверочного кода

После входа в приложение пользователь может настроить свой аккаунт, написать другому пользователю через вкладку контактов, создать групповой чат, рисунки 14-16.

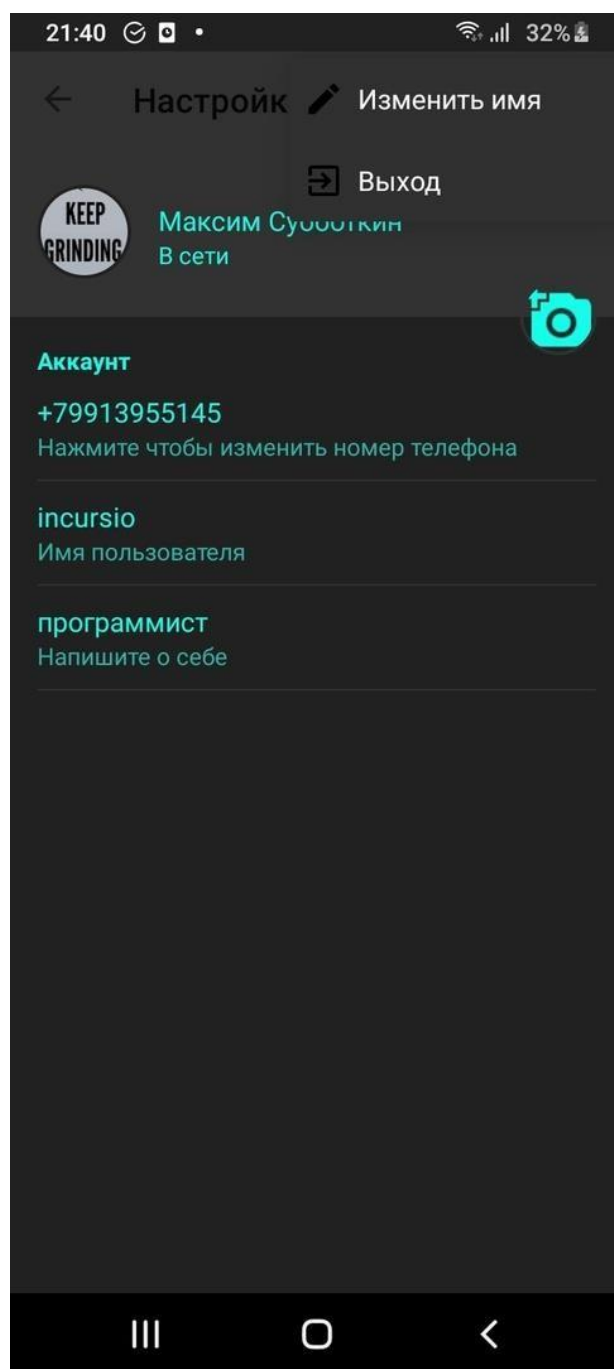


Рисунок 14 - Окно настроек

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		47

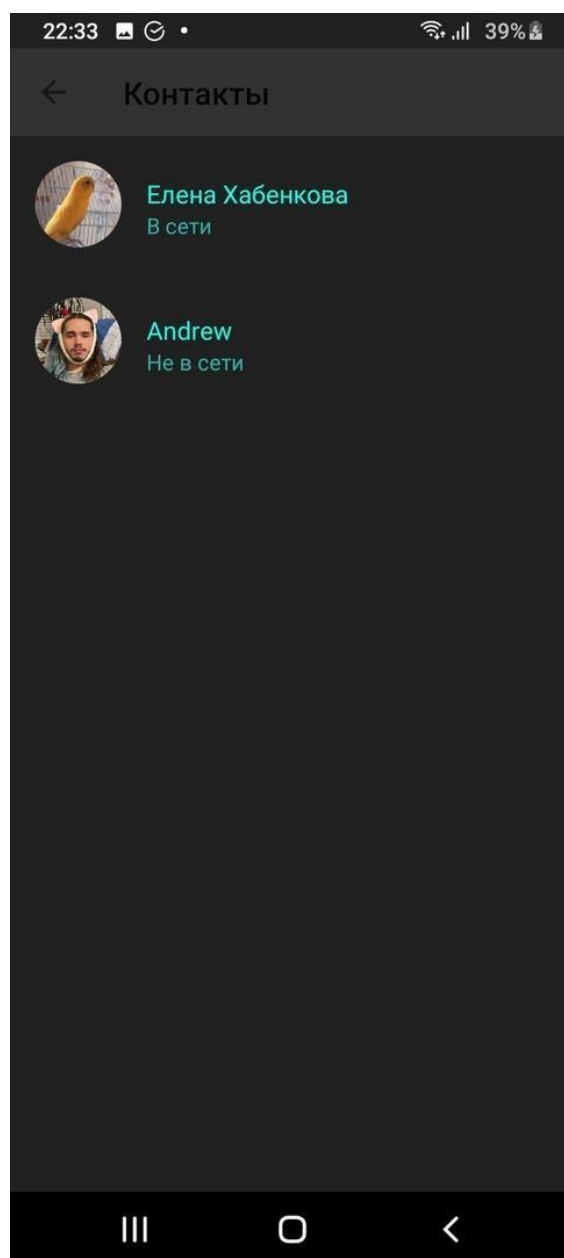


Рисунок 15 - Окно контактов

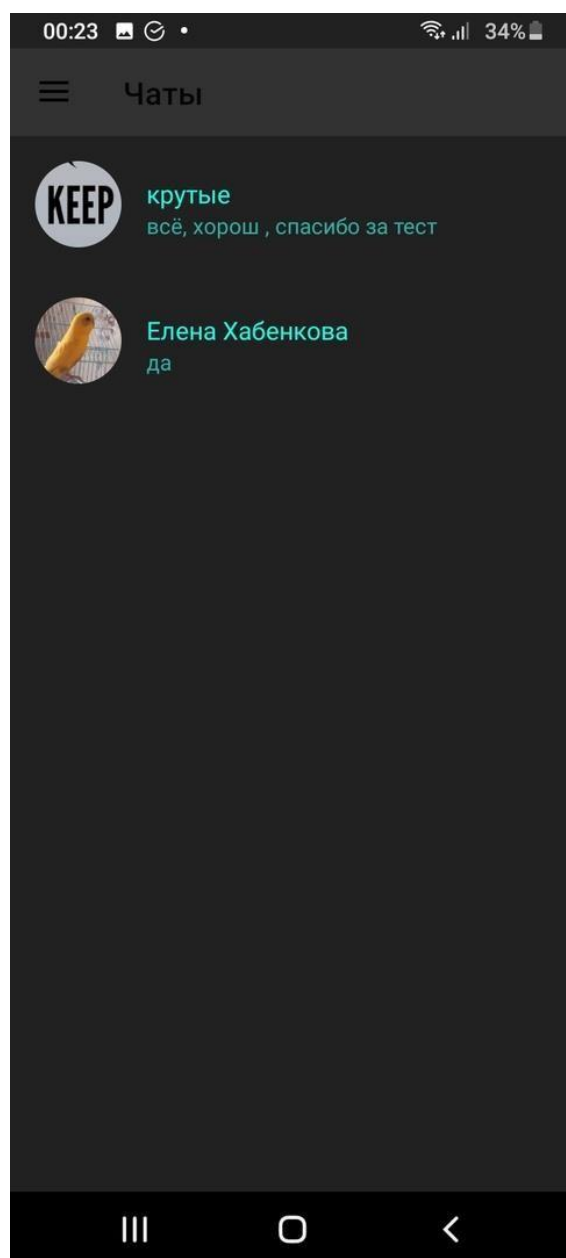


Рисунок 16 - Окно чатов

2 Безопасность жизнедеятельности

На всех этапах жизненного цикла проекта, будь то его разработка, внедрение или эксплуатация, важно обеспечить безопасность человека. Чтобы создать безопасные и комфортные условия работы, необходимо правильно оценить опасные и вредные факторы, связанные с производственной средой, физической и умственной нагрузкой, а также нервно-эмоциональным напряжением. Различные причины могут приводить к схожим изменениям в организме человека, поэтому важно принимать их во внимание и предпринимать соответствующие меры для обеспечения безопасности.

2.1 Потенциально опасные факторы при работе с ПЭВМ

Производственные факторы, которые представляют опасность и вредность, могут быть разделены на следующие группы по их природе: физические, химические, психофизиологические и биологические. При работе с компьютером (ПЭВМ) на программиста(пользователя) могут негативно влиять различные физические факторы, такие как изменение температуры воздуха (повышенная или пониженная), наличие избыточной пыли или газов в воздухе, избыточная или недостаточная влажность, высокий уровень шума, наличие ионизирующего излучения, риск поражения электрическим током, статическое электричество, плохое качество экрана дисплея и недостаточное освещение рабочего места. К химическим опасным факторам, которые могут постоянно воздействовать на программиста, относятся активные частицы, возникающие при ионизации воздуха в процессе работы на компьютере. В данном помещении отсутствуют биологические вредные производственные факторы. К психологическим вредным факторам, которые могут оказывать воздействие на программиста в течение рабочей смены, можно отнести нервно-эмоциональные перегрузки, умственное напряжение и перенапряжение зрительного анализатора.

2.2 Классификация помещения

Кабинет программиста, относящийся к категории В1-В4 (пожароопасность) I категории (выполняются легкие физические работы), должен соответствовать следующим требованиям: температура воздуха должна быть 22 °С (допустимый диапазон - 20-24 °С), оптимальная относительная влажность составляет 40-60% (допустимый предел - не более 75%), а скорость движения воздуха не должна превышать 0.1 м/с. Для обеспечения и автоматического поддержания оптимальных значений температуры, влажности, чистоты и скорости движения воздуха на рабочем месте, независимо от внешних условий, используется система водяного отопления в холодное время года и кондиционирование воздуха в теплое время года. Кондиционер, как часть вентиляционной системы, автоматически регулирует параметры воздушной среды в помещении. Владелец предприятия должен обеспечить следующие меры: поддержание работоспособности электрооборудования и его эксплуатацию в соответствии с требованиями нормативной технической документации, своевременное и качественное проведение профилактических работ, ремонта, модернизации и реконструкции энергетического оборудования, надежность работы электроустановок и безопасность их обслуживания, предотвращение использования технологий и методов работы, которые могут негативно влиять на окружающую среду, учет и анализ нарушений в работе электроустановок, несчастных случаев и принятие мер для устранения их причин, разработку должностных инструкций и производственных инструкций для электротехнического персонала, а также выполнение предписаний органов Госэнергонадзора.

2.3 Санитарно-гигиеническая характеристика

Ключевыми показателями, которые характеризуют микроклимат, являются следующие: температура воздуха, скорость движения воздуха и

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		51

относительная влажность. Работа программиста категории IA предполагает соблюдение оптимальных требований и норм, включающих следующие значения: оптимальная температура воздуха составляет 22-24 °С, относительная влажность - 40-60%, а скорость движения воздуха должна быть 0,1 м/с. При работе с компьютером программист проводит продолжительное время в статическом положении и часто напрягает глаза. Так как работа за компьютером обычно происходит при искусственном освещении, необходимо обеспечить комфортные условия для сотрудников, чтобы уменьшить нагрузку на глаза. Отсутствие или недостаток освещения в помещении может негативно сказаться на здоровье глаз, снизить работоспособность, ухудшить кровообращение, вызвать сильное напряжение и доставить человеку дискомфорт. Это также может стать причиной сильных головных болей. После долгого рабочего дня возникают чувство усталости и слезоточивость глаз. При работе за компьютером следует соблюдать следующие требования: использовать лампы для освещения рабочего места; сочетать локальное и общее освещение; не позволять прямому и яркому попаданию солнечных лучей на монитор, так как это может повлиять на отображаемую картинку и вызвать искажения; не работать за компьютером в темных помещениях; помещения с компьютерами не должны находиться рядом с помещениями, где уровень шума и вибраций превышает нормальные значения; уровень шума на рабочем месте не должен превышать 50 дБА.

2.4 Расчет освещенности рабочего места

Правильный выбор и расчет освещенности рабочего места обеспечивает создание нормальных условий для зрения обслуживающего персонала, способствует повышению производительности труда. Освещение, соответствующее санитарным нормам, является главнейшим условием гигиены труда и культуры производства. При хорошем освещении устраняется напряжение зрения, ускоряется темп работы.

Для рационального освещения необходимо выполнение следующих условий:

- постоянная освещенность рабочих поверхностей;
- достаточная и равномерно распределенная яркость освещаемых рабочих поверхностей;
- отсутствие резких контрастов между яркостью рабочей поверхностью и окружающего пространства.

Для рационального освещения производственных помещений и рабочих мест большое значение имеет выбор цвета для окраски потолка, стен и производственного оборудования.

Рассчитаем люминесцентное освещение кабинета, предназначенного для выполнения работ с размером объекта различения от 0,3 мин до 1 мин. Размеры помещений: А(длина)=4 м, В(ширина)=3 м, Н(высота)=3 м. Площадь S помещения определяем по формуле:

$$S=A*B.$$

Подставив значения получим:

$$S=4*3=12 \text{ м}^2.$$

Освещение проектируется при помощи светильников ШОД минимальной освещенностью $E_{\text{мин}}=200$ лк, средней удельной мощностью 17-23 Вт/м². Высота подвеса над рабочей поверхностью Нр=2 м.

Освещение выполняется лампами ЛБ-2*40, световой поток ламп $\Phi_{\text{л}}=2480$ лм, длина 1.2 м, коэффициент запаса равен $k=1.5$. Определим показатель помещения F по формуле:

$$F=A*B/H_p(A+B).$$

Подставив значения получим:

$$F=4*3/2*(4+3)=0.86.$$

Затем, для $F=0,86$, коэффициентов отражения потолка $R_{\text{п}}=0,7$, стен $R_{\text{с}}=0,5$ и расчетной плоскости $R_{\text{р}}=0,3$ находим коэффициент использования светового потока $h=0,37$. Потребное число светильников определяется по формуле:

$$N = E_{\text{мин}} \cdot S \cdot K / F_{\text{л}} \cdot h \cdot Z \cdot n,$$

где $n=2$ общее число ламп в светильнике.

Подставив значения получим:

$$N = 200 \cdot 12 \cdot 1,5 / 2480 \cdot 0,37 \cdot 0,9 \cdot 2 = 2 \text{ штуки.}$$

Общее количество ламп равно $n=2 \cdot 2=4$.

Таким образом, выполнение перечисленных выше требований к организации рабочего места обеспечит комфортные условия умственного труда.

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		54

Заключение

В результате проделанной работы было успешно разработано мобильное приложение, обладающее удобным и интуитивно понятным пользовательским интерфейсом. Данный проект представляет собой корпоративный мессенджер, который включает все необходимые функции и соответствует требованиям. Разработка данного приложения позволила значительно улучшить навыки создания клиентских приложений для платформы Android, а также расширить знания о работе с базами данных, API и интерфейсами.

Полученные навыки в процессе разработки этого приложения безусловно окажутся полезными при последующем трудоустройстве и развитии собственных компетенций.

Список использованных источников

1. Дейтел П., Дейтел Х., Уолд А. Android для разработчика. 3-е изд. – СПб.: Питер, 2017. – С. 37-374.
2. Firebase – RealtimeDatabase[Электронный ресурс] URL:
<https://firebase.google.com/docs/database>
3. Э. Гамма Р. Хелм Р. Джонсон Дж. Влссидес Приемы объекто-ориентированного проектирования. Паттеры проектирования. – СПб: Питер, 2001г. – 368 с
4. Мартин Р. Чистый Код: создание, анализ и рефакторинг. Пер. с англ.– М.: Издательский дом “Питер”, 2018. — С. 50-130

Приложение А

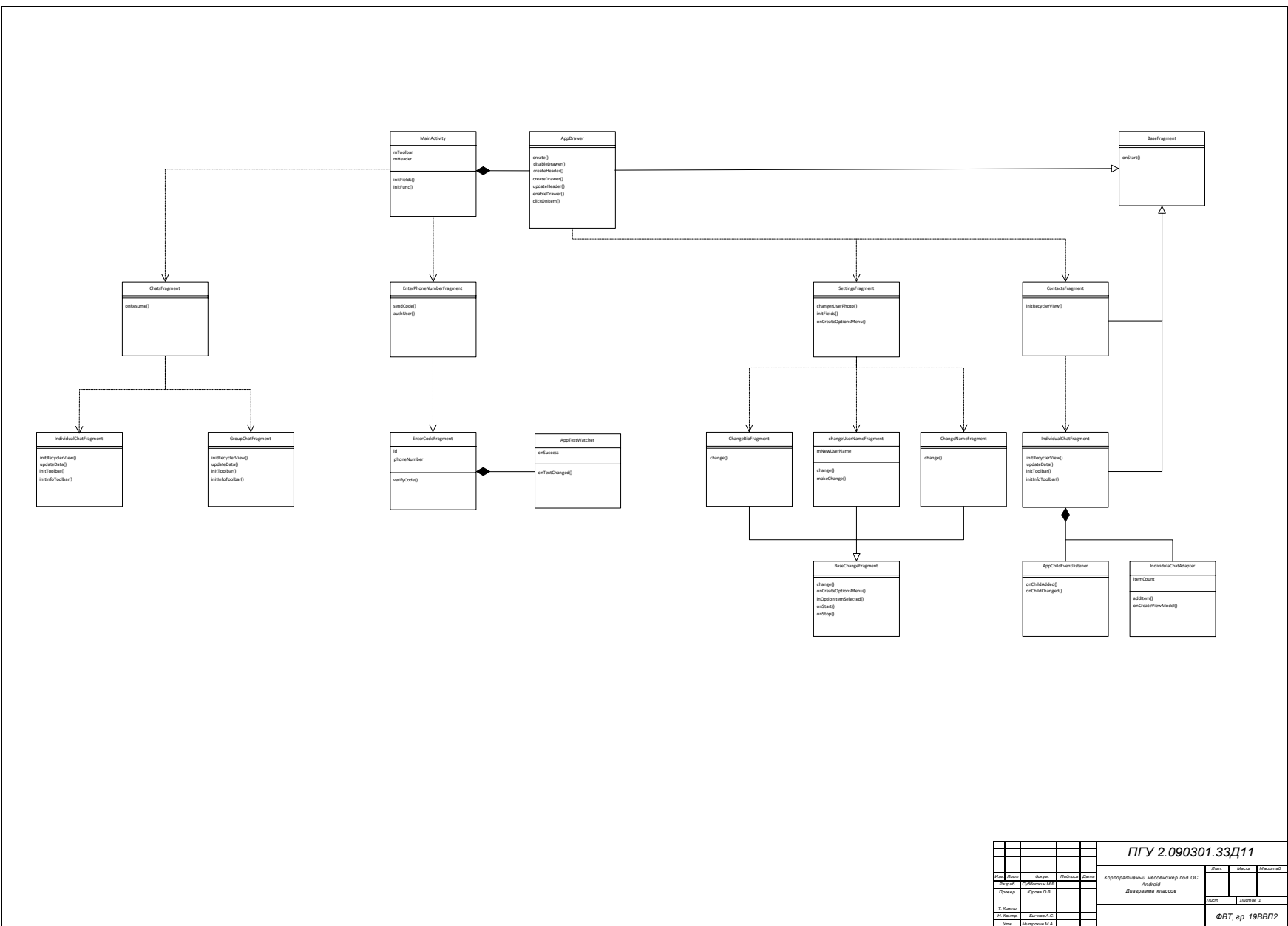
(Обязательное)

Корпоративный мессенджер под ОС Android

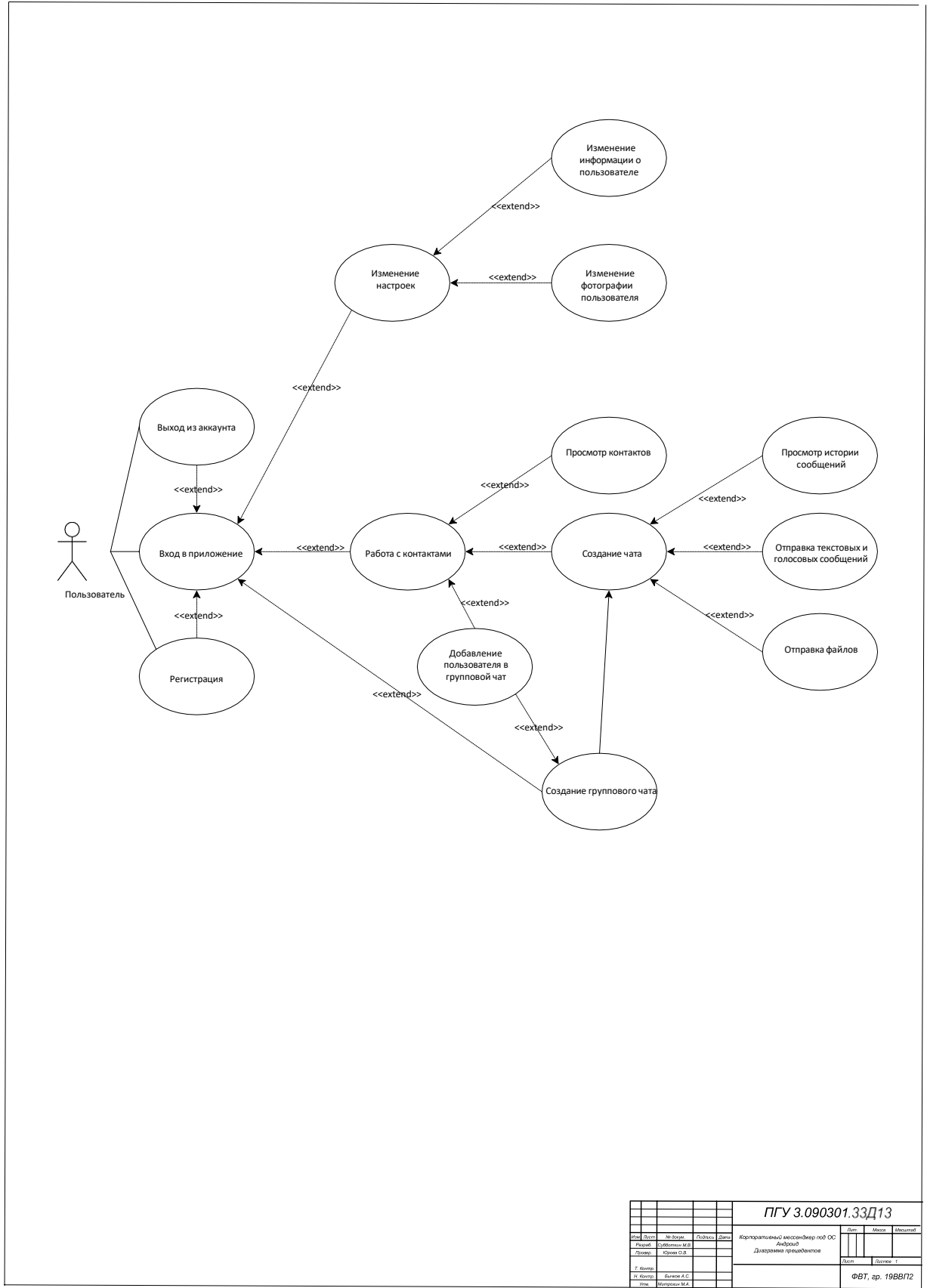
UML-диаграммы

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		57

Приложение А.1 – Диаграмма классов



Приложение А.2 – Диаграмма прецедентов



```

graph TD
    subgraph Client [Клиент]
        App[Приложение]
    end
    subgraph Internet [Интернет]
    end
    subgraph Server [Сервер]
        subgraph Firebase
            FirebaseAuth[FirebaseAuth]
        end
        DB[(База данных Firebase)]
        FirebaseAuth <--> DB
    end
    Client <--> Internet
    Internet <--> Server
  
```

ПГУ 4.090301.33Д19

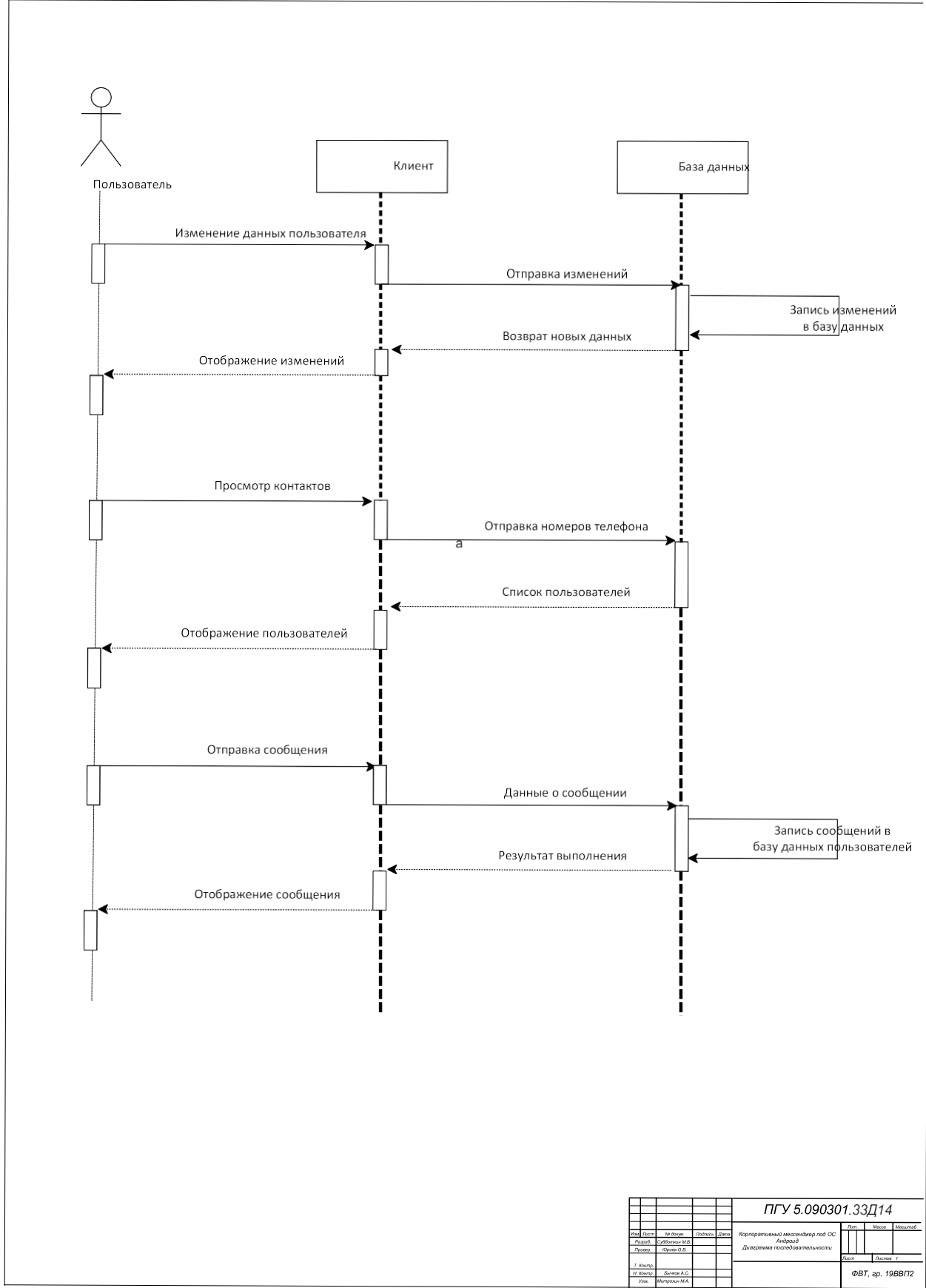
Имя	Фамилия	Отчество	Подпись	Место
Руднев	Степанович	Игорь		
Гусев	Юрий	Олег		
Г. Ковалев	И. Ковалев	Виктор		
И. Ковалев	И. Ковалев	Виктор		

Корпоративный мессенджер под ОС Android
Доверенное развёртывание

ФВТ, стр. 198ВВ/П

[illegible]

Приложение А.4 – Диаграмма последовательности



Приложение Б

(Обязательное)

Корпоративный мессенджер под ОС Android

Листинг программы

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		62

Файл MainActivity.kt

```
package com.example.massmess.allPackage

import android.content.pm.PackageManager
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.core.content.ContextCompat
import com.example.massmess.allPackage.database.AUTH
import com.example.massmess.allPackage.database.initFirebase
import com.example.massmess.allPackage.database.initUserModel
import
com.example.massmess.allPackage.ui.fragments.main_list.ChatsFrag
ment
import
com.example.massmess.allPackage.ui.fragments.register.EnterPhone
NumberFragment
import com.example.massmess.allPackage.ui.objects.AppDrawer
import com.example.massmess.allPackage.utilits.*
import com.example.massmess.databinding.ActivityMainBinding
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class MainActivity : AppCompatActivity() {

    private lateinit var mBinding : ActivityMainBinding
    lateinit var mAppDrawer : AppDrawer
    lateinit var mToolbar: Toolbar

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        mBinding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(mBinding.root)
        ACTIVITY = this
        initFirebase()
        initUserModel {
            CoroutineScope(Dispatchers.IO).launch {
                initContacts()
            }
            initFields()
            initFunc()
        }

    }

}
```

```

        private fun initFunc() {

setSupportActionBar(mToolbar)

        if ( AUTH.currentUser != null ) {
            mAppDrawer.create()
            switchFragment(ChatsFragment() ,false)
        }
        else {
            switchFragment(EnterPhoneNumberFragment() , false)
        }

    }

    private fun initFields() {

        mToolbar = mBinding.mainToolbar
        mAppDrawer = AppDrawer()

    }

    override fun onStart() {
        super.onStart()
        States.updateState(States.ONLINE)
    }

    override fun onStop() {
        super.onStop()
        States.updateState(States.OFFLINE)
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode,
permissions, grantResults)
        if (ContextCompat.checkSelfPermission(ACTIVITY ,
READ_CONTACTS) == PackageManager.PERMISSION_GRANTED){
            initContacts()
        }
    }

}

```

Файл States.kt

```
package com.example.massmess.allPackage.utilits
import com.example.massmess.allPackage.database.*
enum class States(val state : String) {

    ONLINE("В сети"),
    OFFLINE("Не в сети"),
    TYPING("Печатает");

    companion object {
        fun updateState (states : States) {

            if (AUTH.currentUser != null) {

                REF_DATABASE_ROOT.child(NODE_USERS).child(UID).child(CHILD_STATE
                )
                    .setValue(states.state)
                    .addOnSuccessListener { USER.state =
states.state }
            }
        }
    }
}
```

Файл permissions.kt

```
package com.example.massmess.allPackage.utilits
import android.Manifest
import android.content.pm.PackageManager
import android.os.Build
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat

const val READ_CONTACTS = Manifest.permission.READ_CONTACTS
const val RECORD_AUDIO = Manifest.permission.RECORD_AUDIO
const val WRITE_FILES =
Manifest.permission.WRITE_EXTERNAL_STORAGE
const val PERMISSION_REQUEST = 200

fun checkPermission(permission: String): Boolean {
    return if (Build.VERSION.SDK_INT >= 23 &&
ContextCompat.checkSelfPermission(
    ACTIVITY,
    permission
    ) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(ACTIVITY,
        arrayOf(permission), PERMISSION_REQUEST)
        false
    }
}
```

```

    } else true
}

```

Файл funs.kt

```

package com.example.massmess.allPackage.utilits

import android.content.Context
import android.content.Intent
import android.net.Uri
import android.provider.ContactsContract
import android.provider.OpenableColumns
import android.view.inputmethod.InputMethodManager
import android.widget.ImageView
import android.widget.Toast
import androidx.fragment.app.Fragment
import com.example.massmess.R
import com.example.massmess.allPackage.MainActivity
import
com.example.massmess.allPackage.database.updatePhonesDatabase
import com.example.massmess.allPackage.models.DefaultModel
import com.squareup.picasso.Picasso
import java.text.SimpleDateFormat
import java.util.*

fun showToast(message: String) {
    Toast.makeText(ACTIVITY, message, Toast.LENGTH_SHORT).show()
}

fun restartActivity() {
    val intent = Intent(ACTIVITY, MainActivity::class.java)
    ACTIVITY.startActivity(intent)
    ACTIVITY.finish()
}

fun switchFragment(fragment: Fragment, addToStack: Boolean =
true) {
    if (addToStack) {
        ACTIVITY.supportFragmentManager.beginTransaction()
            .addToBackStack(null)
            .replace(R.id.data_container, fragment)
            .commit()
    } else {
        ACTIVITY.supportFragmentManager.beginTransaction()
            .replace(R.id.data_container, fragment)
            .commit()
    }
}

```

```

fun closeKeyboard () {
    val manager : InputMethodManager =
ACTIVITY.getSystemService(Context.INPUT_METHOD_SERVICE) as
InputMethodManager

manager.hideSoftInputFromWindow(ACTIVITY.window.decorView.window
Token, 0 )
}

fun ImageView.downloadAndSetImage(url : String) {
    Picasso.get()
        .load(url)
        .fit()
        .placeholder(R.drawable.ic_menu_contacts)
        .into(this)
}

fun initContacts() {
    if (checkPermission(READ_CONTACTS)) {
        var contactsArray = arrayListOf<DefaultModel>()
        val cursor = ACTIVITY.contentResolver.query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
            null,
            null,
            null,
            null
        )
        cursor?.let {
            while (it.moveToNext()) {
                val fullName =

it.getString(it.getColumnIndexOrThrow(ContactsContract.Contacts.
DISPLAY_NAME))

                val phone =

it.getString(it.getColumnIndexOrThrow(ContactsContract.CommonDat
aKinds.Phone.NUMBER))

                val newModel = DefaultModel()
                newModel.fullname = fullName
                newModel.phone = phone.replace(Regex("[\\s,-]"),
"")

                contactsArray.add(newModel)
            }
        }

        cursor?.close()
        updatePhonesDatabase(contactsArray)
    }
}

```

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		67

```

fun String.asTime(): String {
    val time = Date(this.toLong())
    val timeFormat = SimpleDateFormat("HH:mm" ,
Locale.getDefault())
    return timeFormat.format(time)
}

fun getFileNameFromUri(uri: Uri): String {
    var result = ""
    val cursor = ACTIVITY.contentResolver.query(uri , null ,
null , null , null)
    try {
        if (cursor != null && cursor.moveToFirst()){
            result =
cursor.getString(cursor.getColumnIndexOrThrow(OpenableColumns.DISPLAY_NAME))
        }
    }catch (e:Exception) {
        showToast(e.message.toString())
    }finally {
        cursor?.close()
        return result
    }
}

fun getPlurals(count : Int) =
ACTIVITY.resources.getQuantityString(
    R.plurals.members_count , count , count
)

```

Файл AppDrawer.kt

```

package com.example.massmess.allPackage.ui.objects

import android.graphics.drawable.Drawable
import android.net.Uri
import android.view.View
import android.widget.ImageView
import androidx.drawerlayout.widget.DrawerLayout
import com.example.massmess.R
import com.example.massmess.allPackage.database.USER
import
com.example.massmess.allPackage.ui.fragments.contacts.ContactsFr
agment
import
com.example.massmess.allPackage.ui.fragments.groups.AddContactsF
ragment
import
com.example.massmess.allPackage.ui.fragments.settings.SettingsFr
agment

```

```

import com.example.massmess.allPackage.utilits.ACTIVITY
import
com.example.massmess.allPackage.utilits.downloadAndSetImage
import com.example.massmess.allPackage.utilits.switchFragment
import com.mikepenz.materialdrawer.AccountHeader
import com.mikepenz.materialdrawer.AccountHeaderBuilder
import com.mikepenz.materialdrawer.Drawer
import com.mikepenz.materialdrawer.DrawerBuilder
import com.mikepenz.materialdrawer.model.PrimaryDrawerItem
import com.mikepenz.materialdrawer.model.ProfileDrawerItem
import com.mikepenz.materialdrawer.model.interfaces.IDrawerItem
import
com.mikepenz.materialdrawer.util.AbstractDrawerImageLoader
import com.mikepenz.materialdrawer.util.DrawerImageLoader

class AppDrawer {

    private lateinit var mDrawer: Drawer
    private lateinit var mHeader: AccountHeader
    private lateinit var mDrawerLayout: DrawerLayout
    private lateinit var mCurrentProfile: ProfileDrawerItem

    fun create() {
        initImageLoader()
        createHeader()
        createDrawer()
        mDrawerLayout = mDrawer.drawerLayout
    }

    fun disableDrawer() {

        mDrawer.actionBarDrawerToggle?.isDrawerIndicatorEnabled
= false

ACTIVITY.supportActionBar?.setDisplayHomeAsUpEnabled(true)

mDrawerLayout.setDrawerLockMode(DrawerLayout.LOCK_MODE_LOCKED_CLOSED)
        ACTIVITY.mToolbar.setNavigationOnClickListener {
            ACTIVITY.supportFragmentManager.popBackStack()
        }
    }

    fun enableDrawer() {

ACTIVITY.supportActionBar?.setDisplayHomeAsUpEnabled(false)
        mDrawer.actionBarDrawerToggle?.isDrawerIndicatorEnabled
= true

```

```

mDrawerLayout.setDrawerLockMode(DrawerLayout.LOCK_MODE_UNLOCKED)
    ACTIVITY.mToolbar.setNavigationOnClickListener {
        mDrawer.openDrawer()
    }
}

private fun createDrawer() {
    mDrawer = DrawerBuilder()
        .withActivity(ACTIVITY)
        .withToolbar(ACTIVITY.mToolbar)
        .withActionBarDrawerToggle(true)
        .withSelectedItem(-1)
        .withAccountHeader(mHeader)
        .addDrawerItems(
            PrimaryDrawerItem().withIdentifier(100)
                .withIconTintingEnabled(true)
                .withName("Создать группу")
                .withSelectable(false)
                .withIcon(R.drawable.ic_menu_create_groups),
            PrimaryDrawerItem().withIdentifier(103)
                .withIconTintingEnabled(true)

                .withName("Контакты")
                .withSelectable(false)
                .withIcon(R.drawable.ic_menu_contacts),
            PrimaryDrawerItem().withIdentifier(106)
                .withIconTintingEnabled(true)
                .withName("Настройки")
                .withSelectable(false)
                .withIcon(R.drawable.ic_menu_settings)
        ).withOnDrawerItemClickListener(object :
Drawer.OnDrawerItemClickListener {
    override fun onItemClick(
        view: View?,
        position: Int,
        drawerItem: IDrawerItem<*>
    ): Boolean {
        clickOnItem(position)
        return false
    }
}).build()

}

private fun clickOnItem(position: Int) {
    when (position) {
        1-> switchFragment(AddContactsFragment())
        3 -> switchFragment(SettingsFragment())
        2 -> switchFragment(ContactsFragment())
    }
}

```

```

private fun createHeader() {
    mCurrentProfile = ProfileDrawerItem()
        .withName(USER.fullname)
        .withEmail(USER.phone)
        .withIcon(USER.photoUrl)
        .withIdentifier(200)
    mHeader = AccountHeaderBuilder()
        .withActivity(ACTIVITY)
        .withHeaderBackground(R.drawable.header)
        .addProfiles(
            mCurrentProfile
        ).build()
}

fun updateHeader() {
    mCurrentProfile
        .withName(USER.fullname)
        .withEmail(USER.phone)
        .withIcon(USER.photoUrl)

    mHeader.updateProfile(mCurrentProfile)
}

private fun initImageLoader() {
    DrawerImageLoader.init(object :
AbstractDrawerImageLoader() {
        override fun set(imageView: ImageView, uri: Uri,
placeholder: Drawable) {
            imageView.downloadAndSetImage(uri.toString())
        }
    })
}
}

```

Файл SettingsFragment.kt

```

package com.example.massmess.allPackage.ui.fragments.settings

import android.app.Activity.RESULT_OK
import android.content.Intent
import android.view.Menu
import android.view.MenuInflater
import android.view.MenuItem
import com.example.massmess.R
import com.example.massmess.allPackage.database.*
import
com.example.massmess.allPackage.ui.fragments.base.BaseFragment
import com.example.massmess.allPackage.utilits.*
import com.theartofdev.edmodo.cropper.CropImage
import com.theartofdev.edmodo.cropper.CropImageView
import kotlinx.android.synthetic.main.fragment_settings.*

```

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		71

```

class SettingsFragment :
BaseFragment(R.layout.fragment_settings) {

    override fun onResume() {
        super.onResume()
        ACTIVITY.title = "Настройки"
        setHasOptionsMenu(true)
        initFields()
    }

    private fun initFields() {
        settings_bio.text = USER.bio
        settings_full_name.text = USER.fullname
        settings_phone_number.text = USER.phone
        settings_status.text = USER.state
        settings_username.text = USER.username
        settings_btn_change_username.setOnClickListener {
switchFragment(ChangeUsernameFragment()) }
        settings_btn_change_bio.setOnClickListener {

switchFragment(ChangeBioFragment()) }

settings_change_photo.setOnClickListener { changeUserPhoto() }
        settings_user_photo.downloadAndSetImage(USER.photoUrl)
    }

    private fun changeUserPhoto() {
        CropImage.activity()
            .setGuidelines(CropImageView.Guidelines.ON)
            .setMultiTouchEnabled(true)
            .setAspectRatio(1, 1)
            .setRequestedSize(250, 250,
CropImageView.RequestSizeOptions.RESIZE_INSIDE)
            .setMinCropWindowSize(0, 0)
            .setCropShape(CropImageView.CropShape.OVAL)
            .start(ACTIVITY, this)
    }

    override fun onCreateOptionsMenu(menu: Menu, inflater:
MenuInflater) {

activity?.menuInflater?.inflate(R.menu.settings_action_menu,
menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean
{
    when (item.itemId) {
        R.id.settings_menu_exit -> {

```

```

States.updateState(States.OFFLINE)
        AUTH.signOut()
        restartActivity()
    }
    R.id.settings_menu_change_name -> {
        switchFragment(ChangeNameFragment())
    }
}
return true
}

override fun onActivityResult(requestCode: Int, resultCode:
Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode ==
CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE && resultCode ==
RESULT_OK && data != null) {
        val uri = CropImage.getActivityResult(data).uri
        val path =
REF_STORAGE_ROOT.child(FOLDER_PROFILE_IMAGE).child(UID)

        putFileToStorage(uri, path) {

            getUrlStorage(path) {
                putUrlIntoDatabase(it) {

settings_user_photo.downloadAndSetImage(it)
                    showToast("Данные обновлены")
                    USER.photoUrl = it
                    ACTIVITY.mAppDrawer.updateHeader()
                }
            }
        }
    }
}

override fun onPause() {
    super.onPause()
}
}

```

Файл EnterPhoneNumberFragment.kt

```

package com.example.massmess.allPackage.ui.fragments.register
import androidx.fragment.app.Fragment
import com.example.massmess.R
import com.example.massmess.allPackage.database.AUTH
import com.example.massmess.allPackage.utilits.ACTIVITY
import com.example.massmess.allPackage.utilits.restartActivity
import com.example.massmess.allPackage.utilits.showToast

```

```

import com.example.massmess.allPackage.utilits.switchFragment
import com.google.firebase.FirebaseException
import com.google.firebase.auth.PhoneAuthCredential
import com.google.firebase.auth.PhoneAuthOptions
import com.google.firebase.auth.PhoneAuthProvider
import
kotlinx.android.synthetic.main.fragment_enter_phone_number.*
import java.util.concurrent.TimeUnit

class EnterPhoneNumberFragment :
    Fragment(R.layout.fragment_enter_phone_number) {

    private lateinit var mPhoneNumber: String
    private lateinit var mCallback:
        PhoneAuthProvider.OnVerificationStateChangedCallbacks

    override fun onStart() {
        super.onStart()
        mCallback = object :
            PhoneAuthProvider.OnVerificationStateChangedCallbacks() {
                override fun onVerificationCompleted(credential:
                    PhoneAuthCredential) {

                    AUTH.signInWithCredential(credential).addOnCompleteListener() {

                        if (it.isSuccessful) {

                            showToast("Добро пожаловать")
                                restartActivity()
                        }
                        else

                            showToast(it.exception?.message.toString())
                                }
                    }

                override fun onVerificationFailed(p0:
                    FirebaseException) {
                        showToast(p0.message.toString())
                    }

                override fun onCodeSent(id: String, token:
                    PhoneAuthProvider.ForceResendingToken) {
                        switchFragment(EnterCodeFragment(mPhoneNumber ,
                            id))
                    }

                }

            register_enter_phone_btn_next.setOnClickListener {
                sendCode()
            }
        }
    }

```

```

    }

    private fun sendCode() {
        if (register_input_phone.text.toString().isEmpty()) {
            showToast("Введите номер телефона")
        } else {
            authUser()
        }
    }

    private fun authUser() {
        mPhoneNumber = register_input_phone.text.toString()

        PhoneAuthProvider.verifyPhoneNumber(
            PhoneAuthOptions.newBuilder()
                .setPhoneNumber(mPhoneNumber)
                .setTimeout(60, TimeUnit.SECONDS)
                .setActivity(ACTIVITY)
                .setCallbacks(mCallback)
                .build()
        )
    }
}

```

Файл EnterCodeFragment.kt

```

package com.example.massmess.allPackage.ui.fragments.register

import androidx.fragment.app.Fragment
import com.example.massmess.R
import com.example.massmess.allPackage.database.*
import com.example.massmess.allPackage.utilits.*
import com.google.firebase.auth.PhoneAuthProvider
import kotlinx.android.synthetic.main.fragment_enter_code.*

class EnterCodeFragment(val phoneNumber: String, val id: String)
:
    Fragment(R.layout.fragment_enter_code) {

    override fun onStart() {
        super.onStart()
        ACTIVITY.title = phoneNumber

        register_input_code.addTextChangedListener(AppTextWatcher {

            val string = register_input_code.text.toString()
            if (string.length == 6) {
                verifyCode()
            }
        })
    }
}

```

```

    })

    }

    private fun verifyCode() {
        val code = register_input_code.text.toString()
        val credential = PhoneAuthProvider.getCredential(id,
code)

AUTH.signInWithCredential(credential).addOnCompleteListener() {
    if (it.isSuccessful) {
        val uid = AUTH.currentUser?.uid.toString()
        var dataMap = mutableMapOf<String, Any>()
        dataMap[CHILD_ID] = uid
        dataMap[CHILD_PHONE] = phoneNumber
        dataMap[CHILD_USERNAME] = uid

        REF_DATABASE_ROOT.child(NODE_USERS).child(uid)

.addListenerForSingleValueEvent(AppValueEventListener{

            if(!it.hasChild(CHILD_USERNAME)) {
                dataMap[CHILD_USERNAME] = uid
            }

REF_DATABASE_ROOT.child(NODE_PHONES).child(phoneNumber).setValue
(uid)

.addOnSuccessListener {

REF_DATABASE_ROOT.child(NODE_USERS).child(uid).updateChildren(da
taMap)

.addOnSuccessListener {
    showToast("Добро
пожаловать")

    restartActivity()
}

}

        })

    } else
        showToast(it.exception?.message.toString())
    }
    showToast("OK")

}

}

```

Файл ViewFactory.kt

```
package
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.views

import com.example.massmess.allPackage.models.DefaultModel
import com.example.massmess.allPackage.utilits.TYPE_MESSAGE_FILE
import
com.example.massmess.allPackage.utilits.TYPE_MESSAGE_IMAGE
import
com.example.massmess.allPackage.utilits.TYPE_MESSAGE_VOICE

class ViewFactory {
    companion object {
        fun getView(message : DefaultModel) : MessageView {
            return when(message.messageType) {
                TYPE_MESSAGE_IMAGE ->
                ViewImageMessage(message.id , message.from ,
                message.timeStamp.toString() , message.fileUrl)
                TYPE_MESSAGE_VOICE ->
                ViewVoiceMessage(message.id , message.from ,
                message.timeStamp.toString() , message.fileUrl)
                TYPE_MESSAGE_FILE -> ViewFileMessage(message.id
                , message.from , message.timeStamp.toString() , message.fileUrl
                , message.messageText)
                else -> {
                    ViewTextMessage(message.id , message.from ,
                    message.timeStamp.toString() , message.fileUrl ,
                    message.messageText)
                }
            }
        }
    }
}
```

Файл MessageView.kt

```
package
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.views

interface MessageView {
    val id: String
    val from: String
    val timeStamp: String
    val fileUrl: String
    val messageText: String
    companion object {
```

					ПГУ 1.090301.33.001 ПЗ	Лист
						77
Изм.	Лист	№ докум	Подпись	Дата		

```

val MESSAGE_IMAGE: Int
    get() = 0
val MESSAGE_TEXT: Int
    get() = 1
val MESSAGE_VOICE: Int
    get() = 2
val MESSAGE_FILE: Int
    get() = 3
}

fun getViewType(): Int
}

```

Файл VoiceMessageHolder.kt

```

package
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.viewholders

import android.view.View
import android.widget.ImageView
import android.widget.TextView
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.recyclerview.widget.RecyclerView
import com.example.massmess.allPackage.database.UID
import
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.views.MessageView
import com.example.massmess.allPackage.utilits.VoicePlayer
import com.example.massmess.allPackage.utilits.asTime
import kotlinx.android.synthetic.main.message_item_voice.view.*

class VoiceMessageHolder (view : View) :
RecyclerView.ViewHolder(view) , MessageHolder {

    private val mVoicePlayer : VoicePlayer = VoicePlayer()

    private val blockReceivedVoice: ConstraintLayout =
view.block_received_voice
    private val blockUserVoice: ConstraintLayout =
view.block_user_voice
    private val chatReceivedVoiceTime: TextView =
view.chats_received_voice_time
    private val chatUserVoiceTime: TextView =
view.chats_user_voice_time

    private val chatReceivedBtnPlay : ImageView =
view.chat_received_btn_play
    private val chatReceivedBtnStop : ImageView =
view.chat_received_btn_stop
    private val chatUserBtnPlay : ImageView =
view.chat_user_btn_play

```

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		78

```

private val chatUserBtnStop : ImageView =
view.chat_user_btn_stop

override fun showMessage(view: MessageView) {
    mVoicePlayer.init()
    if (view.from == UID) {

        blockReceivedVoice.visibility = View.GONE
        blockUserVoice.visibility = View.VISIBLE

        chatUserVoiceTime.text =
            view.timeStamp.asTime()

    } else {

        blockReceivedVoice.visibility = View.VISIBLE
        blockUserVoice.visibility = View.GONE

        chatReceivedVoiceTime.text =
            view.timeStamp.asTime()
    }
}

override fun onAttach(view: MessageView) {

    if(view.from == UID) {
        chatUserBtnPlay.setOnClickListener {

            chatUserBtnPlay.visibility = View.GONE
            chatUserBtnStop.visibility = View.VISIBLE

            chatUserBtnStop.setOnClickListener{

stop {
                                chatUserBtnStop.setOnClickListener(null)
                                chatUserBtnPlay.visibility =
View.VISIBLE
                                chatUserBtnStop.visibility = View.GONE
                            }
                        }

            play(view){
                chatUserBtnPlay.visibility = View.VISIBLE
                chatUserBtnStop.visibility = View.GONE
            }
        }
    }
    else {

```

```

chatReceivedBtnPlay.setOnClickListener {
    chatReceivedBtnPlay.visibility = View.GONE
    chatReceivedBtnStop.visibility = View.VISIBLE
    chatReceivedBtnStop.setOnClickListener{
        stop {

chatReceivedBtnStop.setOnClickListener(null)
            chatReceivedBtnPlay.visibility =
View.VISIBLE
            chatReceivedBtnStop.visibility =
View.GONE
        }
    }

    play(view){
        chatReceivedBtnPlay.visibility =
View.VISIBLE
        chatReceivedBtnStop.visibility = View.GONE
    }
}

}

private fun play(view: MessageView, function: () -> Unit) {
    mVoicePlayer.play(view.id , view.fileUrl) {
        function()
    }
}

private fun stop(function: () -> Unit) {
    mVoicePlayer.stop {
        function()
    }
}

override fun onDetach() {
    chatReceivedBtnPlay.setOnClickListener(null)
    chatUserBtnPlay.setOnClickListener(null)
    mVoicePlayer.release()
}
}

```

Файл TextMessageHolder.kt

```
package
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.viewholders

import android.view.View
import android.widget.TextView
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.recyclerview.widget.RecyclerView
import com.example.massmess.allPackage.database.UID
import
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.views.MessageView
import com.example.massmess.allPackage.utilits.asTime
import kotlinx.android.synthetic.main.message_item_text.view.*

class TextMessageHolder(view : View) :
RecyclerView.ViewHolder(view) , MessageHolder {

    private val blockUserMessage: ConstraintLayout =
view.block_user_message
    private val chatUserMessage: TextView =
view.chats_user_message
    private val chatUserMessageTime: TextView =
view.chats_user_message_time

    private val blockReceivedMessage: ConstraintLayout =
view.block_received_message
    private val chatReceivedMessage: TextView =
view.chats_received_message
    private val chatReceivedMessageTime: TextView =
view.chats_received_message_time

    override fun showMessage(view: MessageView) {
        if (view.from == UID) {
            blockUserMessage.visibility = View.VISIBLE
            blockReceivedMessage.visibility = View.GONE
            chatUserMessage.text = view.messageText
            chatUserMessageTime.text =
                view.timeStamp.asTime()

        } else {

            blockUserMessage.visibility = View.GONE
            blockReceivedMessage.visibility = View.VISIBLE
            chatReceivedMessage.text = view.messageText
            chatReceivedMessageTime.text =
                view.timeStamp.asTime()
        }
    }
}
```

```

    }

    override fun onAttach(view: MessageView) {

    }

    override fun onDetach() {

    }
}

```

Файл ImageMessageHolder.kt

```

package
com.example.massmess.allPackage.ui.fragments.message_recycler_view.viewholders

import android.view.View
import android.widget.ImageView
import android.widget.TextView
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.recyclerview.widget.RecyclerView
import com.example.massmess.allPackage.database.UID
import
com.example.massmess.allPackage.ui.fragments.message_recycler_view.views.MessageView
import com.example.massmess.allPackage.utilits.asTime
import
com.example.massmess.allPackage.utilits.downloadAndSetImage
import kotlinx.android.synthetic.main.message_item_image.view.*

class ImageMessageHolder (view : View) :
RecyclerView.ViewHolder(view) , MessageHolder {

    private val blockReceivedImage: ConstraintLayout =
view.block_received_image
    private val blockUserImage: ConstraintLayout =
view.block_user_image
    private val chatReceivedImage: ImageView =
view.chat_received_image
    private val chatUserImage: ImageView = view.chat_user_image
    private val chatReceivedImageTime: TextView =
view.chats_received_image_time
    private val chatUserImageTime: TextView =
view.chats_user_image_time

    override fun showMessage(view: MessageView) {
        if (view.from == UID) {

            blockReceivedImage.visibility = View.GONE
            blockUserImage.visibility = View.VISIBLE

```

```

        chatUserImage.downloadAndSetImage(view.fileUrl)
        chatUserImageTime.text =
            view.timeStamp.asTime()

    } else {

        blockReceivedImage.visibility = View.VISIBLE
        blockUserImage.visibility = View.GONE

        chatReceivedImage.downloadAndSetImage(view.fileUrl)
        chatReceivedImageTime.text =
            view.timeStamp.asTime()
    }
}

override fun onAttach(view: MessageView) {

}

override fun onDetach() {

}
}

```

Файл FileMessageHolder.kt

```

package
com.example.massmess.allPackage.ui.fragments.message_recycler_view.viewholders

import android.os.Environment
import android.view.View
import android.widget.ImageView
import android.widget.ProgressBar
import android.widget.TextView
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.recyclerview.widget.RecyclerView
import com.example.massmess.allPackage.database.UID
import
com.example.massmess.allPackage.database.getFileFromStorage
import
com.example.massmess.allPackage.ui.fragments.message_recycler_view.views.MessageView

import com.example.massmess.allPackage.utilits.WRITE_FILES
import com.example.massmess.allPackage.utilits.asTime
import com.example.massmess.allPackage.utilits.checkPermission
import com.example.massmess.allPackage.utilits.showToast
import kotlinx.android.synthetic.main.message_item_file.view.*
import java.io.File

```

```

class FileMessageHolder (view : View) :
RecyclerView.ViewHolder(view) , MessageHolder {

    private val blockReceivedFile: ConstraintLayout =
view.block_received_file
    private val blockUserFile: ConstraintLayout =
view.block_user_file
    private val chatReceivedFileTime: TextView =
view.chats_received_file_time
    private val chatUserFileTime: TextView =
view.chats_user_file_time

    private val chatUserFileName : TextView =
view.chat_user_filename
    private val chatUserBtnDownload : ImageView =
view.chat_user_btn_download
    private val chatUserProgressBar : ProgressBar =
view.chat_user_progress_bar

    private val chatReceivedFileName : TextView =
view.chat_received_filename
    private val chatReceivedBtnDownload : ImageView =
view.chat_received_btn_download
    private val chatReceivedProgressBar : ProgressBar =
view.chat_received_progress_bar


    override fun showMessage(view: MessageView) {
        if (view.from == UID) {

            blockReceivedFile.visibility = View.GONE
            blockUserFile.visibility = View.VISIBLE
            chatUserFileTime.text = view.timeStamp.asTime()
            chatUserFileName.text = view.messageText

        } else {

            blockReceivedFile.visibility = View.VISIBLE
            blockUserFile.visibility = View.GONE
            chatReceivedFileTime.text = view.timeStamp.asTime()

            chatReceivedFileName.text = view.messageText

        }
    }
}

```

```

        override fun onAttach(view: MessageView) {

            if (view.from == UID) {
                chatUserBtnDownload.setOnClickListener {
                    clickOnFileBtn(view)
                }
            } else {
                chatReceivedBtnDownload.setOnClickListener {
                    clickOnFileBtn(view)
                }
            }
        }

        private fun clickOnFileBtn(view: MessageView) {
            if (view.from == UID) {
                chatUserBtnDownload.visibility = View.INVISIBLE
                chatUserProgressBar.visibility = View.VISIBLE
            } else {
                chatReceivedBtnDownload.visibility = View.INVISIBLE
                chatReceivedProgressBar.visibility = View.VISIBLE
            }

            val file = File(
                Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS),
                view.messageText
            )

            try {
                if (checkPermission(WRITE_FILES)) {
                    file.createNewFile()
                    getFileFromStorage(file, view.fileUrl) {
                        if (view.from == UID) {
                            chatUserBtnDownload.visibility =
                                View.VISIBLE
                            chatUserProgressBar.visibility =
                                View.INVISIBLE
                        } else {
                            chatReceivedBtnDownload.visibility =
                                View.VISIBLE
                            chatReceivedProgressBar.visibility =
                                View.INVISIBLE
                        }
                    }
                }
            } catch (e: Exception) {

                showToast(e.message.toString())
            }
        }
    }

```

```

override fun onDetach() {

    chatUserBtnDownload.setOnClickListener(null)
    chatReceivedBtnDownload.setOnClickListener(null)

}

}

```

Файл HolderFactory.kt

```

package
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.viewholders

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.example.massmess.R
import
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.views.MessageView

class HolderFactory {
    companion object {

        fun getHolder(parent : ViewGroup , viewType : Int) :
RecyclerView.ViewHolder {
            return when (viewType) {
                MessageView.MESSAGE_IMAGE -> {
                    val view =
LayoutInflater.from(parent.context).inflate(R.layout.message_ite
m_image, parent, false)
                    ImageMessageHolder(view)
                }
                MessageView.MESSAGE_VOICE -> {
                    val view =
LayoutInflater.from(parent.context).inflate(R.layout.message_ite
m_voice, parent, false)
                    VoiceMessageHolder(view)
                }
                MessageView.MESSAGE_FILE -> {
                    val view =
LayoutInflater.from(parent.context).inflate(R.layout.message_ite
m_file, parent, false)
                    FileMessageHolder(view)
                }
                else -> {

```

```

        val view =
        LayoutInflater.from(parent.context).inflate(R.layout.message_ite
m_text, parent, false)
                TextMessageHolder(view)
            }
        }
    }
}

```

Файл ChatsFragment.kt

```

package com.example.massmess.allPackage.ui.fragments.main_list

import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.RecyclerView
import com.example.massmess.R
import com.example.massmess.allPackage.database.*
import com.example.massmess.allPackage.models.DefaultModel
import com.example.massmess.allPackage.utilits.*
import kotlinx.android.synthetic.main.fragment_chats.*

class ChatsFragment : Fragment(R.layout.fragment_chats) {

    private lateinit var mRecyclerView : RecyclerView
    private lateinit var mAdapter : ChatsAdapter
    private val mRefMainList =
    REF_DATABASE_ROOT.child(NODE_MAIN_LIST).child(UID)
    private val mRefUsers = REF_DATABASE_ROOT.child(NODE_USERS)
    private val mRefMessages =

    REF_DATABASE_ROOT.child(NODE_MESSAGES).child(UID)
    private var mListItems = listOf<DefaultModel>()

    override fun onResume() {
        super.onResume()
        ACTIVITY.title = "Чаты"
        ACTIVITY.mAppDrawer.enableDrawer()
        closeKeyboard()
        initRecyclerView()
    }

    private fun initRecyclerView() {
        mRecyclerView = main_list_recycler_view
        mAdapter = ChatsAdapter()

        mRefMainList.addListenerForSingleValueEvent(AppValueEventListener { dataSnapshot ->

```

```

mListItems = dataSnapshot.children.map { it.getDefaultModel() }
    mListItems.forEach { model->

        when(model.messageType) {
            TYPE_CHAT -> showChat(model)
            TYPE_GROUP -> showGroup(model)
        }

    }

})

mRecyclerView.adapter = mAdapter

}

private fun showGroup(model: DefaultModel) {

    REF_DATABASE_ROOT.child(NODE_GROUPS).child(model.id).addListener
    ForSingleValueEvent(AppValueEventListener { dataSnapshot1->
        val newModel = dataSnapshot1.getDefaultModel()

    REF_DATABASE_ROOT.child(NODE_GROUPS).child(model.id).child(NODE_
    MESSAGES)

    .limitToLast(1).addListenerForSingleValueEvent(AppValueEventList
    ener { dataSnapshot2->
        val tempList = dataSnapshot2.children.map {
    it.getDefaultModel() }
        if(tempList.isEmpty()) {
            newModel.lastMessage = "Чат очищен"
        } else {
            newModel.lastMessage =

tempList[0].messageText
        }
        newModel.messageType = TYPE_GROUP
        mAdapter.updateItemsList(newModel)
    })
    })
}

private fun showChat(model: DefaultModel) {

    mRefUsers.child(model.id).addListenerForSingleValueEvent(AppValu
    eEventListener { dataSnapshot1->
        val newModel = dataSnapshot1.getDefaultModel()

    mRefMessages.child(model.id).limitToLast(1).addListenerForSingle
    ValueEvent(AppValueEventListener { dataSnapshot2->
        val tempList = dataSnapshot2.children.map {
    it.getDefaultModel() }

```

```

        if (tempList.isEmpty()) {
            newModel.lastMessage = "Чат очищен"
        } else {
            newModel.lastMessage =
tempList[0].messageText
        }

        if (newModel.fullname.isEmpty()) {
            newModel.fullname = newModel.phone
        }
        newModel.messageType = TYPE_CHAT
        mAdapter.updateItemsList(newModel)
    })
})
}

}

```

Файл IndividualChatFragment.kt

```

package
com.example.massmess.allPackage.ui.fragments.individual_chat

import android.content.Intent
import android.net.Uri
import android.view.*
import android.widget.AbsListView
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import androidx.swiperefreshlayout.widget.SwipeRefreshLayout
import com.example.massmess.R
import com.example.massmess.allPackage.database.*
import com.example.massmess.allPackage.models.DefaultModel
import com.example.massmess.allPackage.models.User
import
com.example.massmess.allPackage.ui.fragments.base.BaseFragment
import
com.example.massmess.allPackage.ui.fragments.main_list.ChatsFrag
ment
import
com.example.massmess.allPackage.ui.fragments.message_recycler_vi
ew.views.ViewFactory
import com.example.massmess.allPackage.utilits.*
import
com.google.android.material.bottomsheet.BottomSheetBehavior
import com.google.firebase.database.DatabaseReference
import com.theartofdev.edmodo.cropper.CropImage
import com.theartofdev.edmodo.cropper.CropImageView
import kotlinx.android.synthetic.main.activity_main.view.*
import kotlinx.android.synthetic.main.fragment_individual_chat.*
import kotlinx.android.synthetic.main.toolbar_chats.view.*
import kotlinx.android.synthetic.main.upload_choice.*

```

```

import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class IndividualChatFragment(private val contact: DefaultModel)
:
    BaseFragment(R.layout.fragment_individual_chat) {

    private lateinit var mListenerInfoToolbar:
AppValueEventListener
    private lateinit var mReceivingUser: User
    private lateinit var mToolBarInfo: View
    private lateinit var mRefToUser: DatabaseReference
    private lateinit var mRefToMessages: DatabaseReference
    private lateinit var mAdapter: IndividualChatAdapter
    private lateinit var mRecyclerView: RecyclerView
    private lateinit var mMessagesListener:
AppChildEventListener
    private var mCountMessagesToLoad = 10
    private var mIsUserScrolling = false
    private var mSmoothScrollToPosition = true
    private lateinit var mSwipeRefreshLayout: SwipeRefreshLayout
    private lateinit var mLayoutManager: LinearLayoutManager
    private lateinit var mAppVoiceRecorder: AppVoiceRecorder
    private lateinit var mBottomSheetBehavior:
BottomSheetBehavior<*>

    override fun onResume() {
        super.onResume()
        initFields()
        initToolbar()
        initRecyclerView()
    }

    private fun initFields() {
        setHasOptionsMenu(true)
        mBottomSheetBehavior =
BottomSheetBehavior.from(bottom_sheet_choice)
        mBottomSheetBehavior.state =
BottomSheetBehavior.STATE_HIDDEN
        mAppVoiceRecorder = AppVoiceRecorder()
        mSwipeRefreshLayout = chats_swipe_refresh
        mLayoutManager = LinearLayoutManager(this.context)

        chats_input_message.addTextChangedListener(AppTextWatcher {
            val input = chats_input_message.text.toString()
            if(input.isEmpty() || input == "Идет запись"){
                chats_btn_send.visibility = View.GONE
                chats_btn_attch.visibility = View.VISIBLE
                chats_btn_voice_message.visibility =

```

```

View.VISIBLE

}else {

        chats_btn_send.visibility = View.VISIBLE
        chats_btn_attch.visibility = View.GONE
        chats_btn_voice_message.visibility = View.GONE
    }

    })

    chats_btn_attch.setOnClickListener {attach()}
    CoroutineScope(Dispatchers.IO).launch {
        chats_btn_voice_message.setOnTouchListener { v,
event ->

        if (checkPermission(RECORD_AUDIO)) {
            if (event.action == MotionEvent.ACTION_DOWN)
{
                chats_input_message.setText("Идет
запись")

                val messageKey =
getMessageKey(contact.id)

mAppVoiceRecorder.startRecording(messageKey)

            } else if (event.action ==
MotionEvent.ACTION_UP) {
                chats_input_message.setText("")
                mAppVoiceRecorder.stopRecording{ file ,
messageKey ->

uploadFileToStorage(Uri.fromFile(file) , messageKey , contact.id
, TYPE_MESSAGE_VOICE)

                mSmoothScrollToPosition = true

            }

        }

    }

    true

}

}

private fun attach() {
    mBottomSheetBehavior.state =
BottomSheetBehavior.STATE_EXPANDED
    attach_file_btn.setOnClickListener {
        attachFile()
    }
    attach_image_btn.setOnClickListener {
        attachImage()
    }
}

```

```

    }

}

private fun attachFile() {

    val intent = Intent(Intent.ACTION_GET_CONTENT)
    intent.type = "*/*"
    startActivityForResult(intent ,
CHOOSE_FILE_REQUEST_CODE)

}

private fun attachImage() {
    CropImage.activity()
        .setGuidelines(CropImageView.Guidelines.ON)
        .setMultiTouchEnabled(true)
        .setAspectRatio(1, 1)
        .setRequestedSize(250, 250,
CropImageView.RequestSizeOptions.RESIZE_INSIDE)
        .setMinCropWindowSize(0, 0)
        .start(ACTIVITY, this)
}

private fun initRecyclerView() {
    mRecyclerView = chats_recycler_view
    mAdapter = IndividualChatAdapter()
    mRefToMessages =
REF_DATABASE_ROOT.child(NODE_MESSAGES).child(UID).child(contact.
id)

    mRecyclerView.adapter = mAdapter
    mRecyclerView.setHasFixedSize(true)
    mRecyclerView.isNestedScrollingEnabled = false

    mRecyclerView.layoutManager = LinearLayoutManager

    mMessagesListener = AppChildEventListener {

        val message = it.getDefaultModel()

        if (mSmoothScrollToPosition) {

mAdapter.addItemToBottom(ViewFactory.getView(message)) {

mRecyclerView.smoothScrollToPosition(mAdapter.itemCount)
        }
        } else {

mAdapter.addItemToTop(ViewFactory.getView(message)) {
            mSwipeRefreshLayout.isRefreshing = false
        }
        }
    }
}

```

```

    }

    mRefToMessages.limitToLast(mCountMessagesToLoad).addChildEventListener(mMessagesListener)

    mRecyclerView.addOnScrollListener(object :
RecyclerView.OnScrollListener() {
        override fun onScrolled(recyclerView: RecyclerView,
dx: Int, dy: Int) {
            super.onScrolled(recyclerView, dx, dy)
            if (mIsUserScrolling && dy < 0 &&
mLayoutManager.findFirstVisibleItemPosition() <= 3) {
                updateData()
            }
        }

        override fun onScrollStateChanged(recyclerView:
RecyclerView, newState: Int) {
            super.onScrollStateChanged(recyclerView,
newState)

            if (newState ==
AbsListView.OnScrollListener.SCROLL_STATE_TOUCH_SCROLL) {
                mIsUserScrolling = true
            }
        }
    })

    mSwipeRefreshLayout.setOnRefreshListener {
        updateData()
    }

}

private fun updateData() {
    mSmoothScrollToPosition = false
    mIsUserScrolling = false
    mCountMessagesToLoad += 10
    mRefToMessages.removeEventListener(mMessagesListener)

    mRefToMessages.limitToLast(mCountMessagesToLoad).addChildEventListener(mMessagesListener)

}

private fun initToolbar() {
    mToolBarInfo = ACTIVITY.mToolbar.toolbar_chats
    mToolBarInfo.visibility = View.VISIBLE

    mListenerInfoToolbar = AppValueEventListener {

```

```

        mReceivingUser = it.getUserModel()
        initInfoToolbar()
    }

    mRefToUser =
REF_DATABASE_ROOT.child(NODE_USERS).child(contact.id)
    mRefToUser.addValueEventListener(mListenerInfoToolbar)
    chats_btn_send.setOnClickListener {
        mSmoothScrollToPosition = true
        val message = chats_input_message.text.toString()
        if (message.isEmpty()) {
            showToast("Введите сообщение")
        } else {
            sendMessage(message, contact.id, TYPE_TEXT) {
                saveToMainList(contact.id , TYPE_CHAT)
                chats_input_message.setText("")
            }
        }
    }
}

private fun initInfoToolbar() {
    if (mReceivingUser.fullname.isEmpty()) {
        mToolBarInfo.toolbar_chats_fullname.text =
contact.fullname
    } else mToolBarInfo.toolbar_chats_fullname.text =
mReceivingUser.fullname

mToolBarInfo.toolbar_chats_image.downloadAndSetImage(mReceivingU
ser.photoUrl)

    mToolBarInfo.toolbar_chats_status.text =
mReceivingUser.state
}

override fun onPause() {
    super.onPause()
    mToolBarInfo.visibility = View.GONE
    mRefToUser.removeEventListener(mListenerInfoToolbar)
    mRefToMessages.removeEventListener(mMessagesListener)
}

override fun onActivityResult(requestCode: Int, resultCode:
Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if(data != null) {
        when (requestCode) {

```

```

        CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE -> {

val uri = CropImage.getActivityResult(data).uri
        val messageKey = getMessageKey(contact.id)
        uploadFileToStorage(uri, messageKey,
contact.id, TYPE_MESSAGE_IMAGE)
        mSmoothScrollToPosition = true
    }
    CHOOSE_FILE_REQUEST_CODE -> {

        val uri = data.data
        val messageKey = getMessageKey(contact.id)
        val filename = getFileNameFromUri(uri!!)
        uploadFileToStorage(uri, messageKey,
contact.id, TYPE_MESSAGE_FILE , filename)
        mSmoothScrollToPosition = true
    }
    }

}

override fun onDestroyView() {
    super.onDestroyView()
    mAppVoiceRecorder.releaseRecorder()
    mAdapter.destroy()
}

override fun onCreateOptionsMenu(menu: Menu, inflater:
MenuInflater) {

activity?.menuInflater?.inflate(R.menu.individual_chat_action_me
nu, menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean
    {
        when (item.itemId) {
            R.id.menu_clear_chat -> clearChat(contact.id){
                showToast("Чат очищен")
                switchFragment(ChatsFragment())
            }
            R.id.menu_delete_chat -> deleteChat(contact.id){
                showToast("Чат удален")
                switchFragment(ChatsFragment())
            }
        }
        return true
    }
}

```

Файл IndividualChatAdapter.kt

```
package
com.example.massmess.allPackage.ui.fragments.individual_chat

import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.viewholders.HolderFactory
import
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.viewholders.MessageHolder
import
com.example.massmess.allPackage.ui.fragments.message_recycler_vie
w.views.MessageView

class IndividualChatAdapter :
RecyclerView.Adapter<RecyclerView.ViewHolder>() {

    private var mListMessagesCache =
mutableListOf<MessageView>()
    private var mListHolders = mutableListOf<MessageHolder>()

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): RecyclerView.ViewHolder {

        return HolderFactory.getHolder(parent , viewType)
    }

    override fun getItemViewType(position: Int): Int {
        return mListMessagesCache[position].getViewType()
    }

    override fun onBindViewHolder(holder:
RecyclerView.ViewHolder, position: Int) {

        (holder as
MessageHolder).showMessage(mListMessagesCache[position])

    }

    override fun onViewAttachedToWindow(holder:
RecyclerView.ViewHolder) {
        (holder as
MessageHolder).onAttach(mListMessagesCache[holder.absoluteAdapte
rPosition])
    }
}
```

					ПГУ 1.090301.33.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		96

```

        mListHolders.add((holder as MessageHolder))
super.onViewAttachedToWindow(holder)
    }

    override fun onViewDetachedFromWindow(holder:
RecyclerView.ViewHolder) {
        (holder as MessageHolder).onDetach()
        mListHolders.remove((holder as MessageHolder))
        super.onViewDetachedFromWindow(holder)
    }

    override fun getItemCount(): Int = mListMessagesCache.size

    fun addItemToBottom(item: MessageView, onSuccess: () ->
Unit) {
        if (!mListMessagesCache.contains(item)) {
            mListMessagesCache.add(item)
            notifyItemInserted(mListMessagesCache.size)
        }
        onSuccess()
    }

    fun addItemToTop(item: MessageView, onSuccess: () -> Unit) {
        if (!mListMessagesCache.contains(item)) {
            mListMessagesCache.add(item)
            mListMessagesCache.sortBy { it.timeStamp }
            notifyItemInserted(0)
        }
        onSuccess()
    }

    fun destroy() {
        mListHolders.forEach {
            it.onDetach()
        }
    }
}

```

Файл GroupChatAdapter.kt

```

package com.example.massmess.allPackage.ui.fragments.groups

import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import
com.example.massmess.allPackage.ui.fragments.message_recycler_vi
ew.viewholders.*
import
com.example.massmess.allPackage.ui.fragments.message_recycler_vi
ew.views.MessageView

```

```

class GroupChatAdapter :
RecyclerView.Adapter<RecyclerView.ViewHolder>() {

    private var mListMessagesCache =
mutableListOf<MessageView>()
    private var mListHolders = mutableListOf<MessageHolder>()

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): RecyclerView.ViewHolder {

        return HolderFactory.getHolder(parent , viewType)
    }

    override fun getItemViewType(position: Int): Int {
        return mListMessagesCache[position].getViewType()
    }

    override fun onBindViewHolder(holder:
RecyclerView.ViewHolder, position: Int) {

        (holder as
MessageHolder).showMessage (mListMessagesCache[position])

    }

    override fun onViewAttachedToWindow(holder:
RecyclerView.ViewHolder) {
        (holder as
MessageHolder).onAttach(mListMessagesCache[holder.absoluteAdapte
rPosition])
        mListHolders.add((holder as MessageHolder))
        super.onViewAttachedToWindow(holder)
    }

    override fun onViewDetachedFromWindow(holder:
RecyclerView.ViewHolder) {
        (holder as MessageHolder).onDetach()
        mListHolders.remove((holder as MessageHolder))
        super.onViewDetachedFromWindow(holder)
    }

    override fun getItemCount(): Int = mListMessagesCache.size

    fun addItemToBottom(item: MessageView, onSuccess: () ->
Unit) {
        if (!mListMessagesCache.contains(item)) {
            mListMessagesCache.add(item)
            notifyItemInserted(mListMessagesCache.size)
        }
    }
}

```

```

        onSuccess()
    }

    fun addItemToTop(item: MessageView, onSuccess: () -> Unit) {
        if (!mListMessagesCache.contains(item)) {
            mListMessagesCache.add(item)
            mListMessagesCache.sortBy { it.timeStamp }
            notifyItemInserted(0)
        }
        onSuccess()
    }

    fun destroy() {
        mListHolders.forEach {
            it.onDetach()
        }
    }
}

```

Файл CreateGroupFragment.kt

```

package com.example.massmess.allPackage.ui.fragments.groups

import android.app.Activity
import android.content.Intent
import android.net.Uri
import androidx.recyclerview.widget.RecyclerView
import com.example.massmess.R
import com.example.massmess.allPackage.database.createGroupInDatabase
import com.example.massmess.allPackage.models.DefaultModel
import com.example.massmess.allPackage.ui.fragments.base.BaseFragment
import com.example.massmess.allPackage.ui.fragments.main_list.ChatsFragment
import com.example.massmess.allPackage.utilits.*
import com.theartofdev.edmodo.cropper.CropImage
import com.theartofdev.edmodo.cropper.CropImageView
import kotlinx.android.synthetic.main.fragment_create_group.*

class CreateGroupFragment(var contactsList: List<DefaultModel>)
: BaseFragment(R.layout.fragment_create_group) {

    private lateinit var mRecyclerView: RecyclerView
    private lateinit var mAdapterter: AddContactsAdapter
    private var mUri = Uri.EMPTY

```

```

        override fun onResume() {
            super.onResume()
            ACTIVITY.title = "Создать группу"
            closeKeyboard()
            initRecyclerView()
            create_group_photo.setOnClickListener {
                addPhoto()
            }
            create_group_btn_complete.setOnClickListener {
                val groupName =
create_group_input_name.text.toString()
                if(groupName.isEmpty()) {
                    showToast("Введите имя группы")
                } else {

                    createGroupInDatabase(groupName , mUri ,
contactsList) {

                        switchFragment(ChatsFragment())

                    }

                }
            }
            create_group_input_name.requestFocus()
            create_group_counts.text = getPlurals(contactsList.size)
        }

        private fun addPhoto() {
            CropImage.activity()
                .setGuidelines(CropImageView.Guidelines.ON)
                .setMultiTouchEnabled(true)
                .setAspectRatio(1, 1)
                .setRequestedSize(250, 250,
CropImageView.RequestSizeOptions.RESIZE_INSIDE)
                .setMinCropWindowSize(0, 0)
                .setCropShape(CropImageView.CropShape.OVAL)
                .start(ACTIVITY, this)
        }

        private fun initRecyclerView() {
            mRecyclerView = create_group_recycler_view
            mAdapter = AddContactsAdapter()
            mRecyclerView.adapter = mAdapter
            contactsList.forEach { mAdapter.updateItemsList(it) }
        }

        override fun onActivityResult(requestCode: Int, resultCode:
Int, data: Intent?) {
            super.onActivityResult(requestCode, resultCode, data)

```

```

        if (requestCode == CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE
        && resultCode == Activity.RESULT_OK && data != null) {
            mUri = CropImage.getActivityResult(data).uri
            create_group_photo.setImageURI(mUri)
        }
    }
}

```

Файл AddContactsAdapter.kt

```

package com.example.massmess.allPackage.ui.fragments.groups

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.massmess.R
import com.example.massmess.allPackage.models.DefaultModel
import
com.example.massmess.allPackage.utilits.downloadAndSetImage
import de.hdodenhof.circleimageview.CircleImageView
import kotlinx.android.synthetic.main.add_groups_item.view.*

class AddContactsAdapter :
    RecyclerView.Adapter<AddContactsAdapter.AddContactsHolder>() {

    private var itemsList = mutableListOf<DefaultModel>()

    class AddContactsHolder(view: View) :
        RecyclerView.ViewHolder(view) {
        val itemName: TextView = view.add_groups_item_name
        val itemLastMessage: TextView =
view.add_groups_last_message
        val itemPhoto: CircleImageView =
view.add_groups_contact_photo
        val itemChoice: CircleImageView =
view.add_groups_item_choice
    }
    override fun onCreateViewHolder(parent: ViewGroup, viewType:
    Int): AddContactsHolder {
        val view =
        LayoutInflater.from(parent.context).inflate(R.layout.add_groups_
        item, parent, false)
        val holder = AddContactsHolder(view)
        holder.itemView.setOnClickListener {
            if
            (itemsList[holder.absoluteAdapterPosition].choiced) {

```

```

        holder.itemChoice.visibility = View.INVISIBLE

        itemsList[holder.absoluteAdapterPosition].choiced =
false
AddContactsFragment.contactsList.remove(itemsList[holder.absolut
eAdapterPosition])
        } else {
            holder.itemChoice.visibility = View.VISIBLE

itemsList[holder.absoluteAdapterPosition].choiced = true

AddContactsFragment.contactsList.add(itemsList[holder.absoluteAd
apterPosition])
        }
    }
    return holder
}

    override fun onBindViewHolder(holder: AddContactsHolder,
position: Int) {
        holder.itemName.text = itemsList[position].fullname
        holder.itemLastMessage.text =
itemsList[position].lastMessage

holder.itemPhoto.downloadAndSetImage(itemsList[position].photoUr

1)

    }

    override fun getItemCount(): Int = itemsList.size

    fun updateItemsList(item: DefaultModel) {
        itemsList.add(item)
        notifyItemInserted(itemsList.size)
    }
}

```

Файл ContactsFragment.kt

```

package com.example.massmess.allPackage.ui.fragments.contacts
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.massmess.R
import com.example.massmess.allPackage.database.*
import

```

```

com.example.massmess.allPackage.utilits.downloadAndSetImage
import com.example.massmess.allPackage.utilits.switchFragment

import com.example.massmess.allPackage.models.DefaultModel
import
com.example.massmess.allPackage.ui.fragments.base.BaseFragment
import
com.example.massmess.allPackage.ui.fragments.individual_chat.IndividualChatFragment
import com.example.massmess.allPackage.utilits.ACTIVITY
import
com.example.massmess.allPackage.utilits.AppValueEventListener

import com.firebase.ui.database.FirebaseRecyclerAdapter
import com.firebase.ui.database.FirebaseRecyclerOptions
import com.google.firebase.database.DatabaseReference
import de.hdodenhof.circleimageview.CircleImageView
import kotlinx.android.synthetic.main.contact_item.view.*
import kotlinx.android.synthetic.main.fragment_contacts.*

class ContactsFragment :
BaseFragment(R.layout.fragment_contacts) {

    private lateinit var mRecyclerView : RecyclerView
    private lateinit var mAdapter :
FirebaseRecyclerAdapter<DefaultModel, ContactsHolder>

    private lateinit var mRefToContacts : DatabaseReference
    private lateinit var mRefToUsers : DatabaseReference
    private lateinit var mRefToUserListener :
AppValueEventListener
    private var mapOfListeners = HashMap<DatabaseReference ,
AppValueEventListener>()

    override fun onResume() {
        super.onResume()
        ACTIVITY.title = "Контакты"
        initRecyclerView()
    }

    private fun initRecyclerView() {
        mRecyclerView = contacts_recycler_view
        mRefToContacts =
REF_DATABASE_ROOT.child(NODE_PHONES_CONTACTS).child(UID)

        val options =
FirebaseRecyclerOptions.Builder<DefaultModel>()
            .setQuery(mRefToContacts, DefaultModel::class.java)
            .build()

```

```

        mAdapterter = object :
        FirebaseRecyclerAdapter<DefaultModel, ContactsHolder>(options){
            override fun onCreateViewHolder(parent: ViewGroup,
            viewType: Int): ContactsHolder {

                val view =
                LayoutInflater.from(parent.context).inflate(R.layout.contact_ite
                m , parent,false)
                return ContactsHolder(view)
            }

            override fun onBindViewHolder(
                holder: ContactsHolder,
                position: Int,
                model: DefaultModel
            ) {
                mRefToUsers =
                REF_DATABASE_ROOT.child(NODE_USERS).child(model.id)
                mRefToUserListener = AppValueEventListener {
                    val contact = it.getDefaultModel()

                    if (contact.fullname.isEmpty()){
                        holder.name.text = model.fullname
                    }
                    else holder.name.text = contact.fullname

                    holder.name.text = contact.fullname
                    holder.status.text = contact.state

                holder.photo.downloadAndSetImage(contact.photoUrl)
                holder.itemView.setOnClickListener {

                    switchFragment(IndividualChatFragment(model))
                }
            }

            mRefToUsers.addValueEventListener(mRefToUserListener)
            mapOfListeners[mRefToUsers] = mRefToUserListener

        }

    }

    mRecyclerView.adapter = mAdapterter
    mAdapterter.startListening()
}

class ContactsHolder(view :
View):RecyclerView.ViewHolder(view){

```

```

        val name : TextView = view.contact_fullname
        val status : TextView = view.contact_status
        val photo : CircleImageView = view.contact_photo
    }

    override fun onPause() {
        super.onPause()
    }

    mAdapter.stopListening()
    mapOfListeners.forEach {
        it.key.removeEventListener(it.value)
    }
}

```

Файл BaseFragment.kt

```

package com.example.massmess.allPackage.ui.fragments.base

import androidx.fragment.app.Fragment
import com.example.massmess.allPackage.utilits.ACTIVITY

open class BaseFragment(layout : Int) : Fragment(layout) {

    override fun onStart() {
        super.onStart()
        (ACTIVITY).mAppDrawer.disableDrawer()
    }
}

```

Файл BaseChangeFragment.kt

```

package com.example.massmess.allPackage.ui.fragments.base

import android.view.*
import androidx.fragment.app.Fragment
import com.example.massmess.R
import com.example.massmess.allPackage.utilits.ACTIVITY
import com.example.massmess.allPackage.utilits.closeKeyboard

open class BaseChangeFragment(layout : Int) : Fragment(layout) {
    override fun onStart() {
        super.onStart()
        setHasOptionsMenu(true)
        (ACTIVITY).mAppDrawer.disableDrawer()
    }
}

```

```

    }

    override fun onStop() {
        super.onStop()
        closeKeyboard()
    }

    override fun onCreateOptionsMenu(menu: Menu, inflater:
MenuInflater) {

        (ACTIVITY).menuInflater.inflate(R.menu.settings_menu_confirm ,
menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean
{

        when (item.itemId){
            R.id.settings_confirm_change -> change()
        }

        return true
    }

    open fun change() {

    }

}

```

Файл User.kt

```

package com.example.massmess.allPackage.models

data class User(
    val id: String = "",
    var username: String = "",
    var bio: String = "",
    var fullname: String = "",
    var state: String = "",
    var phone: String = "",
    var photoUrl: String = "null",
)

```

Файл DefaultModel.kt

```

package com.example.massmess.allPackage.models
data class DefaultModel(
    val id: String = "",

```

```

var username: String = "",
var bio: String = "",
var fullname: String = "",
var state: String = "",
var phone: String = "",
var photoUrl: String = "null",

var messageText : String = "",
var messageType : String = "",
var from : String = "",
var timeStamp : Any = "",

var fileUrl : String = "empty",

var lastMessage : String = "",
var choiced : Boolean = false

) {
    override fun equals(other: Any?): Boolean {
        return (other as DefaultModel).id == id
    }
}

```

Файл databaseFuncs.kt

```

package com.example.massmess.allPackage.database

import android.net.Uri
import com.example.massmess.allPackage.models.DefaultModel
import com.example.massmess.allPackage.models.User
import com.example.massmess.allPackage.utilits.ACTIVITY
import

com.example.massmess.allPackage.utilits.AppValueEventListener
import com.example.massmess.allPackage.utilits.TYPE_GROUP
import com.example.massmess.allPackage.utilits.showToast
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ServerValue
import com.google.firebase.storage.FirebaseStorage
import com.google.firebase.storage.StorageReference
import java.io.File

fun initFirebase() {
    AUTH = FirebaseAuth.getInstance()
    REF_DATABASE_ROOT = FirebaseDatabase.getInstance().reference
    USER = User()
    UID = AUTH.currentUser?.uid.toString()
    REF_STORAGE_ROOT = FirebaseStorage.getInstance().reference
}

```

```

inline fun putUrlIntoDatabase(url: String, crossinline function:
() -> Unit) {

REF_DATABASE_ROOT.child(NODE_USERS).child(UID).child(CHILD_PHOTO
_URL)
    .setValue(url)
    .addOnSuccessListener {
        function()
    }
    .addOnFailureListener {
        showToast(it.message.toString())
    }

}

inline fun getUrlStorage(path: StorageReference, crossinline
function: (url: String) -> Unit) {
    path.downloadUrl
        .addOnSuccessListener {
            function(it.toString())
        }
        .addOnFailureListener {
            showToast(it.message.toString())
        }
}

inline fun putFileToStorage(uri: Uri, path: StorageReference,
crossinline function: () -> Unit) {
    path.putFile(uri)
        .addOnSuccessListener {
            function()
        }
        .addOnFailureListener {

            showToast(it.message.toString())

        }
}

inline fun initUserModel(crossinline function: () -> Unit) {
    REF_DATABASE_ROOT.child(NODE_USERS).child(UID)
        .addListenerForSingleValueEvent(AppValueEventListener {
            USER = it.getValue(User::class.java) ?: User()
            if (USER.username.isEmpty()) {
                USER.username = UID
            }
            function()
        })
}

```

```

fun updatePhonesDatabase(contactsArray: ArrayList<DefaultModel>)
{
    if (AUTH.currentUser != null) {

REF_DATABASE_ROOT.child(NODE_PHONES).addListenerForSingleValueEv
ent(AppValueEventListener {
        it.children.forEach { dataSnapshot ->
            contactsArray.forEach { contact ->
                if (dataSnapshot.key == contact.phone) {

REF_DATABASE_ROOT.child(NODE_PHONES_CONTACTS).child(UID)

                .child(dataSnapshot.value.toString()).child(CHILD_ID)

                .setValue(dataSnapshot.value.toString())
                    .addOnFailureListener {
                        showToast("Error")
                    }

REF_DATABASE_ROOT.child(NODE_PHONES_CONTACTS).child(UID)

                .child(dataSnapshot.value.toString()).child(CHILD_FULLNAME)
                    .setValue(contact.fullname)
                    .addOnFailureListener {
                        showToast("Error")
                    }

            }
        }
    })
}

}

fun dataSnapshot.getDefaultModel(): DefaultModel =
    this.getValue(DefaultModel::class.java) ?: DefaultModel()

fun dataSnapshot.getUserModel(): User =
    this.getValue(User::class.java) ?: User()
fun sendMessage(message: String, receivingId: String, typeText:
String, function: () -> Unit) {

    val refDialogUser = "$NODE_MESSAGES/$UID/$receivingId"
    val refDialogReceivingUser =
"$NODE_MESSAGES/$receivingId/$UID"

```

```

        val messageKey =
REF_DATABASE_ROOT.child(refDialogUser).push().key

        val mapMessage = HashMap<String, Any>()

        mapMessage[CHILD_FROM] = UID
        mapMessage[CHILD_TYPE] = typeText
        mapMessage[CHILD_TEXT] = message
        mapMessage[CHILD_ID] = messageKey.toString()
        mapMessage[CHILD_TIMESTAMP] = ServerValue.TIMESTAMP

        val mapDialog = HashMap<String, Any>()

        mapDialog["$refDialogUser/$messageKey"] = mapMessage
        mapDialog["$refDialogReceivingUser/$messageKey"] =
mapMessage

REF_DATABASE_ROOT.updateChildren(mapDialog).addOnSuccessListener
{
    function()
}
    .addOnFailureListener { showToast(it.message.toString())
}
}

fun updateCurrentUsername(newUserName: String) {

REF_DATABASE_ROOT.child(NODE_USERS).child(UID).child(CHILD_USERN
AME).setValue(newUserName)
    .addOnCompleteListener {
        if (it.isSuccessful) {
            showToast("Обновлено")
            deleteOldUsername(newUserName)
        } else {
            showToast(it.exception?.message.toString())
        }
    }
}

private fun deleteOldUsername(newUserName: String) {

REF_DATABASE_ROOT.child(NODE_USERNAMES).child(USER.username).rem
oveValue()
    .addOnSuccessListener {
        showToast("Обновлено")
        ACTIVITY.supportFragmentManager.popBackStack()
        USER.username = newUserName
    }.addOnFailureListener {

```

```

showToast(it.message.toString()) }
}

fun pushBioToDatabase(newBio: String) {

REF_DATABASE_ROOT.child(NODE_USERS).child(UID).child(CHILD_BIO).
setValue(newBio)
    .addOnSuccessListener {
        showToast("ДАННЫЕ ОБНОВЛЕНЫ")
        USER.bio = newBio
        ACTIVITY.supportFragmentManager.popBackStack()

    }.addOnFailureListener {
showToast(it.message.toString()) }
}

fun pushFullnameToDatabase(fullname: String) {

REF_DATABASE_ROOT.child(NODE_USERS).child(UID).child(CHILD_FULLN
AME).setValue(fullname)
    .addOnSuccessListener {
        showToast("УСПЕШНО ОБНОВЛЕНО")

        USER.fullname = fullname
        ACTIVITY.mAppDrawer.updateHeader()
        ACTIVITY.supportFragmentManager.popBackStack()

    }.addOnFailureListener {
showToast(it.message.toString()) }
}

fun sendFile(
    receivingId: String,
    fileUrl: String,
    messageKey: String,
    messageType: String,
    filename: String
) {

    val refDialogUser = "$NODE_MESSAGES/$UID/$receivingId"
    val refDialogReceivingUser =
"$NODE_MESSAGES/$receivingId/$UID"

    val mapMessage = HashMap<String, Any>()

    mapMessage[CHILD_FROM] = UID
    mapMessage[CHILD_TYPE] = messageType

```

```

        mapMessage[CHILD_ID] = messageKey
        mapMessage[CHILD_TIMESTAMP] = ServerValue.TIMESTAMP
        mapMessage[CHILD_FILE_URL] = fileUrl
        mapMessage[CHILD_TEXT] = filename

        val mapDialog = HashMap<String, Any>()

        mapDialog["$refDialogUser/$messageKey"] = mapMessage
        mapDialog["$refDialogReceivingUser/$messageKey"] =
mapMessage

        REF_DATABASE_ROOT.updateChildren(mapDialog)
            .addOnFailureListener { showToast(it.message.toString())
    }

    }

fun getMessageKey(id: String) =

REF_DATABASE_ROOT.child(NODE_MESSAGES).child(UID).child(id).push
().key.toString()

fun uploadFileToStorage(
    uri: Uri,
    messageKey: String,
    receivedID: String,

    messageType: String,
    filename: String = ""
) {

    val path =
REF_STORAGE_ROOT.child(FOLDER_FILES).child(messageKey)

    putFileToStorage(uri, path) {
        getUrlStorage(path) {
            sendFile(receivedID, it, messageKey, messageType,
filename)

        }
    }
}

fun getFileFromStorage(mFile: File, fileUrl: String, function:
() -> Unit) {
    val path =
REF_STORAGE_ROOT.storage.getReferenceFromUrl(fileUrl)
    path.getFile(mFile)
        .addOnSuccessListener {

```

```

function()

    }
    .addOnFailureListener {
        showToast(it.message.toString())
    }
}

fun saveToMainList(id: String, type: String) {

    val refUser = "$NODE_MAIN_LIST/$UID/$id"
    val refReceived = "$NODE_MAIN_LIST/$id/$UID"

    val mapUser = HashMap<String, Any>()
    val mapReceived = HashMap<String, Any>()

    mapUser[CHILD_ID] = id
    mapUser[CHILD_TYPE] = type

    mapReceived[CHILD_ID] = UID
    mapReceived[CHILD_TYPE] = type

    val defaultMap = HashMap<String, Any>()

    defaultMap[refUser] = mapUser
    defaultMap[refReceived] = mapReceived

    REF_DATABASE_ROOT.updateChildren(defaultMap)
        .addOnFailureListener { showToast(it.message.toString())
    }

}

fun deleteChat(id: String, function: () -> Unit) {

    REF_DATABASE_ROOT.child(NODE_MAIN_LIST).child(UID).child(id).removeValue()
        .addOnFailureListener { showToast(it.message.toString())
    }
        .addOnSuccessListener { function() }
    }

fun clearChat(id: String, function: () -> Unit) {
    REF_DATABASE_ROOT.child(NODE_MESSAGES).child(UID).child(id)
        .removeValue()
        .addOnFailureListener { showToast(it.message.toString())
    }
        .addOnSuccessListener {

```

```

        REF_DATABASE_ROOT.child(NODE_MESSAGES).child(id).child(UID)

            .removeValue()
            .addOnSuccessListener { function() }
            .addOnFailureListener {
showToast(it.message.toString()) }
        }

    }

fun createGroupInDatabase(
    groupName: String,
    uri: Uri,
    contactsList: List<DefaultModel>,
    function: () -> Unit
) {
    val groupKey =
REF_DATABASE_ROOT.child(NODE_GROUPS).push().key.toString()
    val path =
REF_DATABASE_ROOT.child(NODE_GROUPS).child(groupKey)
    val pathToStorage =
REF_STORAGE_ROOT.child(GROUPS_IMAGE_FOLDER).child(groupKey)

    val dataMap = hashMapOf<String, Any>()
    dataMap[CHILD_ID] = groupKey
    dataMap[CHILD_FULLNAME] = groupName
    dataMap[CHILD_PHOTO_URL] = "empty"

    val membersMap = hashMapOf<String, Any>()
    contactsList.forEach {
        membersMap[it.id] = USER_MEMBER
    }
    membersMap[UID] = USER_CREATOR

    dataMap[NODE_MEMBERS] = membersMap

    path.updateChildren(dataMap)
        .addOnSuccessListener {
            function()
            if (uri != Uri.EMPTY) {
                putFileToStorage(uri, pathToStorage) {
                    getUrlStorage(pathToStorage) {

                        path.child(CHILD_PHOTO_URL).setValue(it)

                        addGroupToChats(dataMap, contactsList) {
                            function()
                        }
                    }
                }
            }
        } else {
            addGroupToChats(dataMap, contactsList) {

```

```

        function()

            }

        }

    }
    .addOnFailureListener {
        showToast(it.message.toString())
    }

}

fun addGroupToChats(
    dataMap: HashMap<String, Any>,
    contactsList: List<DefaultModel>,
    function: () -> Unit
) {

    val path = REF_DATABASE_ROOT.child(NODE_MAIN_LIST)
    val map = hashMapOf<String, Any>()

    map[CHILD_ID] = dataMap[CHILD_ID].toString()
    map[CHILD_TYPE] = TYPE_GROUP

    contactsList.forEach {

path.child(it.id).child(map[CHILD_ID].toString()).updateChildren

    (map)
    }

    path.child(UID).child(map[CHILD_ID].toString()).updateChildren(m
    ap)
        .addOnSuccessListener { function() }
        .addOnFailureListener { showToast(it.message.toString())
    }
    }

fun sendMessageToGroup(message: String, groupID: String,
typeText: String, function: () -> Unit) {

    var refMessages = "$NODE_GROUPS/$groupID/$NODE_MESSAGES"
    val messageKey =
REF_DATABASE_ROOT.child(refMessages).push().key

    val mapMessage = HashMap<String, Any>()

    mapMessage[CHILD_FROM] = UID

```

```

        mapMessage[CHILD_TYPE] = typeText
        mapMessage[CHILD_TEXT] = message
        mapMessage[CHILD_ID] = messageKey.toString()
        mapMessage[CHILD_TIMESTAMP] = ServerValue.TIMESTAMP

    REF_DATABASE_ROOT.child(refMessages).child(messageKey.toString())
    ).updateChildren(mapMessage).addOnSuccessListener {
        function()

    }

    .addOnFailureListener { showToast(it.message.toString())
}
}

```

Файл databaseConsts.kt

```

package com.example.massmess.allPackage.database

import com.example.massmess.allPackage.models.User
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.DatabaseReference
import com.google.firebase.storage.StorageReference

lateinit var AUTH: FirebaseAuth
lateinit var UID: String

lateinit var REF_DATABASE_ROOT: DatabaseReference
lateinit var REF_STORAGE_ROOT: StorageReference
lateinit var USER: User
const val TYPE_TEXT = "text"
const val NODE_PHONES = "phones"
const val NODE_PHONES_CONTACTS = "phones_contacts"
const val NODE_USERS = "users"
const val NODE_USERNAME = "usernames"
const val NODE_MESSAGES = "messages"
const val CHILD_ID = "id"
const val CHILD_PHONE = "phone"
const val CHILD_USERNAME = "username"
const val CHILD_FULLNAME = "fullname"
const val CHILD_BIO = "bio"
const val CHILD_PHOTO_URL = "photoUrl"
const val CHILD_FILE_URL = "fileUrl"
const val CHILD_STATE = "state"
const val CHILD_TEXT = "messageText"
const val CHILD_TYPE = "messageType"
const val CHILD_FROM = "from"
const val CHILD_TIMESTAMP = "timeStamp"
const val FOLDER_PROFILE_IMAGE = "profile_image"
const val FOLDER_FILES = "messages_files"

```

```

const val NODE_MAIN_LIST = "main_list"
const val NODE_MEMBERS = "members"

const val NODE_GROUPS = "groups"
const val GROUPS_IMAGE_FOLDER = "groups_images"
const val USER_CREATOR = "creator"
const val USER_ADMIN = "admin"
const val USER_MEMBER = "member"

```

					ПГУ 1.090301.33.001 ПЗ	Лист
						117
Изм.	Лист	№ докум	Подпись	Дата		

Приложение В

(Обязательное)

Корпоративный мессенджер под ОС Android

Презентация

					ПГУ 1.090301.33.001 ПЗ	Лист
						118
Изм.	Лист	№ докум	Подпись	Дата		

Тема ВКР: «Корпоративный мессенджер под ОС Android»

Автор работы студент группы 19БВП2 Субботкин М.В.
Научный руководитель к.т.н., доцент каф. ВТ ПГУ Юрова О.В.

Пенза 2023

Цели и объекты работы

ЦЕЛЬЮ РАБОТЫ ЯВЛЯЕТСЯ РАЗРАБОТКА ПРИЛОЖЕНИЯ ПОД УПРАВЛЕНИЕМ ОС ANDROID , ПОЗВОЛЯЮЩАЯ ОБЩАТЬСЯ ПОЛЬЗОВАТЕЛЯМ В ПРЕДЕЛАХ ОДНОЙ КОМПАНИИ

ОБЪЕКТОМ РАБОТЫ ЯВЛЯЕТСЯ ПРИЛОЖЕНИЕ, ОБЛАДАЮЩЕЕ СЛЕДУЮЩИМИ ВОЗМОЖНОСТЯМИ

- РЕГИСТРАЦИЯ И АВТОРИЗАЦИЯ ПОЛЬЗОВАТЕЛЯ
- ИЗМЕНЕНИЕ НАСТРОЕК ПОЛЬЗОВАТЕЛЯ
- ОТПРАВКА СООБЩЕНИЙ ПОЛЬЗОВАТЕЛЮ ИЗ СПИСКА КОНТАКТОВ
- СОЗДАНИЕ МНОГОПОЛЬЗОВАТЕЛЬСКОГО ЧАТА
- ОТПРАВКА ФАЙЛОВ
- ОТПРАВКА ГОЛОСОВЫХ СООБЩЕНИЙ

Актуальность

Современный мир требует от предприятий и компаний быстрой и эффективной коммуникации между сотрудниками. В связи с этим, все большую популярность набирают корпоративные мессенджеры, которые позволяют ускорить и улучшить внутреннюю коммуникацию в организации.

Существует множество инструментов, которые предназначены для обеспечения внутренней коммуникации между сотрудниками, такие как электронная почта, голосовая и видеосвязь, а также корпоративные социальные сети. Но наиболее популярным и удобным способом обмена сообщениями являются мессенджеры

Аналоги



Стек технологий



Диаграмма прецедентов

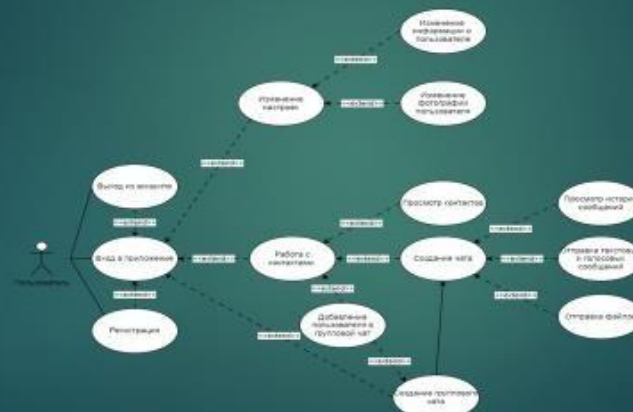
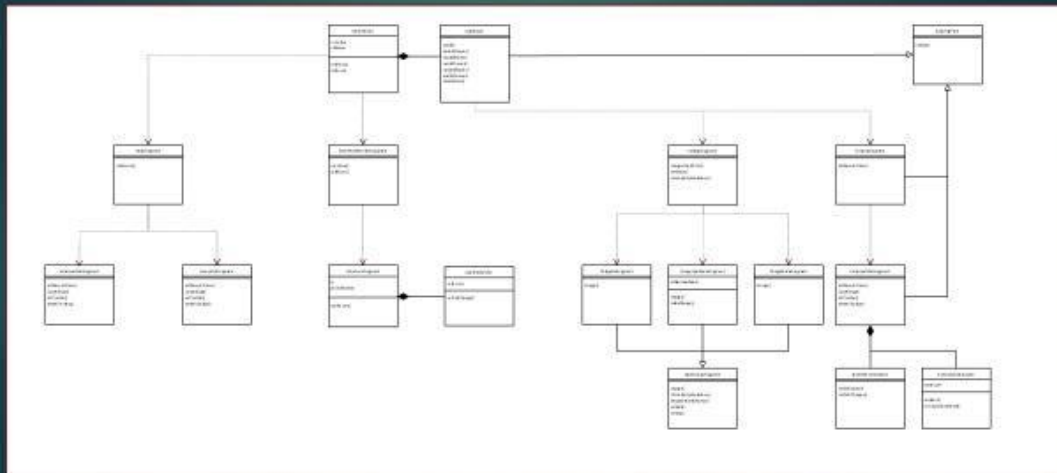


Диаграмма классов



Демонстрация приложения

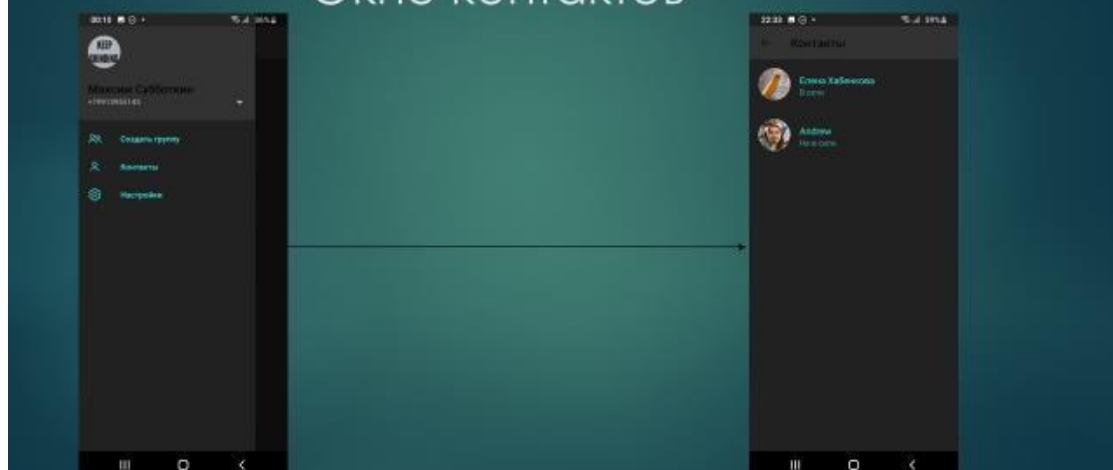
Главное окно



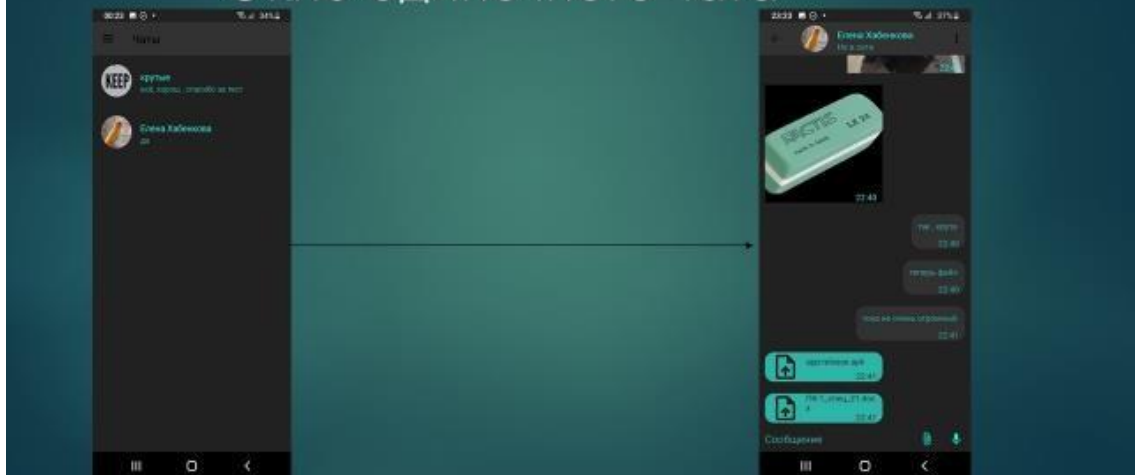
Демонстрация приложения Окно настроек



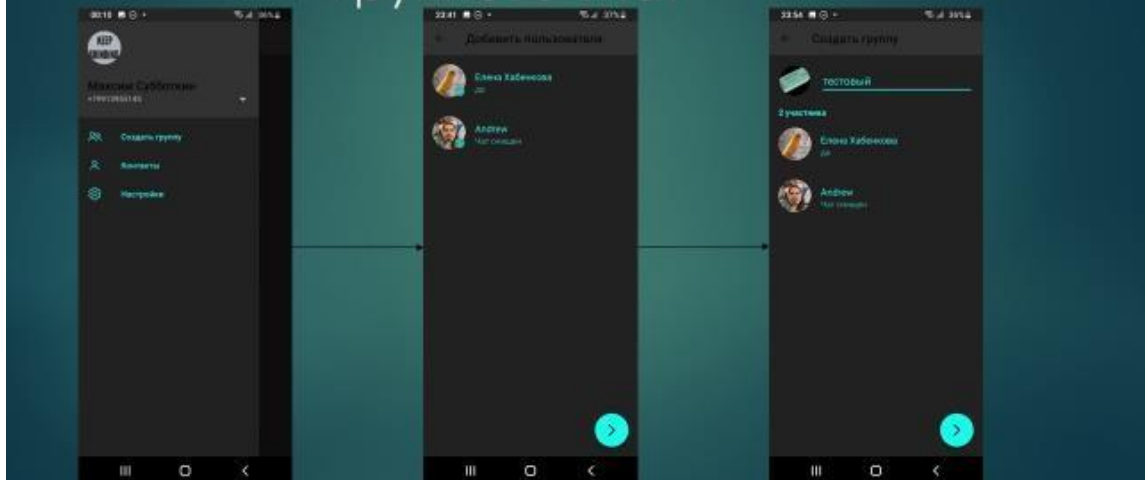
Демонстрация приложения Окно контактов



Демонстрация приложения Окно одиночного чата



Демонстрация приложения Групповой чат



Заключение

В результате выполнения данной выпускной квалификационной работы было создано мобильное приложение с удобным и понятным пользовательским интерфейсом. Созданный проект дает возможность пользователям общаться в пределах одной компании.

Разработка корпоративного мессенджера может повысить эффективность и оперативность работы в компании любой сферы деятельности.

Спасибо за внимание!

					ПГУ 1.090301.33.001 ПЗ	Лист
						125
Изм.	Лист	№ докум	Подпись	Дата		