

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 ВТ и автоматизированных систем»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
БАКАЛАВРА**

на тему

**Клиент обмена текстовыми и медиа
сообщениями для мобильной платформы
«Аврора»**

Студент

 03.06.2020
(подпись, дата)

Родионов Владислав Сергеевич

Руководитель

 03.06.2020
(подпись, дата)

Дубравин А.В.

Нормоконтролёр


(подпись, дата)

А.В. Кучин

Работа допущена к защите (протокол зас. кафедры от 03.06.2020г. № 12)

Заведующий кафедрой


(подпись)

М.А. Митрохин

Работа защищена с отметкой
11.06.2020 № 2



(протокол заседания ГЭК от

Секретарь ГЭК


(подпись)

Е.И. Гурин

Пенза 2020

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«30» апреля 2020г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Родионов Владислав Сергеевич гр.16BB2 факультета вычислительной техники.
2. Тема работы: «Клиент обмена текстовыми и медиа сообщениями для мобильной платформы «Аврора»»

Тема утверждена приказом ПГУ № 0335а/о от "30" апреля 2020г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры «ВТ» Дубравин А.В.

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

Разработать и отладить приложение «Клиент обмена текстовыми и медиа сообщениями для мобильной платформы «Аврора»». Приложение должно содержать следующий функционал: авторизация пользователей в общей системе, обмен текстовыми данными между двумя и более пользователями, которые зарегистрированы в общей системе, обмен аудио и видео данными между двумя и более пользователями, ранее зарегистрированными в общей системе.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить концепцию систем мгновенного обмена данными, изучить целевую платформу для приложения, изучить архитектурные вопросы и проблемы приложений подобного типа, изучить механизмы взаимодействия с сервером по протоколам TCP/IP и WebSocket. Необходимо изучить методологии проектирования приложений для мобильных операционных систем, а также изучить функционал фреймворка Qt.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	10	30.04.2020 – 03.05.2020	<i>Дубравин</i>
2.	Анализ требований	10	03.05.2020 – 05.05.2020	<i>Дубравин</i>
3.	Изучение технических средств решения поставленной задачи	15	05.05.2020 – 08.05.2020	<i>Дубравин</i>
4.	Разработка и тестирование приложения	45	08.05.2020 – 31.05.2020	<i>Дубравин</i>
5.	Оформление пояснительной записки	10	31.05.2020 – 03.06.2020	<i>Дубравин</i>

Дата выдачи задания " 30 " апреля 2020 г.

Руководитель бакалаврской работы *Дубравин* 30.04.2020 Дубравин А.В.
(подпись, дата)

Задание к исполнению принял студент *[подпись]* 30.04.2020 Родионов В.С.
(подпись, дата)

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

РЕФЕРАТ

Бакалаврская работа содержит 99 листов, 41 рисунок, 2 таблицы, 12 источников, 5 приложений.

МЕССЕНДЖЕР, IMS-КЛИЕНТ, АВРОРА, СИСТЕМА МГНОВЕННОГО ОБМЕНА СООБЩЕНИЯМИ, КЛИЕНТ-СЕРВЕР, TCP/IP, WEBSOCKET.

Цель работы – разработать клиент, позволяющий пользователям мгновенно обмениваться данными.

Объектом разработки является приложение, обеспечивающее выполнение следующих функций: авторизацию пользователя в системе, обмен текстовыми данными между двумя и более пользователями, которые зарегистрированы в общей системе, обмен аудио и видео данными между двумя и более пользователями, ранее зарегистрированными в общей системе. Хранение информации о пользователе и его текущих сессиях обмена должно осуществляться с помощью локальной БД. Кроме того, должно быть реализовано взаимодействие с сервером по протоколам TCP/IP и WebSocket.

В результате проведенной работы разработано приложение, удовлетворяющее поставленной задаче, а именно: разработана архитектура приложения, разработана архитектура локальной БД, реализованы протоколы взаимодействия с сервером. Приложение успешно протестировано и готово к использованию.

					ПГУ 2.090301.31.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Родионов В.С.		23.06.2025	Клиент обмена текстовыми и мультимедийными сообщениями для мобильной платформы «Аврора» Пояснительная записка		
Провер.		Дубравин А.В.		17.06.2025			
Н. Контр.		Кучин А.В.					
Утверд.		Митрохин М.А.					
					Лит.	Лист	Листов
						4	99
					ФВТ, каф. ВТ		
					гр. 16ВВ2		

Содержание

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ.....	7
ВВЕДЕНИЕ.....	8
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	9
1.1 Аналитический обзор систем мгновенного обмена данными.....	9
1.1.1.1 Текущее распространение и применение систем мгновенного обмена данными в России и мире.....	11
1.1.1.2 Анализ основных требований к системе мгновенного обмена данными.....	13
1.2 Анализ уже существующих систем мгновенного обмена данными для целевой платформы.....	14
1.2.1 Tweetian.....	15
1.2.2 Kat.....	17
1.2.3 Depecher.....	18
1.2.4 Whatsup.....	19
1.3 Исследование технических средств разработки ПО для платформы «Аврора».....	20
1.3.1 Изучение среды разработки Sailfish SDK.....	20
1.3.2 Изучение средств отладки ПО для платформы «Аврора».....	22
1.4 Вывод.....	23
2 РЕАЛИЗАЦИЯ.....	25
2.1 Описание QML-компонентов.....	26
2.1.1 Экран авторизации.....	26
2.1.2 Экран списка активных взаимодействий пользователя.....	27
2.1.3 Экран списка возможных контактов.....	28
2.1.4 Экран поиска.....	29
2.1.5 Экран создания новой группы.....	29
2.1.6 Экран настроек личного аккаунта пользователя.....	30
2.1.7 Экран взаимодействия.....	31
2.1.8 Экран деталей взаимодействия.....	31
2.2 Описание структуры БД.....	33
2.3 Описание механизмов передачи данных с помощью протоколов TCP/IP и WebSocket.....	34
2.4 Описание C++ классов.....	37
2.5 Вывод.....	40
3 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ.....	42
3.1 Авторизация в системе.....	42
3.2 Экран списка активных взаимодействий.....	43

3.3Экран настроек профиля.....	45
3.4Экран создания новой группы.....	47
3.5Экран поиска.....	50
3.6Экран возможных контактов.....	51
3.7Экран взаимодействия с пользователем или группой пользователей.....	52
3.8Экран свойств взаимодействия.....	56
3.9Вывод.....	59
4 БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ.....	60
ЗАКЛЮЧЕНИЕ.....	62
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	63
ПРИЛОЖЕНИЯ.....	65
Приложение А Листинг класса DBManager.....	65
Приложение В Листинг класса RestApi.....	82
Приложение С Листинг класса «QMLRestApi».....	86
Приложение D Листинг класса «HttpMultipart».....	88
Приложение С Презентация.....	93

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

HTTP (англ. HyperText Transfer Protocol) – протокол передачи гипертекста;

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

IMS (англ. Instant Messaging Service) – служба мгновенных сообщений;

IRC (англ. Internet Relay Chat) – протокол общения в реальном времени посредством текста;

REST (англ. Representational State Transfer) – передача состояния представления;

SMS (англ. Short Message Service) – служба коротких сообщений;

VoIP (англ. Voice over Internet Protocol) – голосовая связь через Интернет;

БД – база данных;

ГОСТ – региональный стандарт, принятый Межгосударственным советом по стандартизации, метрологии и сертификации Содружества Независимых Государств;

ОС – операционная система;

ПЭВМ – персональная электронно-вычислительная машина;

РФ – Российская Федерация;

СУБД – система управления базами данных;

США – Соединённые Штаты Америки.

ВВЕДЕНИЕ

В настоящее время развитие систем мгновенного обмена данными в сфере информационных технологий тесно связано с общественной жизнью, так как их использование значительно облегчает взаимодействие между людьми в любой сфере общественных отношений. Это достигается за счёт обеспечения возможности удалённого взаимодействия между большим кругом лиц, что позволяет оперативно решать самые разнообразные вопросы. Одним из актуальных вопросов развития данной предметной области является внедрение подобных систем в различные мобильные устройства, в частности, смартфоны, что позволит повысить значимость и актуальность предметной области.

К сожалению, в Российской Федерации данное направление развития практически не реализовано и не конкурентоспособно. На территории РФ есть компании, занимающиеся выпуском мобильных устройств, а также разработана отечественная мобильная операционная система «Аврора» [1], однако текущие разработки в сфере систем мгновенного обмена данными для платформы «Аврора» обладают существенным набором недостатков, которые не позволяют им конкурировать с зарубежными аналогами. Таким образом, актуальной задачей является разработка современной системы мгновенного обмена данными, которая отвечает всем актуальным требованиям рынка.

В данной работе будет рассмотрена реализация приложения «Клиент обмена текстовыми и медиа сообщениями для мобильной платформы «Аврора»». Данное приложение направлено на обеспечение взаимодействия между двумя и более пользователями в рамках общей системы. Также будет произведен сравнительный анализ уже существующих систем мгновенного обмена сообщениями для указанной целевой платформы, выявлены их преимущества и недостатки.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

При разработке программных продуктов необходимо провести анализ существующих вариантов для решения задачи. Данный раздел включает в себя обзор систем мгновенного обмена данными: требования к подобным системам и их применение. Производится исследование и анализ доступных средств разработки и отладки для целевой мобильной платформы.

Аналитический обзор систем мгновенного обмена данными

Система мгновенного обмена данными в настоящее время является одним из наиболее широко используемых средств взаимодействия между людьми как в глобальной сети «Интернет», так и в корпоративных, а также локальных сетях [2].

Данное направление развития информационных технологий существует с мая 1997 года, когда специалисты американской компании «America Online» разработали и внедрили сервис под названием «AOL Instant Messenger», который сразу начал пользоваться огромной популярностью, так как серьёзно упрощал взаимодействие между людьми [3].

С тех пор данная область сетевой жизни людей не только приобретает всё большую и большую востребованность и популярность, но и динамично развивается, стараясь удовлетворить современные запросы и потребности пользователей.

Глобально все системы мгновенного обмена данными можно разделить на две категории: службы IRC, и службы IMS.

Служба IRC (Или «Чат») является первым в мире средством для мгновенного общения в сети «Интернет» или локальных сетях. Она предоставляет большую площадку для проведения дискуссий с единомышленниками. Чат является исключительно текстовым диалогом в режиме реального времени.

Данная служба базируется на сетевой архитектуре «клиент-сервер». Это означает, что потенциальным пользователям подобных служб необходимо установить на своё устройство специальный клиент, разработанный для взаимодействия с IRC-сетью (IRC-клиент). При запуске подобного IRC-клиента, он устанавливает связь с IRC-сервером средствами сетевых протоколов взаимодействия, после чего пользователь может выбрать контакт или целую группу контактов для общения [4].

С развитием данного направления информационных технологий появилось множество IRC-сетей, которые абсолютно независимы друг от друга и имеют свои доменные имена (DALnet, IRCnet, UNDERnet, RusNet, WeNet, IrcNet.ru и т.д.). С помощью глобальной сети «Интернет» можно найти множество IRC-клиентов, которые предназначены как для Unix-подобных ОС, так и для Windows-систем, а также мобильных устройств.

В качестве отдельного подвида IRC-службы необходимо выделить Web-чаты. Такие сервисы предназначены для мгновенного обмена данными на сервере с помощью браузера, поэтому пользователю нет необходимости устанавливать IRC-клиент на своё устройство. Web-чат представляет собой интернет-страницу, зайдя на которую можно обмениваться текстовыми сообщениями с другими посетителями.

Результатом развития служб IRC стали службы IMS. Основные архитектурные решения службы IMS напрямую унаследовали у служб IRC, в частности, клиент-серверную архитектуру взаимодействия.

В большинстве случаев, IMS-сети имеют отдельные сервера, хотя встречаются и децентрализованные сети, использующие, например, протокол «Jabber» [5], и свои протоколы взаимодействия, которые являются закрытыми или проприетарными.

Так же, как и у служб IRC, службы IMS имеют свои IMS-клиенты, за которыми закрепилось название «Мессенджер», хотя есть службы IMS, которые реализовали Web-версию доступа к своей сети.

С развитием данных технологий и проектов они стали предоставлять своим пользователям множество дополнительных полезных функций: IP-телефония, видеочаты, индикацию о сетевом статусе пользователей и множество других полезных и удобных инструментов взаимодействия.

Список основного функционала, который могут предоставлять современные системы мгновенного обмена данными:

- чат (видеочат, текстовый и голосовой);
- VoIP сервисы: звонки на компьютер, звонки на стационарные и мобильные телефоны;
- возможность отправки SMS;
- передача файлов;
- инструменты для совместной работы в режиме реального времени;
- возможность общаться в чате непосредственно на веб-странице;
- напоминания и оповещения;
- хранение истории общения по каждому контакту;
- индикация о сетевом статусе пользователей (в сети, нет на месте и т.д.), занесенных в список контактов.

Таким образом, можно заключить, что системы мгновенного обмена данными, появившись в 1997 году, совершили огромный скачок в развитии, предоставляя актуальный современный функционал, который может облегчить социальную жизнь своего пользователя. Полезность и удобство использования привели к широкому распространению систем мгновенного обмена данными по всему миру.

Текущее распространение и применение систем мгновенного обмена данными в России и мире

В настоящее время использование различных систем мгновенного обмена данными прочно вошло в образ жизни современного человека, что обусловлено широким функционалом и удобством использования.

Подобные системы применяются для самых широких целей: организация личного общения и досуга, организация корпоративного взаимодействия, осуществление образовательной деятельности, осуществление экономической деятельности и многое другое.

Согласно актуальным данным в США самой популярной системой мгновенного обмена данными является «Facebook Messenger». Этой системой активно пользуется 73% населения страны в возрасте от 18 до 29 лет. Это означает аудиторию примерно в 170 миллионов человек из 320 миллионов жителей страны [6]. Всего в мире «Facebook Messenger» насчитывает около миллиарда пользователей. Данная система входит в состав социальной сети «Facebook», которой пользуются 2.5 миллиарда человек по всему миру.

В Китае среди подобных систем лидирует «WeChat» [7], поддерживая мгновенное взаимодействие для 1,057 миллиарда пользователей согласно статистике китайской компании «Tencent».

Больше половины жителей Ирана используют систему мгновенного обмена данными «Telegram» [8]. Аудитория данной системы в стране превышает 40 миллионов человек. Всего же в мире системой «Telegram» пользуются 400 миллионов человек.

В Южной Корее разработали свою узкоспециализированную систему мгновенного обмена сообщениями под названием «KakaoTalk» [9]. Несмотря на малую популярность в мире, в самой Южной Корее этой системой пользуется более 80% населения, а именно 42 миллиона человек.

Во всём остальном мире безоговорочным лидером на рынке систем мгновенного обмена данными является «WhatsApp», которой пользуются 2 миллиарда человек по всему миру.

Например, в Великобритании данной системой пользуются 84% жителей в возрасте от 24 до 35 лет, а также 78% населения в возрастной группе от 35 до 44 лет. В Германии 81% всех жителей этой страны также пользуются системой «WhatsApp».

В России самой востребованной системой мгновенного обмена данными также является «WhatsApp» [10]. Согласно данным компании «Mediascope» по состоянию на декабрь 2019 года 56.3% населения РФ в возрасте от 12 лет пользуются данной системой. Второе место по популярности занимает система «Viber», которой пользуются 30.5% жителей страны, что составляет аудиторию 37.3 миллиона человек. 20.5 миллионов человек используют систему «Telegram», что составляет 16.8% жителей в возрасте от 12 лет.

Эти данные позволяют заключить, что технологическое развитие систем мгновенного обмена данными в области информационных технологий и коммуникаций действительно является актуальной задачей на сегодняшний день. Таким образом актуальной задачей является и разработка конкурентоспособной отечественной системы мгновенного обмена данными, которая соответствует всем актуальным требованиям рынка.

Анализ основных требований к системе мгновенного обмена данными

Основные требования к системам мгновенного обмена данными ясно проистекают из назначения подобных систем.

Основное и главное требование заключается в функциональных возможностях взаимодействия между пользователями или группой пользователей. Взаимодействие должно заключаться в обеспечении возможности передачи между пользователями данных различного формата: текстовые данные, произвольные пользовательские файлы, аудиоданные, видеоданные, изображения и прочее.

Также необходимым требованием к системе мгновенного обмена данными является простота и удобство использования. Выполнение этого требования должно обеспечиваться удобным и интуитивным пользовательским интерфейсом, который позволит пользователю просто и эффективно реализовывать функционал системы.

Ключевым вопросом в реализации подобной системы является безопасность личных данных пользователей. Она достигается за счёт применения современных технологий шифрования, правильного инструктажа пользователя относительно личной безопасности, а также правильной организацией доступа к внутренней базе данных приложения и внутренней логикой приложения.

Обеспечение комфортного взаимодействия пользователей является ещё одним актуальным требованием. Это возможно обеспечить следующим образом: организовать правильную и простую идентификацию пользователей между собой, например, с помощью имён-идентификаторов и возможности пользователям установить уникальное изображение-аватар, а также организовать удобное взаимодействие пользователей в группе через реализацию системы привилегий для членов одной группы.

Реализация вышеописанных требований, в сочетании с уникальным функционалом, позволит системе мгновенного обмена данными стать привлекательной для пользователей, а значит повысит конкурентноспособность разработанной системы на рынке.

Уже разработанные системы мгновенного обмена сообщениями для отечественной мобильной операционной системы «Аврора», к сожалению, имеют ряд ключевых недостатков в реализации обозначенных требований.

Анализ уже существующих систем мгновенного обмена данными для целевой платформы

На текущий момент времени платформа «Аврора» может поддерживать два пути реализации приложений: эмуляция приложений, разработанных для мобильной платформы «Android», а также установка приложений, разработанных специально для данной платформы.

Эмуляция приложений, разработанных для «Android» имеет ряд критических недостатков:

- малый модельный ряд устройств, способных к эмуляции;

— из представленного ряда устройств, способных к эмуляции приложений для «Android», существенная часть может работать только с приложениями, разработанными для «Android» версии 4.0 или ниже;

— слабая интеграция эмулированных приложений в конечную среду платформы «Аврора», что может приводить как к чисто визуальным проблемам в восприятии, так и к критическим ошибкам в работе программного обеспечения.

Совокупность этих недостатков приводит к тому, что проводить анализ данного сегмента приложений нецелесообразно с точки зрения исследования, проводящегося в рамках этой работы, поэтому анализу будут подвергнуты только те приложения, которые разработаны специально для платформы «Аврора».

В официальном магазине платформы удалось найти три приложения IMS-клиента, а также ещё одно приложение данного направления удалось найти в открытых источниках глобальной сети «Интернет»:

- «Tweetian» - IMS-клиент сети «Twitter»;
- «Kat» - клиент для популярной сети «ВКонтакте»;
- «Depecher» - клиент, осуществляющий доступ к сети «Telegram»;
- «Whatsup» - клиент доступа к IMS-сети «WhatsApp».

Tweetian

«Tweetian» представляет собой приложение-клиент популярной IMS-сети «Twitter». Изначально данное приложение разрабатывалось для мобильной платформы «Symbian» усилиями Диксона Лонга и его небольшой команды.

Затем усилиями Веса-Матти Хартикайнен и небольшой группы тестировщиков было осуществлено портирование на платформу «Аврора», после чего проект получил дополнительное развитие.

Сейчас последнюю версию приложения возможно загрузить из официального магазина платформы, однако тот факт, что последнее

обновление было загружено в 2017 году позволяет с большой долей вероятности предположить, что проект заброшен и дорабатываться не будет.

Преимущества «Tweetian»:

— удобный и интуитивный интерфейс, использующий уникальные возможности целевой платформы (основной экран приложения представлен на рисунке 1);

— реализация взаимодействия с основным функционалом IMS-сети «Twitter»;

— открытый исходный код.

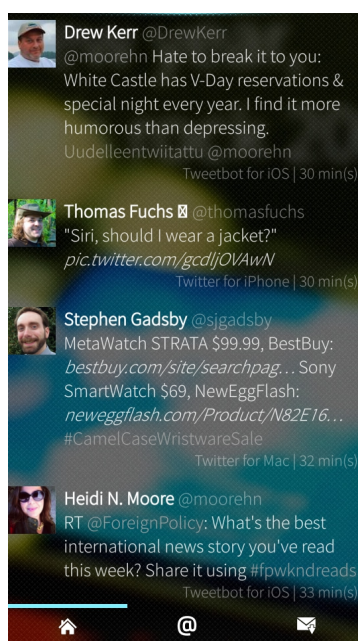


Рисунок 1 - Главный экран приложения «Tweetian»

Недостатки данного приложения:

— отсутствие возможности прикрепить к сообщению пользовательские файлы;

— отсутствие какой-либо локализации кроме англоязычной;

— отсутствие перспектив дальнейшей разработки и модернизации целенаправленными усилиями команды людей;

— некритичные ошибки интерфейса.

Kat

«Kat» является неофициальным IMS-клиентом для доступа к функционалу сети «ВКонтакте», который разрабатывается одним человеком по имени Пётр Витовтов. Данное приложение можно загрузить как с помощью официального магазина платформы «Аврора», так и с помощью официальной группы проекта в сети «ВКонтакте».

Как и «Tweetian», данный проект, скорее всего заброшен. Такой вывод можно сделать исходя из того факта, что последнее обновление в официальный магазин было выложено в 2015 году. Образец интерфейса программы представлен на рисунке 2.

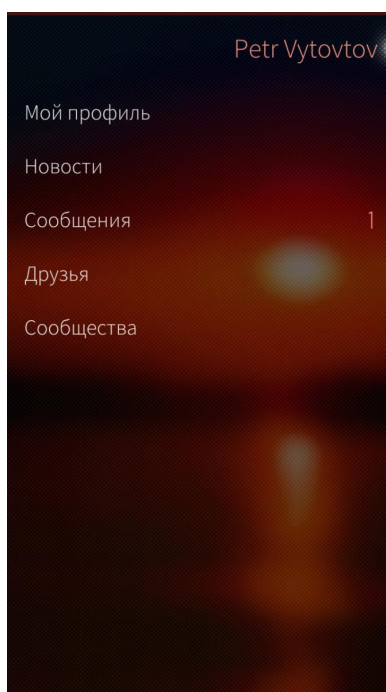


Рисунок 2 - Главный экран приложения «Kat»

Среди достоинств данного приложения можно выделить минималистичный интерфейс, разработанный специально для интеграции в среду платформы «Аврора» и возможность доступа к самому базовому функционалу представленной IMS-сети.

Недостатки заключаются в отсутствии реализации доступа к большей части функционала сети «ВКонтакте», явной недоработанности приложения,

что приводит к критичным ошибкам в работе, а также отсутствие явной поддержки приложения со стороны разработчика.

Depecher

«Depecher» это неофициальный IMS-клиент с открытым исходным кодом для доступа к функциям IMS-сети «Telegram», разрабатываемый небольшой командой людей. Данное приложение можно загрузить как из официального магазина платформы, так и с помощью открытых источников глобальной сети «Интернет». Пример интерфейса данного приложения продемонстрирован на рисунке 3.

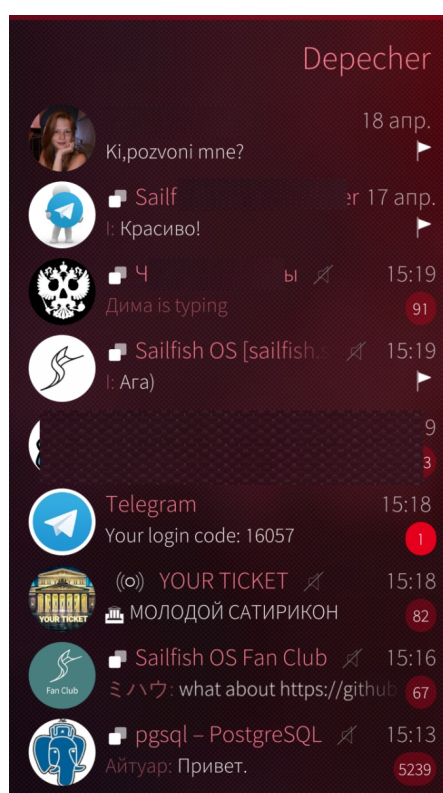


Рисунок 3 - Главный экран приложения "Depecher"

Среди других приложений данный IMS-клиент выделяется самым проработанным функционалом, реализующим доступ практически ко всем средствам взаимодействия, которые может предоставить IMS-сеть «Telegram», а также самым приятным и практичным интерфейсом.

Также немаловажно то, что последнее обновление для данного приложения было опубликовано в сентябре прошлого года, что оставляет надежды на дальнейшее развитие проекта.

Среди недостатков можно отметить отдельные неинтуитивные моменты в реализации интерфейса, отсутствие поддержки незначительной части функционала исходной IMS-сети, а также отсутствие какой-либо локализации, кроме англоязычной. Однако ключевым недостатком данного приложения является, на текущий момент времени, невозможность его использования на территории Российской Федерации в связи с принятием 13 апреля 2018 года Таганским судом Москвы решения в пользу Роскомнадзора, которое позволило с 16 апреля 2018 года блокировать использование данной IMS-сети.

Однако, следует отметить, что на текущий момент времени подготовлено несколько проектов по отмене блокировки использования сети «Telegram».

Whatsup

Данное приложение представляет собой неофициальный клиент доступа к одной из самых популярных IMS-сетей «WhatsApp». Приложение на данный момент не зарегистрировано в официальном магазине платформы «Аврора», поэтому загрузить его можно только из открытых источников глобальной сети «Интернет».

Представленный клиент находится на крайне ранней стадии разработки и, скорее всего, никогда не будет завершён полностью по нескольким причинам: официальная политика IMS-сети «WhatsApp» запрещает использование сторонних клиентов для доступа к своей сети, поэтому за использование данного клиента грозит блокировка личного аккаунта пользователя в данной сети, в связи с этим разработчик забросил разработку данного приложения, опубликовав последнее обновление в 2017 году. Разработка официального клиента IMS-сети «WhatsApp», согласно открытым источникам глобальной сети «Интернет», не ведётся.

Проведённый анализ уже реализованных приложений позволяет сделать заключение о том, что разработка современного клиента для системы

мгновенного обмена сообщениями действительно является актуальной задачей для развития мобильной платформы «Аврора».

Для разработки программного обеспечения компания-разработчик данной платформы представляет удобный набор инструментов, скомпонованных в IDE на основе «Qt Creator» под названием «Sailfish OS SDK».

Исследование технических средств разработки ПО для платформы «Аврора»

Для разработки приложений для ОС «Аврора» предусмотрена только одна доступная на текущий момент времени среда разработки – «Sailfish OS SDK».

Данная IDE включает в себя все необходимые средства для разработки, тестирования, и отладки программного обеспечения для мобильной платформы «Аврора».

Изучение среды разработки Sailfish SDK

Данная среда разработки базируется на IDE «Qt Creator» и бесплатно распространяется на сайте компании-разработчика ОС «Аврора».

Последняя версия на данный момент: 3.0. Доступны варианты для установки на платформах под управлением операционных систем Linux 64, Linux 32, OS X, а также Microsoft Windows.

Для решения поставленных задач была выбрана версия для персональных компьютеров под управлением Microsoft Windows. Вес установочного файла «Sailfish OS SDK» для данной платформы составляет 1.1 Гб.

В состав SDK входят следующие компоненты:

- QtCreator. IDE, в которой предлагается вести весь процесс разработки;
- Mer Build Engine, который необходим для сборки приложений;
- Эмулятор ОС «Аврора»;

— Примеры, руководства и документация к API.

Mer Build Engine и эмулятор платформы поставляются в виде образов виртуальных машин для VirtualBox. Однако, сам VirtualBox в состав Sailfish OS SDK не входит. Поэтому, перед непосредственной установкой SDK, необходимо вначале установить VirtualBox версии не ниже чем 4.1.18. Кроме того, при работе в Microsoft Windows, перед установкой SDK так же необходимо установить пакет Windows Microsoft Visual C++ 2010 redistributable (x86).

SDK поставляется в виде графического инсталлятора, поэтому установка представляет собой интуитивно понятный и простой процесс. Главное окно инсталлятора представлено на рисунке 4.

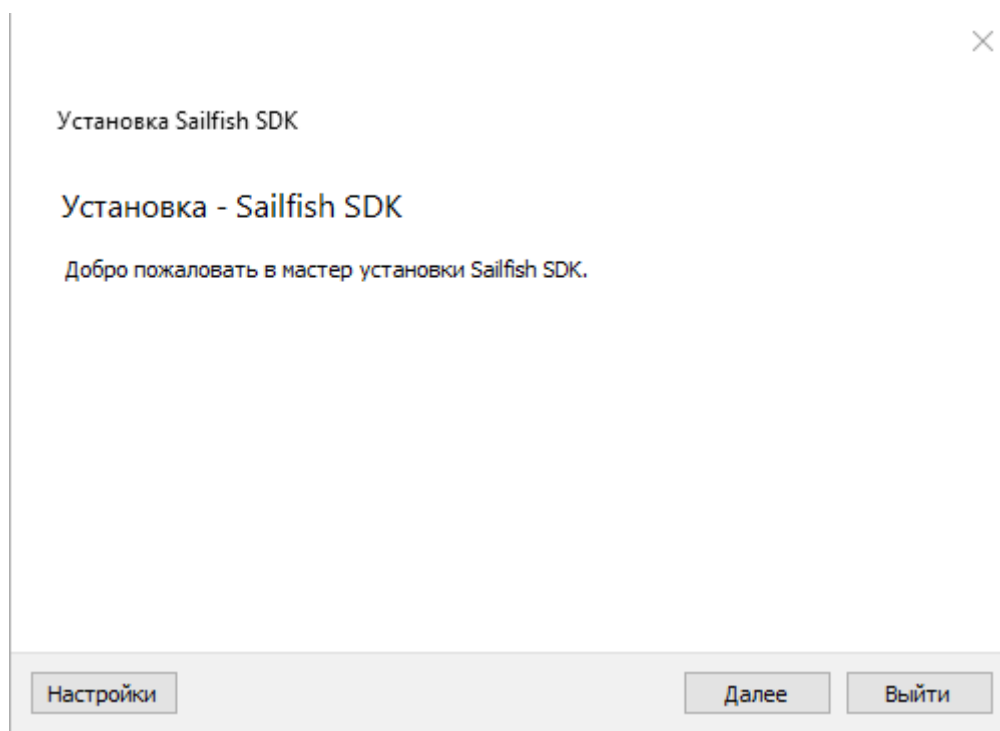


Рисунок 4 - Главное окно инсталлятора среды разработки

Главное окно редактора кода среды разработки представлено на рисунке 5.

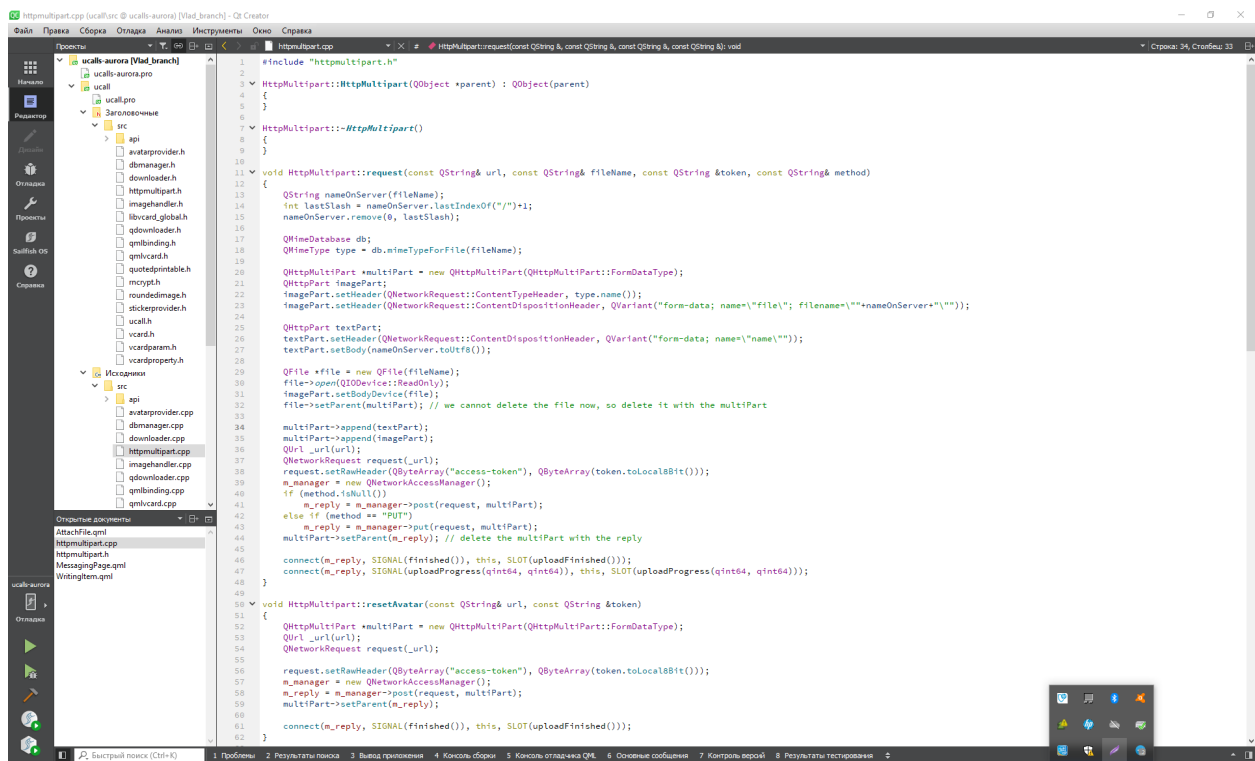


Рисунок 5 - Главное окно редактора кода среды разработки

Так как данная среда основывается на IDE «QtCreator», то она обладает всем необходимым набором функций, который доступен и в исходной среде разработки, что делает процесс реализации приложений комфортным и легко осваемым для всех, кто знаком с исходной IDE.

Развитие и поддержка среды разработки активно ведётся усилиями компании-разработчика платформы «Аврора».

В состав среды разработки «Sailfish OS SDK» входит, помимо средств разработки, достаточный объём средств тестирования и отладки разработанного программного обеспечения.

Изучение средств отладки ПО для платформы «Аврора»

Для тестирования и отладки приложения поддерживается два основных направления:

— тестирование и отладка с помощью встроенного в среду разработки эмулятора, интерфейс которого представлен на рисунке 6.

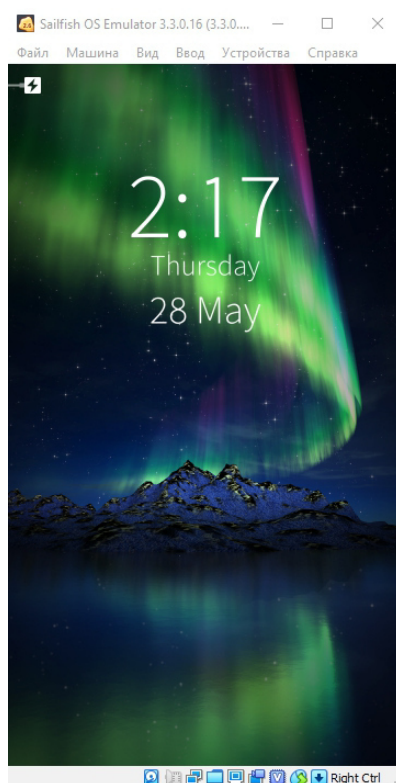


Рисунок 6 - Пример интерфейса эмулятора

— запуск приложения в режиме отладки на реальном устройстве под управлением операционной системы «Аврора».

В зависимости от текущих задач и условий разработки можно свободно переключаться между этими двумя режимами отладки с помощью удобного контекстного меню в левом нижнем углу интерфейса IDE.

В целом, данная среда разработки предоставляет исчерпывающий набор функциональных возможностей для реализации поставленной задачи.

Вывод

В данной главе было произведено исследование и анализ предметной области. В ходе проведения анализа была изучена концепция систем мгновенного обмена данными, а также установлены основные требования к реализации данного направления информационных технологий и коммуникаций.

Проведение исследования уже готовых решений для заданной целевой платформы, а также анализа распространённости использования систем

данного класса, позволили установить высокую степень актуальности задачи, поставленной для решения в рамках данной работы.

Также были проанализированы технические средства, которые будут использоваться для решения поставленной задачи. В частности, были изучены функциональные возможности IDE разработки программного обеспечения для мобильной платформы «Аврора». В ходе изучения была установлена достаточность и исчерпываемость функциональных возможностей «Sailfish OS SDK» для решения поставленной задачи.

2 РЕАЛИЗАЦИЯ

Разработку приложений для ОС «Аврора» предлагается вести с помощью трёх языков программирования:

- QML, с помощью которого описывается графический интерфейс приложения;
- JavaScript, который поддерживается QML и позволяет реализовать внутри графического интерфейса простые алгоритмы, не требующие больших затрат ресурсов устройства;
- C++ для реализации требовательных и ресурсоёмких вычислений.

Разрабатываемое приложение должно иметь удобный графический интерфейс, базироваться на архитектуре «клиент-сервер», а значит иметь возможность связи с сервером по протоколам TCP/IP и WebSocket, а также иметь локальную базу данных, которая позволит хранить личные данные пользователя.

Общая схема взаимодействия компонентов системы представлена на рисунке 7.

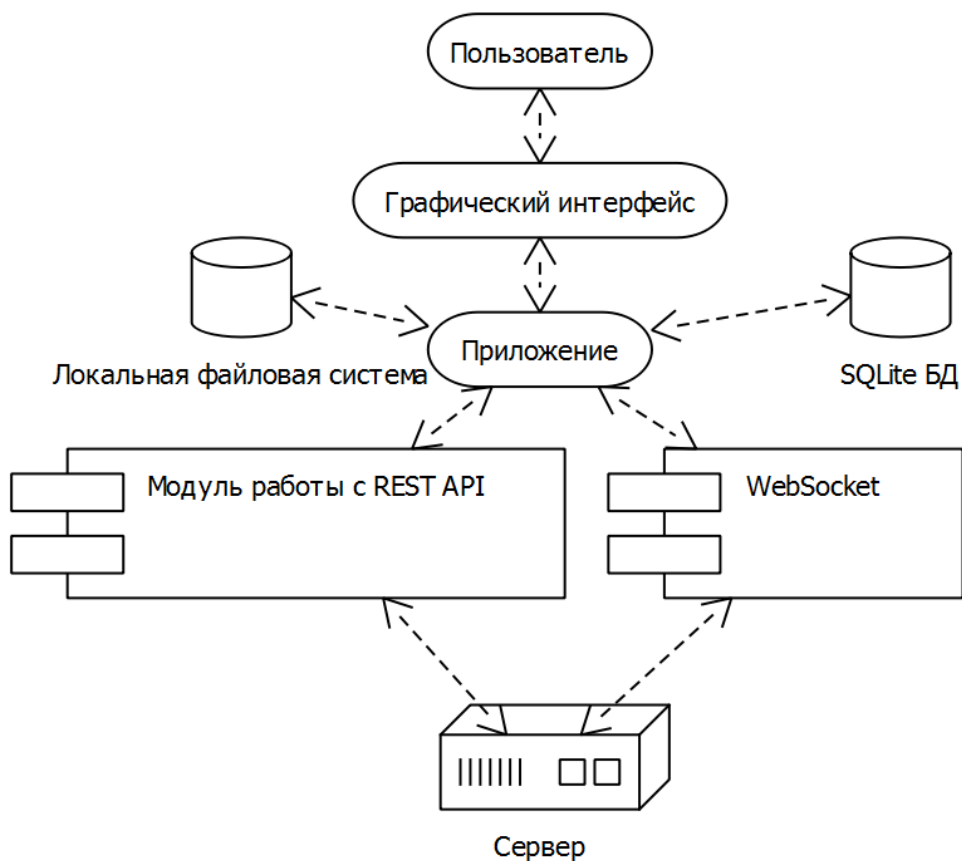


Рисунок 7 - Схема взаимодействия компонентов системы

Таким образом, разработку приложения можно разделить на следующие этапы:

- реализация QML-компонентов;
- разработка структуры БД;
- разработка механизмов передачи данных с помощью протоколов TCP/IP и WebSocket;
- разработка C++ классов для трудоёмких вычислений;

Описание QML-компонентов

QML-компоненты представляют собой графические элементы интерфейса приложения, главным образом описывая в декларативном стиле экраны приложения.

Экран авторизации

Данный экран предоставляет пользователю возможности авторизации в системе с помощью пары значений логин-пароль.

Для обеспечения безопасности учётных записей пароль не должен быть короче 6 символов.

В случае успешной авторизации пользователь перенаправляется на экран инициализации страницы со список активных взаимодействий пользователя. Схема взаимодействия QML-компонентов в рамках процесса авторизации и перехода к экрану со списком активных взаимодействий пользователя представлена на рисунке 8.

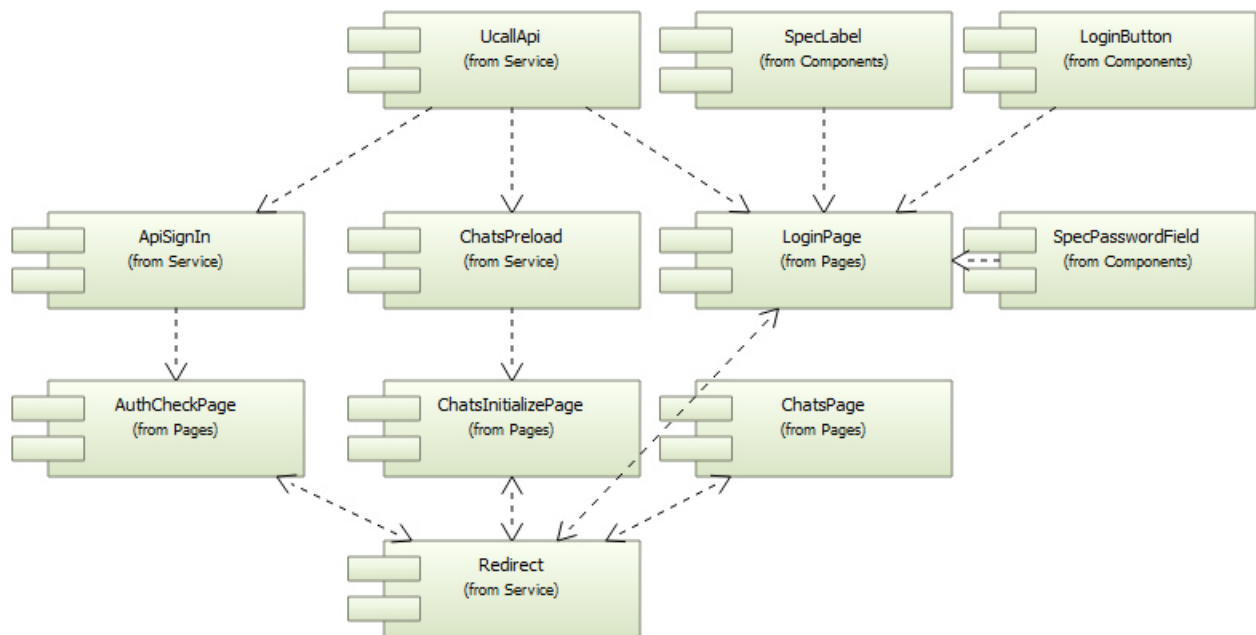


Рисунок 8 - Схема взаимодействия QML-компонентов при авторизации пользователя

Экран списка активных взаимодействий пользователя

Этот экран представляет пользователю следующий функционал:

- список всех активных взаимодействий пользователя с описанием каждого взаимодействия;
- переход к странице конкретного взаимодействия;
- выход пользователя из личного аккаунта;
- переход к странице возможных контактов пользователя;
- переход к странице поиска внутри системы;
- переход к странице создания новой группы;

— переход к активности просмотра настроек личного аккаунта пользователя.

Схема взаимодействия QML-компонентов в рамках данного экрана представлена на рисунке 9.

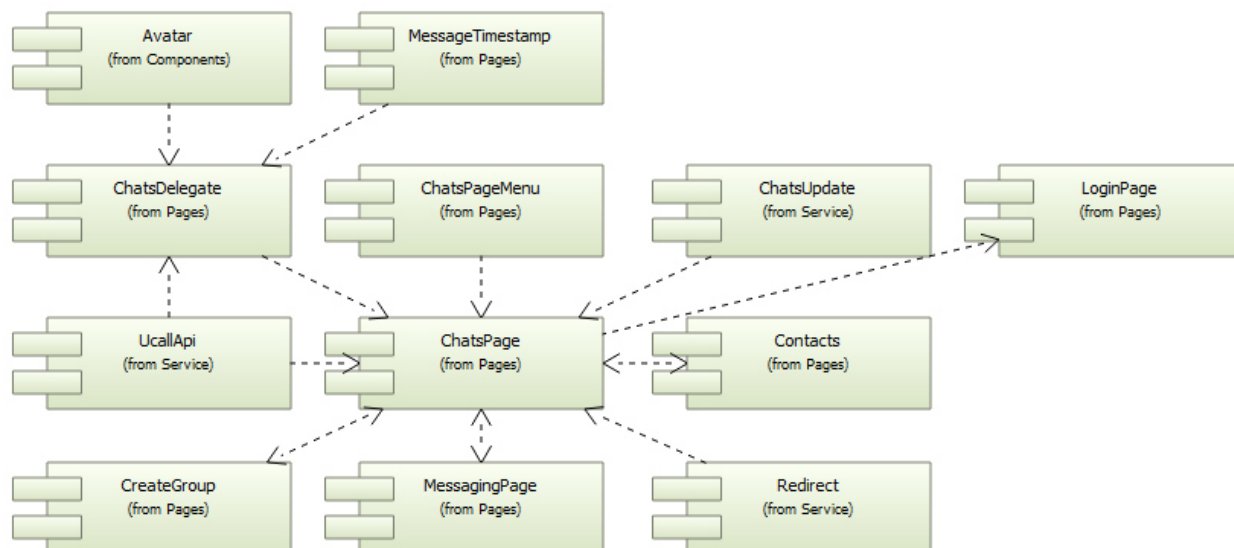


Рисунок 9 - Схема взаимодействия QML-компонентов в рамках экрана списка активных взаимодействий

Экран списка возможных контактов

Данный экран предоставляю список пользователей для организации взаимодействия в рамках системы с возможностью перехода к экрану взаимодействия.

Схема взаимодействия QML-компонентов в рамках данного экрана представлена на рисунке 10.

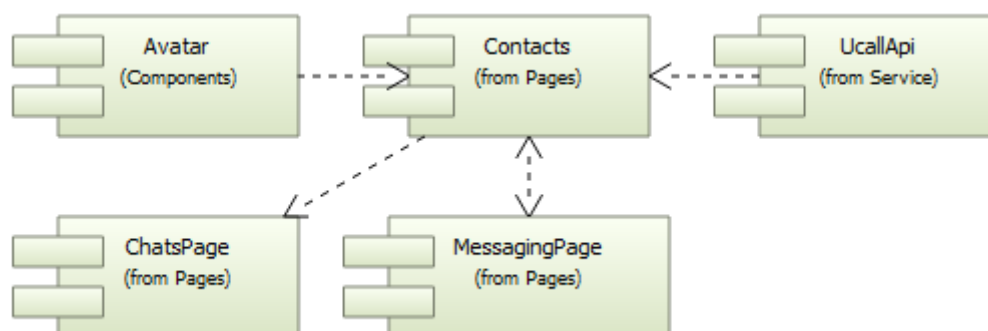


Рисунок 10 - Схема взаимодействия QML-компонентов в рамках экрана ВОЗМОЖНЫХ КОНТАКТОВ

Экран поиска

Данная активность предоставляет пользователю возможность поиска внутри системы конкретных контактов, сообщений в составе активных активностей и т.д.

Схема взаимодействия QML-компонентов в рамках поиска представлена на рисунке 11.

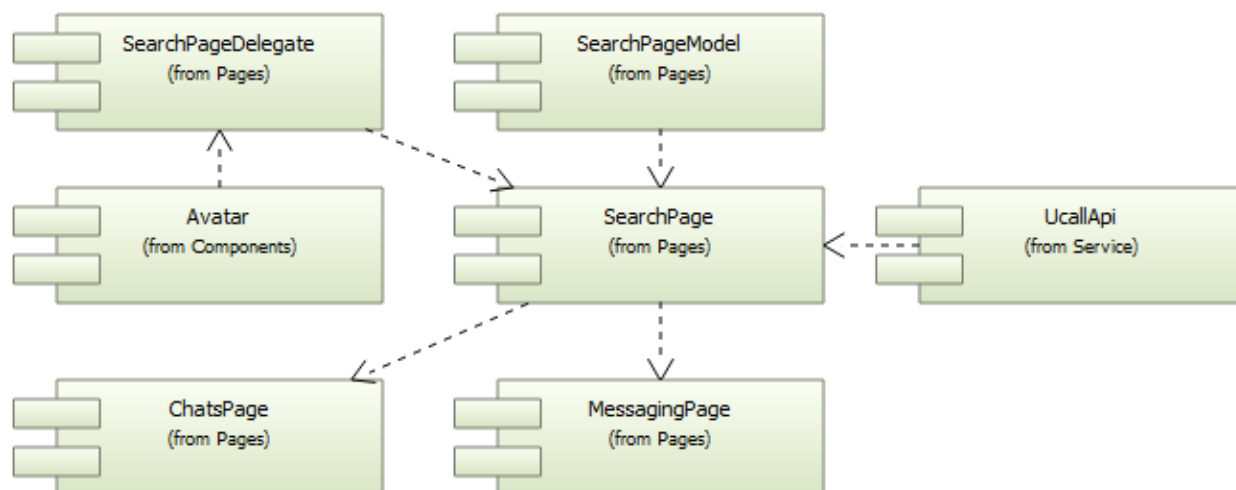


Рисунок 11 - Схема взаимодействия QML-компонентов в рамках поиска

Экран создания новой группы

Данный экран предоставляет пользователю возможность создать новую группу с заданным составом участников. Настройки группы включают в себя название, описание, и аватар.

Схема взаимодействия QML-компонентов в рамках процесса создания новой группы представлена на рисунке 12.

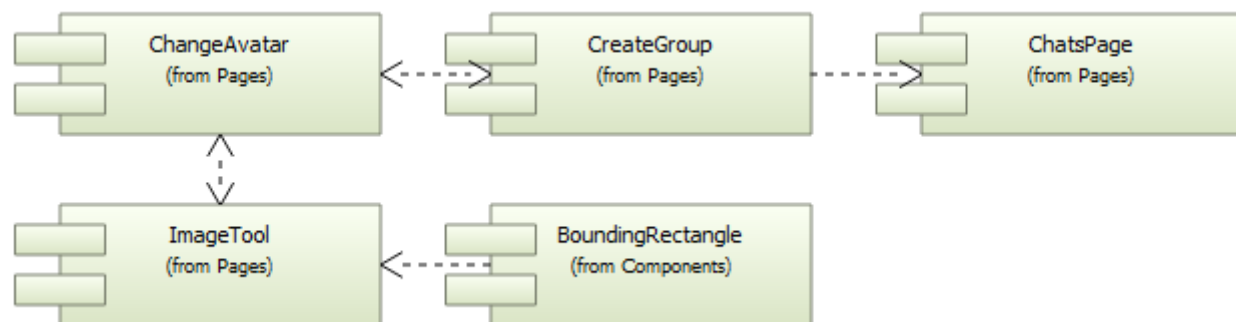


Рисунок 12 - Схема взаимодействия QML-компонентов в рамках создания новой группы

Экран настроек личного аккаунта пользователя

Данный экран предоставляет следующий функционал:

- просмотр настроек личного профиля пользователя;
- изменение описания профиля;
- изменение пароля;
- изменение личного аватара;
- просмотр списка взаимодействий, у которых отключены уведомления.

Схема взаимодействия QML-компонентов в рамках данного экрана представлена на рисунке 13.

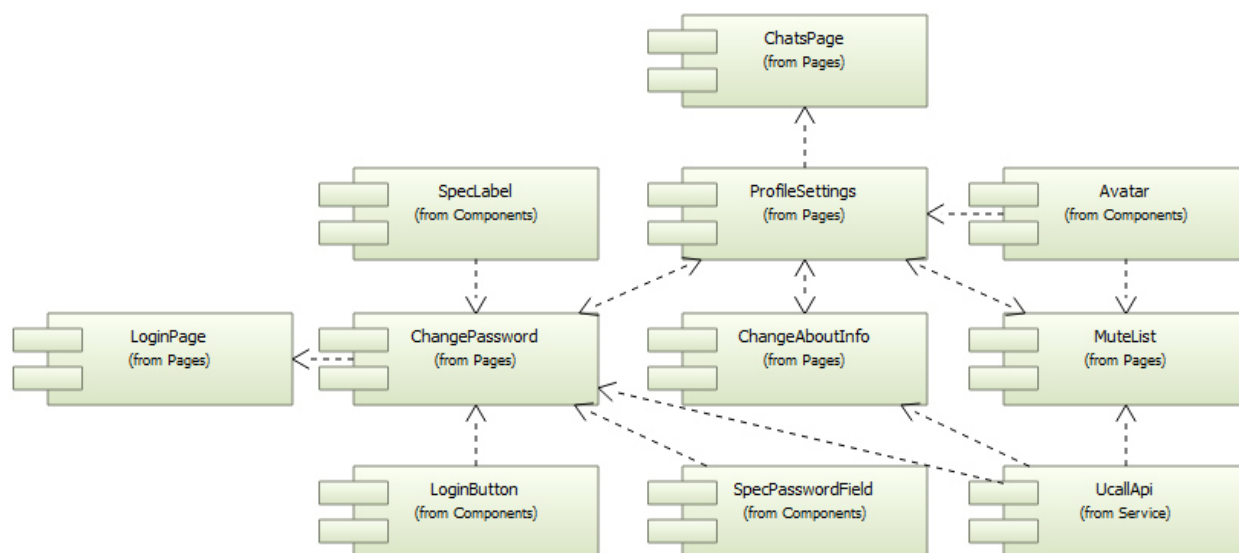


Рисунок 13 - Схема взаимодействия QML-компонентов в рамках экрана личного профиля

Экран взаимодействия

Данный экран представляет функционал взаимодействия с пользователем или списком пользователей в виде отправки и приёма данных различного типа, включая текст, изображения, видеофайлы и другое. Также предоставляется возможность перехода к экрану деталей взаимодействия.

Диаграмма связей QML-компонентов в рамках экрана взаимодействия между пользователями представлена на рисунке 14.

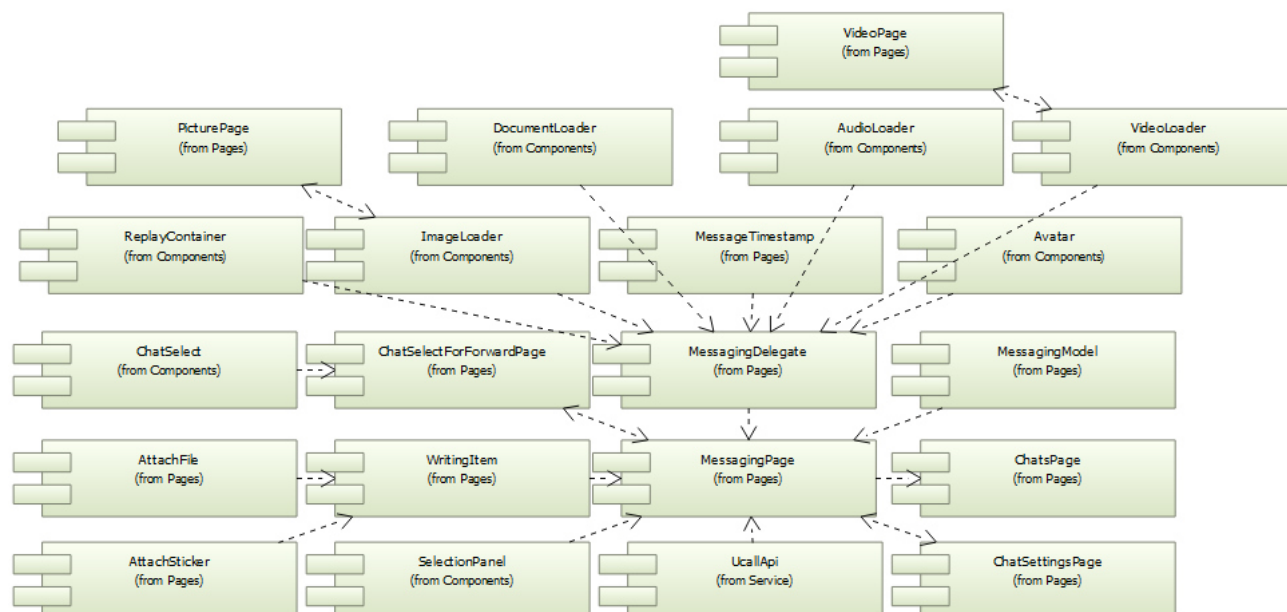


Рисунок 14 - Диаграмма связей QML-компонентов в рамках экрана взаимодействия между пользователями

Экран деталей взаимодействия

Данный экран отображает детали взаимодействия. Детали включают в себя название взаимодействия, меню просмотра файлов в рамках взаимодействия, для взаимодействия с конкретным пользователем доступен просмотр общих групповых взаимодействий, а для групповых взаимодействий доступен просмотр состава участников.

Для взаимодействия с конкретным пользователем доступно удаление взаимодействия.

Для групповых взаимодействий доступна возможность выйти из группы. Если пользователь является создателем или администратором группы, то он может редактировать её свойства, либо удалить группу целиком.

Диаграмма связей QML-компонентов в рамках экрана деталей взаимодействия между пользователями представлена на рисунке 15.

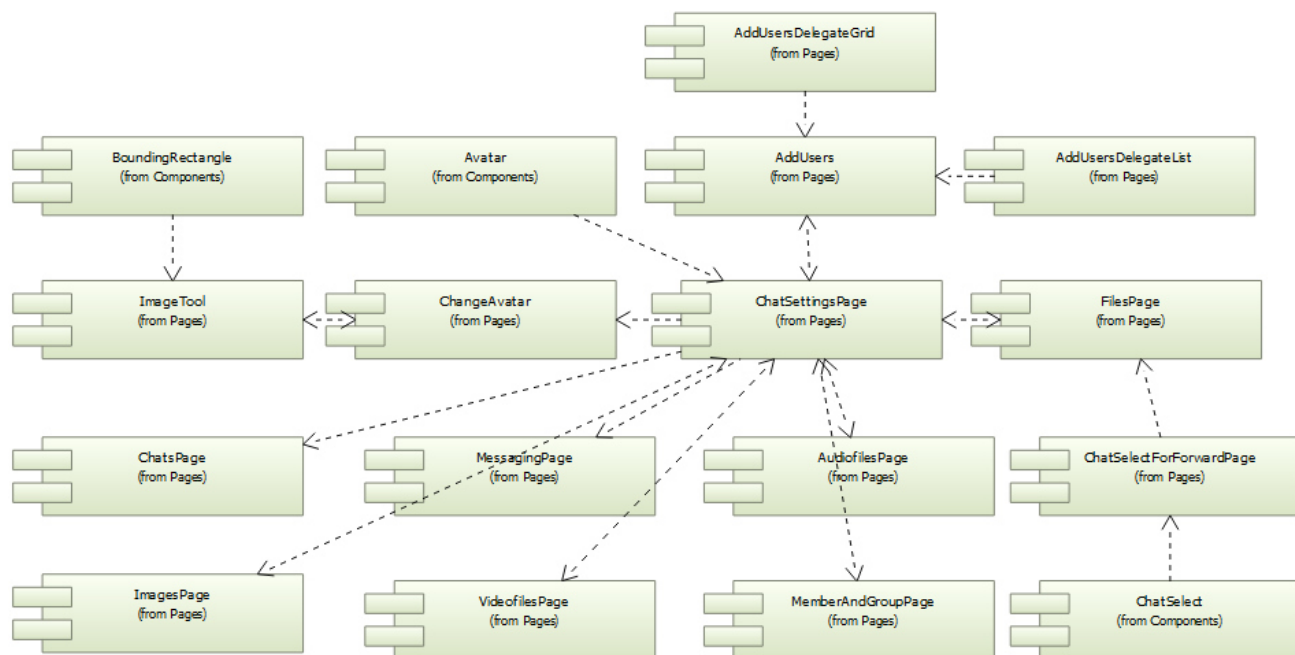


Рисунок 15 - Диаграмма связей QML-компонентов в рамках экрана деталей взаимодействия между пользователями

Описанные выше экраны представляют пользователю удобный графический интерфейс для взаимодействия с функционалом приложения. К данным QML-компонентам подключены классы, написанные на языке C++, которые реализуют сложные и трудоёмкие операции.

Общая схема взаимодействия пользователя с интерфейсом представлена на рисунке 16.

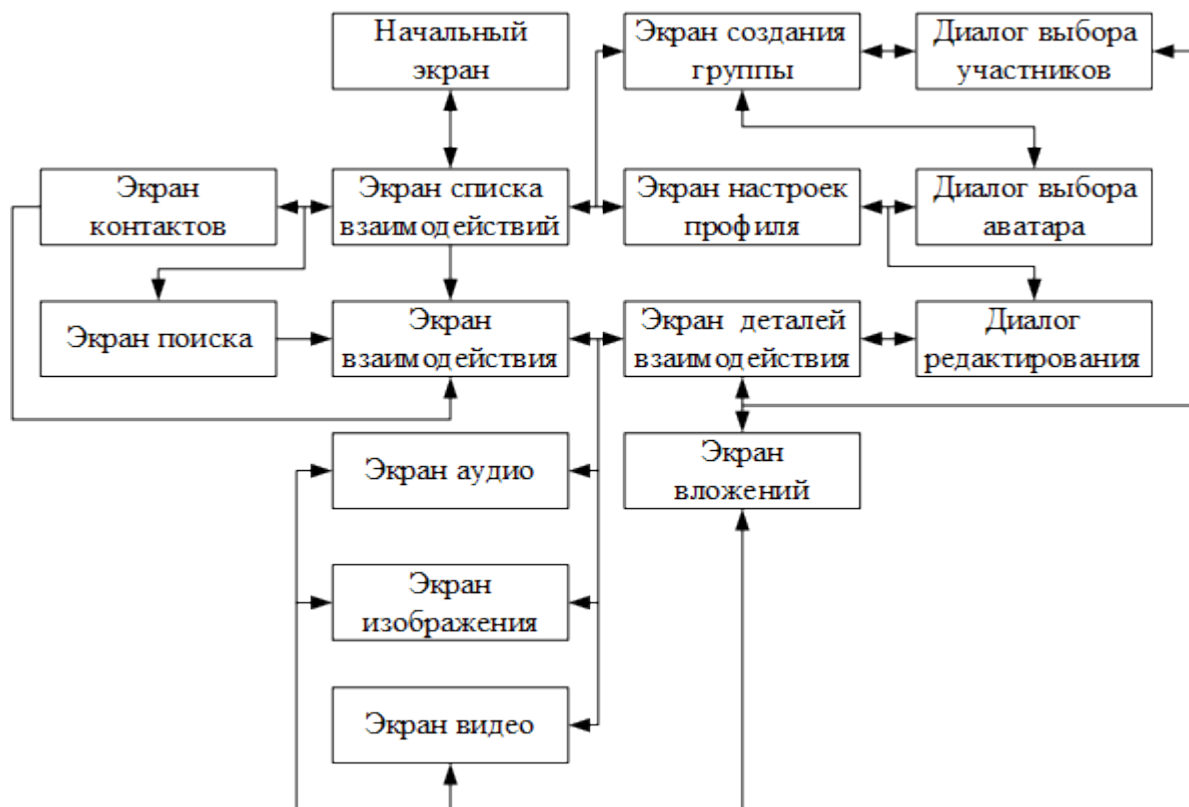


Рисунок 16 – Схема взаимодействия пользователя с интерфейсом

Описание структуры БД

Для реализации БД разрабатываемого приложения использована поддерживаемая IDE СУБД SQLite. Для работы с БД реализован отдельный класс «DBManager». Листинг класса «DBManager» представлен в Приложении А.

SQLite осуществляет хранение данных в едином стандартном файле на выполняющем устройстве [11]. При этом простота достигается благодаря блокировке файла, хранящего данные, в начале выполнения процесса записи.

В рамках данного приложения было реализовано две таблицы в базе данных: одна для хранения информации о текущих активных взаимодействиях пользователя, другая для хранения отдельных записей в рамках активных взаимодействий пользователя.

Ниже представлены таблицы с описанием полей и структуры разработанной БД.

В Таблице 1 представлена структура таблицы «Чаты».

Таблица 1 - Структура таблицы «Чаты»

Название поля	Описание поля	Тип данных
Id	Уникальный идентификатор	String
Type	Тип взаимодействия	Integer
Name	Имя чата	String
ChatModel	Модель взаимодействия	String
Timestamp	Время последнего сообщения	Integer
Pinned	Флаг закреплённости	Integer
MediaInfo	Сведения о файлах в активности	String
Draft	Черновик активности	String

Таблица 2 описывает структуру таблицы «Сообщения».

Таблица 2 - Структура таблицы «Сообщения»

Название поля	Описание поля	Тип данных
localId	Уникальный идентификатор на локальном устройстве	String
serverId	Уникальный идентификатор сервера	String
chatId	Идентификатор взаимодействия, к которому относится данная запись	String
content	Текст записи	String
timestamp	Время записи	Integer
messageModel	Модель записи	String

Данная база данных позволяет локально сохранять всю необходимую информацию, предоставляя к ней доступ без необходимости повторных обращений к серверу.

Описание механизмов передачи данных с помощью протоколов TCP/IP и WebSocket

Так как приложение базируется на архитектуре «клиент-сервер», то необходимо выстроить механизмы сетевого взаимодействия с конечным сервером [12]. Эти механизмы должны включать в себя отправку REST-запросов с использованием протокола TCP/IP, а также обработку ответа на

подобные запросы, и механизм передачи и приёма команд с использованием протокола WebSocket.

Для организации работы с протоколом передачи WebSocket достаточно использовать предложенную разработчиками целевой платформы библиотеку, которая подключается к проекту и предоставляет весь необходимый функционал.

Для взаимодействия с сервером по протоколу TCP/IP был разработан базовый класс «RestApi», листинг которого представлен в приложении В, и QML-оболочка этого класса «QmlRestApi», листинг которого содержится в приложении С.

Отправка запроса и считывание ответа осуществляется с помощью классов QUrl, QNetworkRequest и QNetworkReply, которые предоставляют удобные инструменты для отправки любых типов REST-запросов на сервер и считывания ответа.

Алгоритм работы разработанного механизма изображён на рисунке 17.

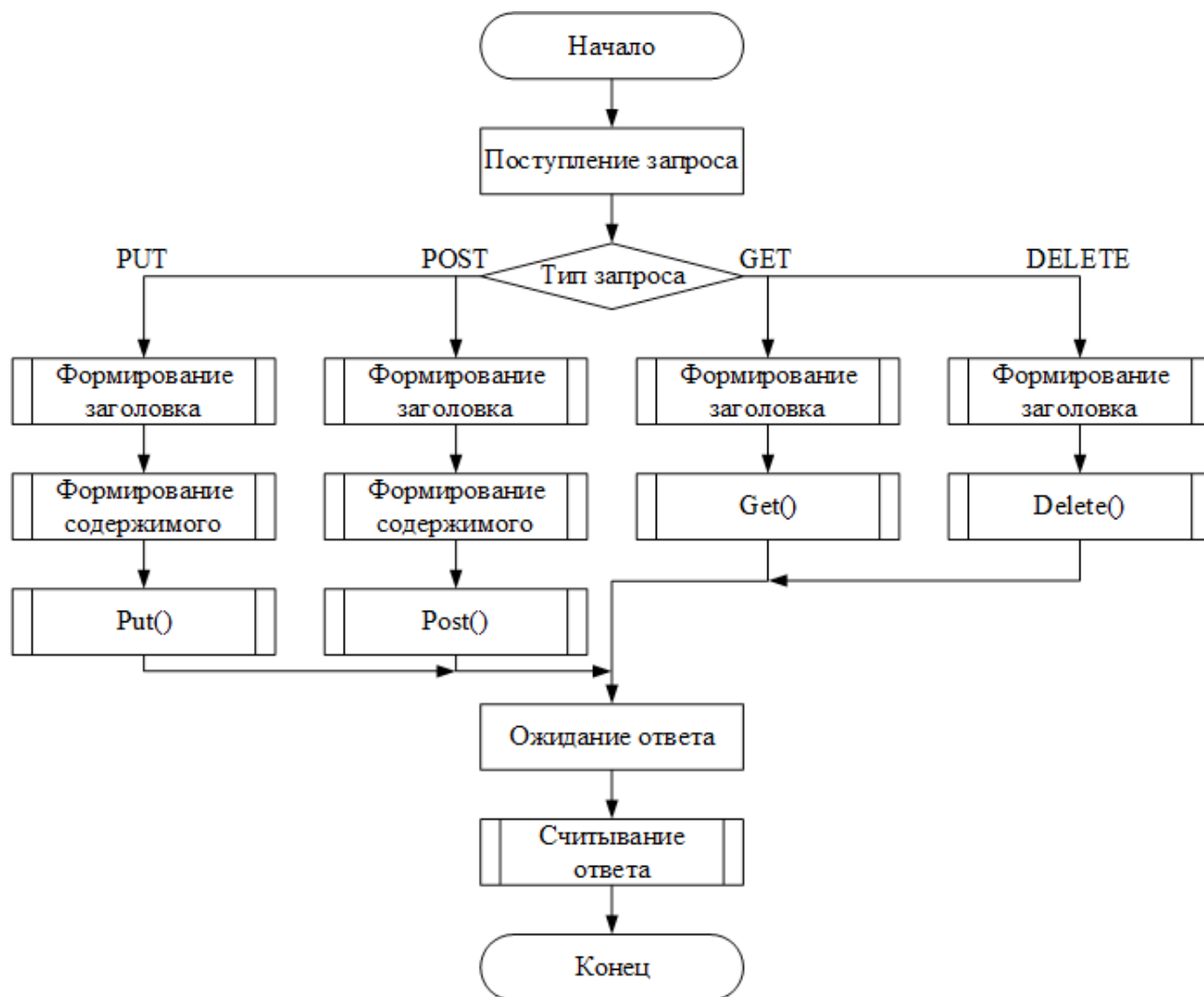


Рисунок 17 – Алгоритм отправки сообщения на сервер.

Данный механизм обеспечивает формирование стандартных REST-запросов, однако механизм взаимодействия с сервером включает в себя необходимость отправки REST HTTP-Multipart запросов. Для реализации механизма отправки подобных запросов и обработки ответа был реализован класс «httpmultipart», листинг которого представлен в Приложении D.

Общая схема сетевого взаимодействия приложения и сервера представлена на рисунке 18.

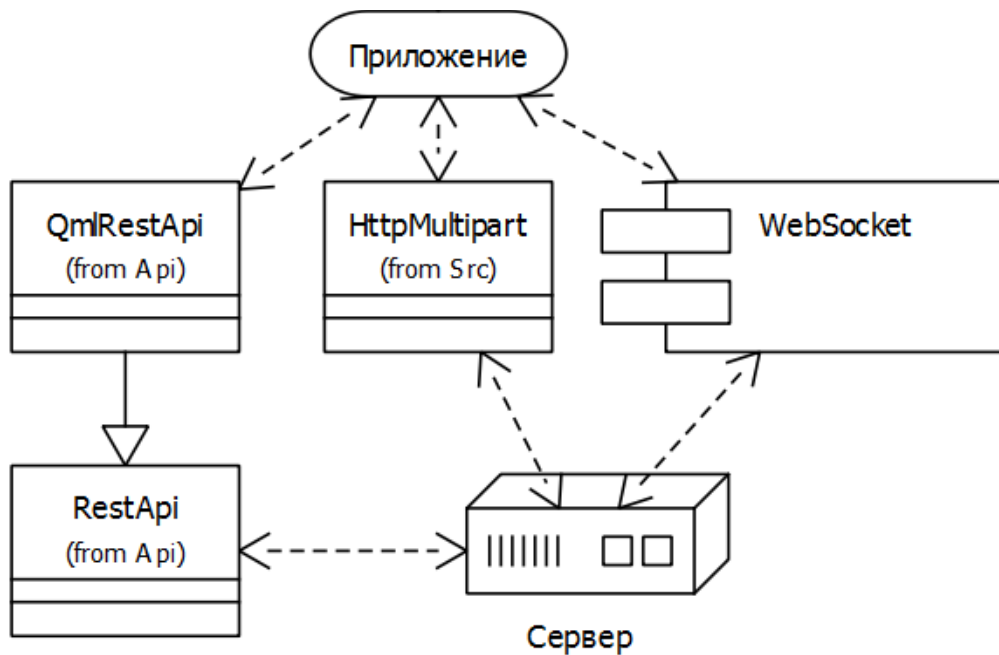


Рисунок 18 – Схема сетевого взаимодействия

Совокупность разработанных механизмов обеспечивает организацию удобного и безопасного сетевого взаимодействия с сервером

Описание C++ классов

Данные классы реализуют ряд дополнительных трудоёмких операций, которые нецелесообразно или невозможно осуществлять средствами языка JavaScript. Список классов включает в себя:

— «Downloader», структура которого представлена на рисунке 19, и «QDownloader», структура которого представлена на рисунке 20, для организации REST-запросов к серверу с целью скачивания данных большого и малого объёма соответственно;

Downloader (from Src)
Q_OBJECT - QNetworkAccessManager* m_manager - QString m_source - QString m_tag - int m_maxRedirects + inline QString source() const { return m_source + } + inline QString tag() const { return m_tag + }
+ explicit Downloader(QObject *parent = Q_NULLPTR) + virtual ~Downloader() + invocable void download(const QString& source, const QString& tag = "") > void finished(Donloader* donloader, QNetworkReply* reply) > void redirectsExceeded(Donloader* donloader, QNetworkReply* reply) -\$ void downloadComplete(QNetworkReply* reply)

Рисунок 19 – Класс «Donloader».

QDownloader (from Src)
Q_OBJECT - QNetworkAccessManager *m_manager - QNetworkReply *m_reply - QFile *m_file - int m_maxRedirects - Q_SIGNALS: - Q_SLOTS:
+ explicit QDownloader(QObject *parent = Q_NULLPTR) + virtual ~QDownloader() + invocable bool validateFileSize(const QString& path, int bytes) + invocable bool validatePath(const QString& path) + invocable void download(const QString& fileURL, const QString& saveFilePath) + invocable QString getFileName(const QString& path) const - void request(const QUrl& url) - void finished() - void progress(qint64 read, qint64 total) - void redirectsExceeded() - void error(QString errorMessage) - void onFinished(QNetworkReply*) - void onDownloadProgress(qint64, qint64) - void onReadyRead() - void onReplyFinished()

Рисунок 20 – Класс «QDonloader».

— «ImageHandler», «AvatarProvider» и «RoundedImage» для организации обработки, хранения, и получения изображений как от локальной файловой системы, так и от сервера. Структура этих классов представлена на рисунках 21, 22 и 23 соответственно;

ImageHandler (from Src)
Q_OBJECT
+ explicit ImageHandler(QObject *parent = nullptr) + virtual ~ImageHandler() + invokable QString save (const QString &path, const double offsetX = 0, const double offsetY = 0, const double width = 0, const double height = 0) + invokable void checkTempFile()

Рисунок 21 – Класс «ImageHandler».

AvatarProvider (from Src)
- struct avatar { QString userName - QString avatarPath - } - QString m_baseUrl - QQueue<avatar> m_queue - QMap<QString, bool> m_complete - QString m_cachePath - QAtomicInt m_parallelCount - QReadWriteLock m_lock
- void download() + explicit AvatarProvider() + QImage requestImage(const QString &id, QSize *size, const QSize &requestedSize) + QString baseUrl() const + void setBaseUrl(const QString &url) + Q_PROPERTY(QString baseUrl READ baseUrl WRITE setBaseUrl) + invokable void add(const QString& userName, const QString& avatarPath) + invokable void update(const QString& userName, const QString& avatarPath) + invokable void clearCache() + invokable QString toLocalFileName(const QString& name) const > invokable void saveAvatarComplete(QString source) -\$ void downloadComplete(Downloader* downloader, QNetworkReply* reply)

Рисунок 22 – Класс «AvatarProvider».

RoundedImage (from Src)
Q_OBJECT - QImage _image - QString _path - bool _group - QString _baseUrl
+ explicit RoundedImage(QQuickItem* parent = NULL) + ~RoundedImage() + Q_PROPERTY(QString path READ path WRITE setPath NOTIFY pathChanged) + Q_PROPERTY(bool isGroup READ isGroup WRITE setIsGroup NOTIFY isGroupChanged) + Q_PROPERTY(QString baseUrl READ baseUrl WRITE setBaseUrl) + void paint(QPainter* painter) + void setPath(QString newPath) + void setIsGroup(bool newFlag) + void setBaseUrl(QString newUrl) + QString path() + bool isGroup() + QString baseUrl() + void download() > void pathChanged() > void isGroupChanged() > void success() -\$ void downloadComplete(Downloader* downloader, QNetworkReply* reply)

Рисунок 23 – Класс «RoundedImage».

— «RNCrypt», с помощью которого осуществляется шифрование и дешифрование данных. Структура класса представлена на рисунке 24;

RNCrypt (from Src)
Q_OBJECT
+ explicit RNCrypt(QObject *parent = 0) + virtual ~RNCrypt() + invocable QVariant encryptText(const QVariant& data) const + invocable QVariant decryptText(const QVariant& data) const + QVariant toMessageModelProxy(const QVariant& msgAsText) const + QVariant toChatModelProxy(const QVariant& hitemAsText) const

Рисунок 24 – Класс «RNCrypt».

Вывод

В данном разделе был описан процесс реализации приложения для решения поставленной задачи.

В процессе реализации были разработаны графические QML-компоненты, обеспечивающие удобный интерфейс взаимодействия пользователя и программы, спроектирована локальная база данных для хранения личных данных пользователя и организована работа с ней, разработки методы передачи и приёма данных с использованием протоколов

TCP/IP и WebSocket, а также реализованы дополнительные C++ классы для вспомогательных трудоёмких задач.

3 ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

Данный раздел содержит краткую инструкцию для пользователя по использованию функционала приложения.

Авторизация в системе

Первый экран, который видит пользователь, запустивший приложения, предлагает пройти авторизацию с помощью пары значений логин-пароль.

После введения данных и нажатия на кнопку «Войти» происходит REST-запрос на сервер. В случае указания правильных данных происходит перенаправление на страницу списка активных взаимодействий, иначе выдаётся сообщение об ошибке с указанием типа ошибки.

Интерфейс страницы авторизации представлен на рисунке 25.

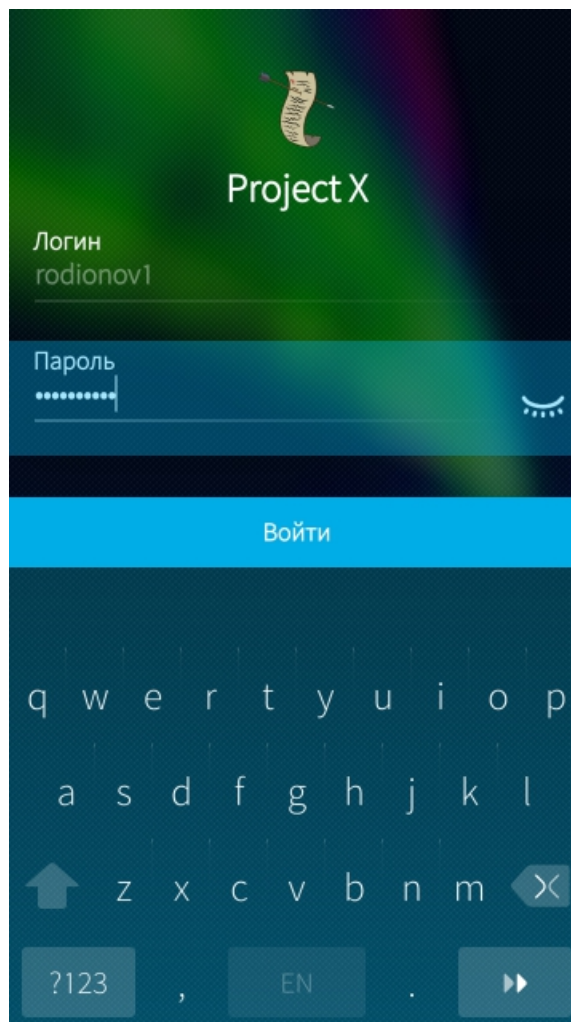


Рисунок 25 - Экран авторизации

Экран списка активных взаимодействий

Интерфейс данного окна представлен на рисунке 26. Функционал, который предоставляет этот графический элемент подробно описан в пункте 2.1.2.

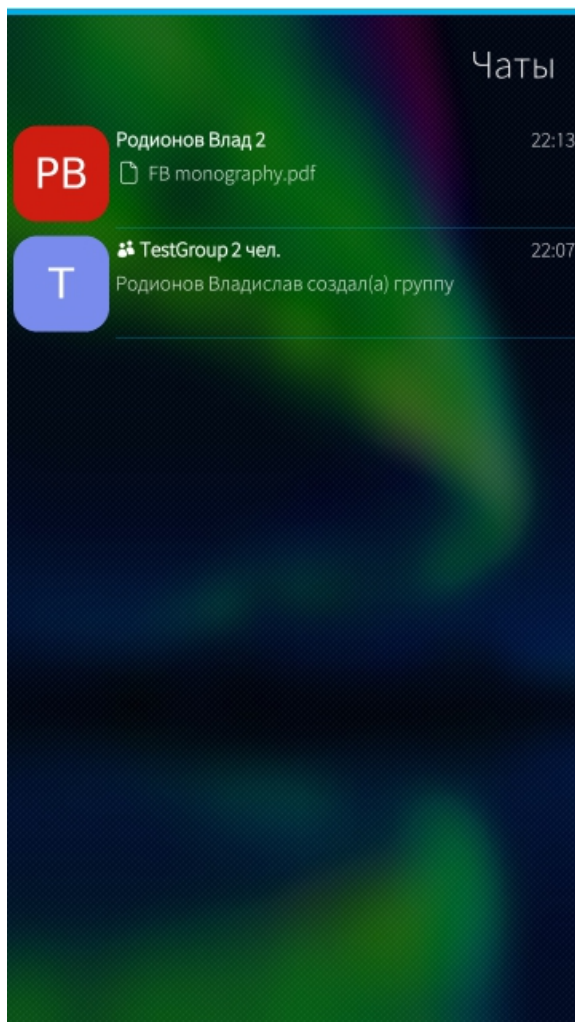


Рисунок 26 - Экран «Чаты»

Чтобы выйти из профиля, перейти к окну настроек личного аккаунта пользователя, создания новой группы, возможных контактов, а также к окну поиска в системе, следует использовать выдвигаемое сверху меню, что продемонстрировано на рисунке 27.

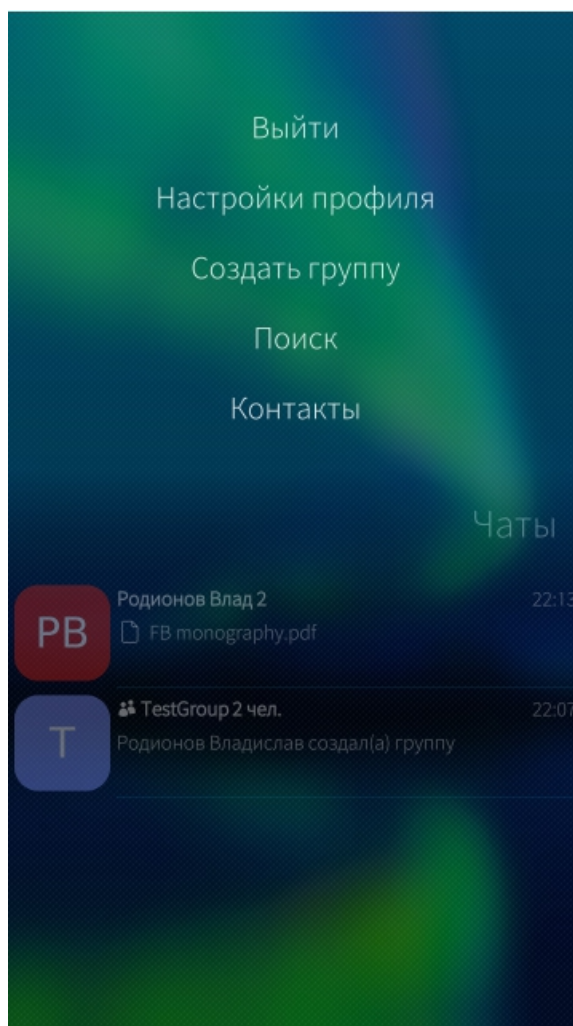


Рисунок 27 - Верхнее меню экрана «Чаты»

Чтобы перейти к экрану конкретного взаимодействия следует нажать по записи этого взаимодействия в списке.

Также записи имеют контекстное меню, выдвигаемое по долгому нажатию на элемент в списке, которое предоставляет функционал прикрепления чата к верхней позиции списка, а также быстрый переход к свойствам взаимодействия. Контекстное меню продемонстрировано на рисунке 28.

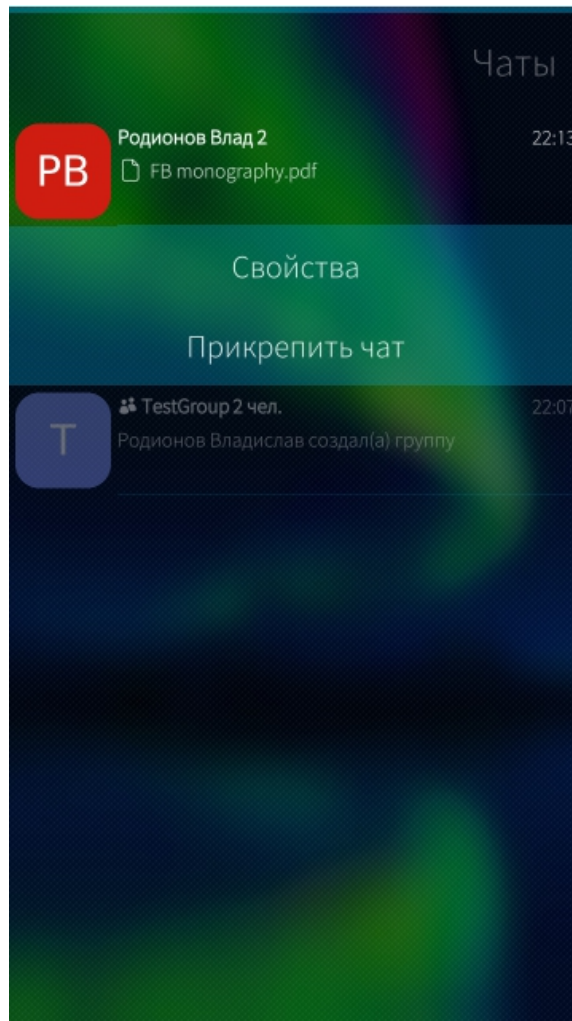


Рисунок 28 - Контекстное меню записи на экране «Чаты»

Экран настроек профиля

Функционал данного экрана описан в пункте 2.1.6. Интерфейс данного окна представлен на рисунке 29.

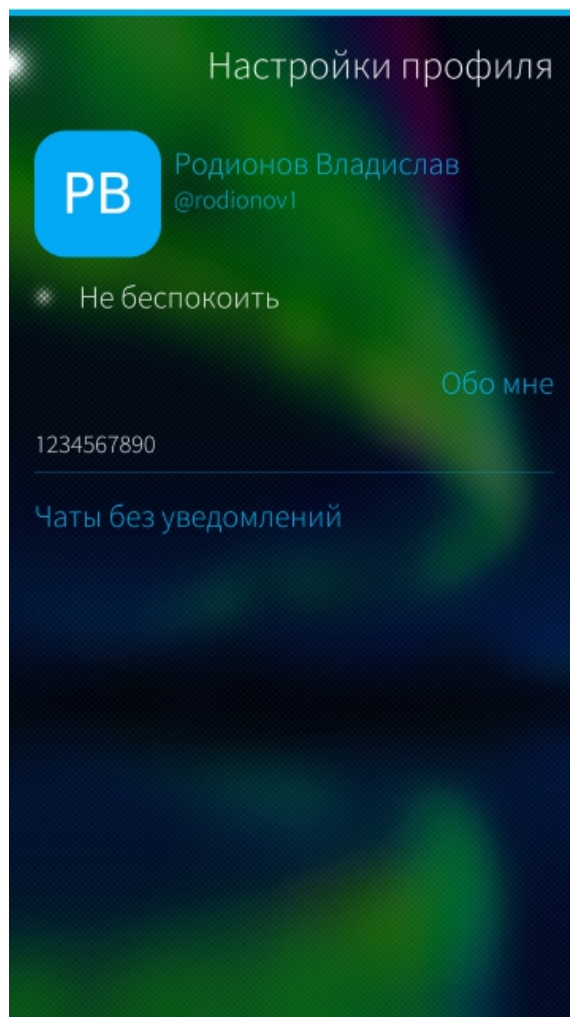


Рисунок 29 - Экран настроек профиля

Чтобы изменить настройки личного аккаунта следует воспользоваться вытягиваемым сверху меню, которое представлено на рисунке 30. После нажатия на один из пунктов меню откроется диалог изменения соответствующего параметра личной записи.

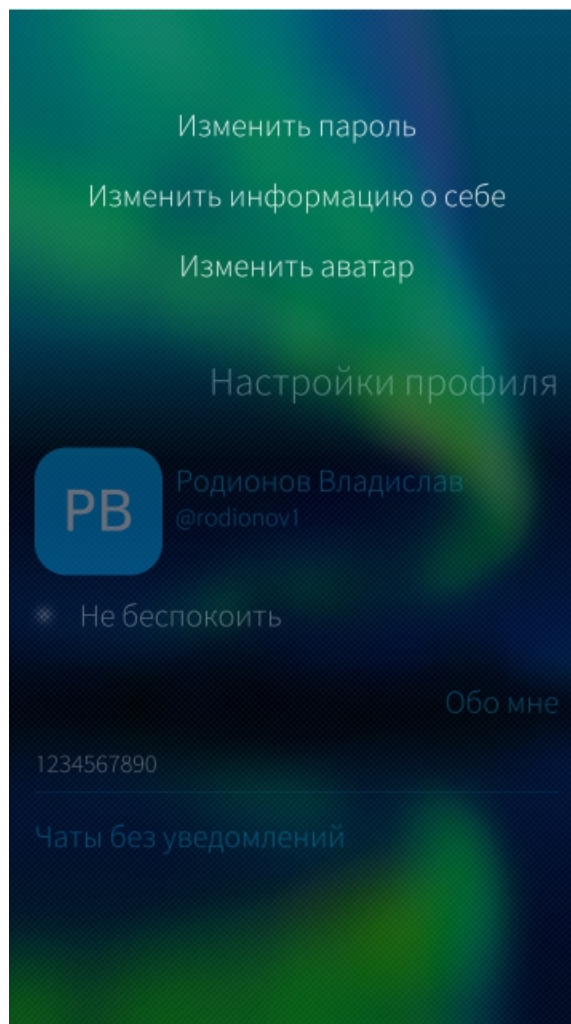


Рисунок 30 - Вытягиваемое меню экрана настроек профиля

Чтобы просмотреть список взаимодействий, у которых отключены уведомления о непрочитанных сообщениях, следует воспользоваться соответствующей кнопкой.

Экран создания новой группы

Функционал данного окна описан в пункте 2.1.5. Интерфейс данного окна представлен на рисунке 31.

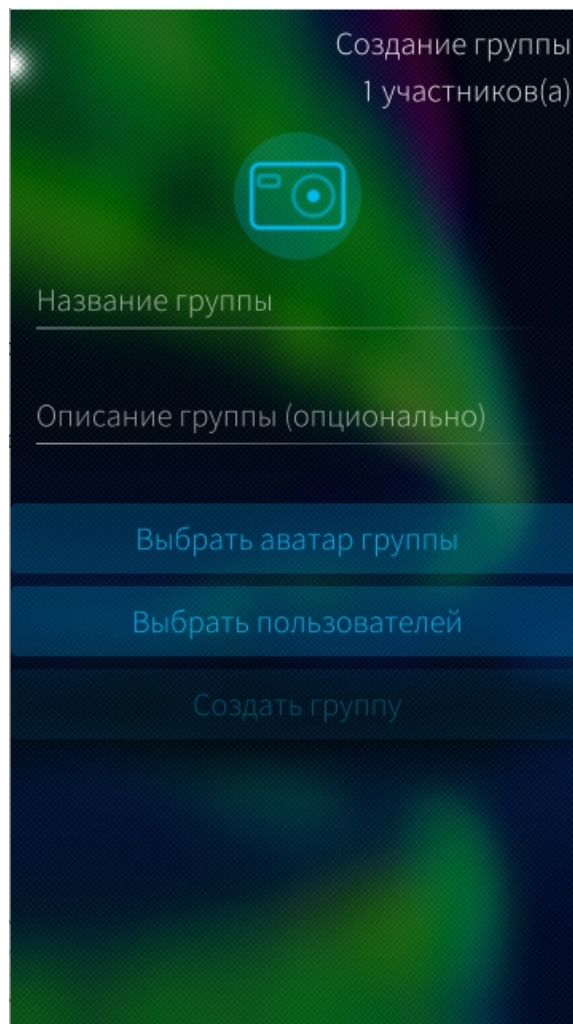


Рисунок 31 - Экран создания новой группы

Чтобы выбрать участников группы необходимо воспользоваться кнопкой «Выбрать пользователей», после чего откроется страница с выбором участников группы из списка контактов. На данной странице можно назначить администраторов группы с помощью нажатия на значок звёздочки в правой части записи. А для того, чтобы завершить выбор, необходимо нажать кнопку «Завершить выбор». Страница выбора пользователей представлена на рисунке 32.

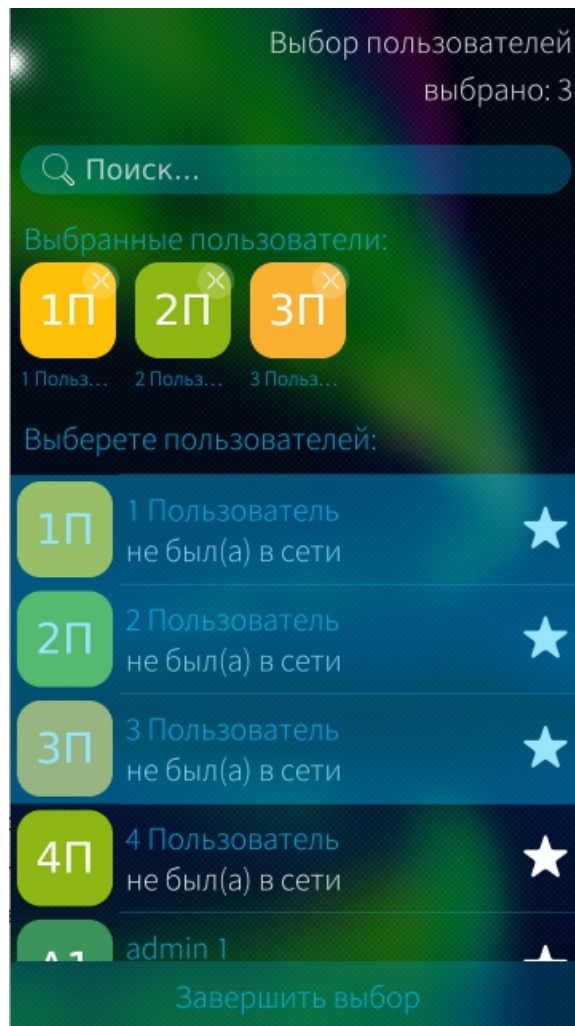


Рисунок 32 - Экран выбора участников

Чтобы выбрать аватар группы необходимо воспользоваться кнопкой «Выбрать аватар группы», после чего откроется диалог выбора аватара группы, представленный на рисунке 33.

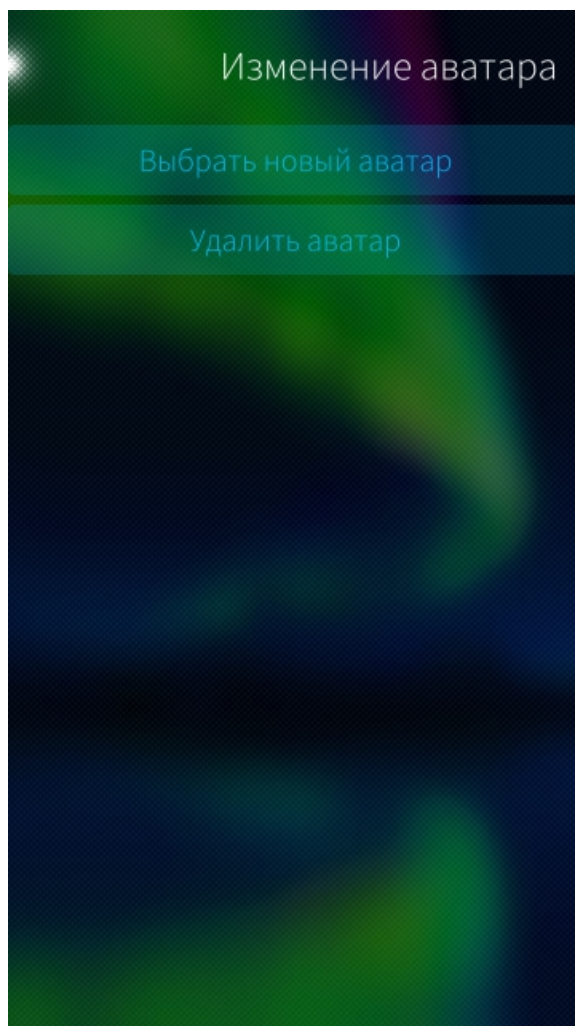


Рисунок 33 - Диалог выбора аватара группы

Выбрать можно любое изображение, находящееся в памяти телефона, после чего будет предложена возможность обрезать изображение.

Для завершения создания группы необходимо нажать на кнопку «Создать группу». Важным условием является то, что имя группы не должно быть короче трёх символов.

Экран поиска

Функциональные возможности данного экрана описаны в пункте 2.1.4. Интерфейс данного экрана представлен на рисунке 34.

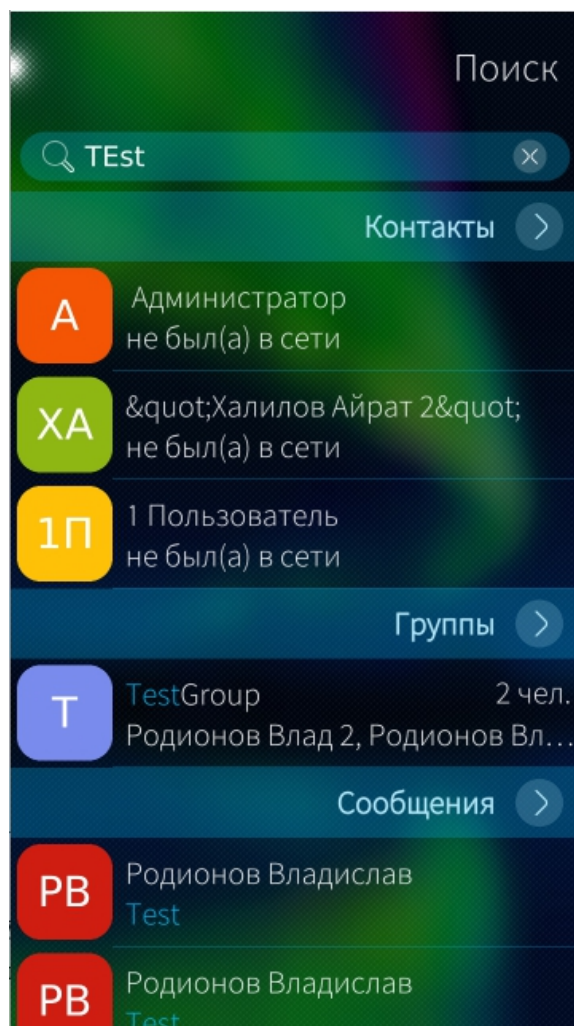


Рисунок 34 - Экран поиска

Для осуществления поиска необходимо ввести искомую фразу в поле «Поиск...». Поиск регистронезависим. Результаты поиска разделены на категории. После осуществления поиска из каждой категории выводятся не все записи, чтобы вывести все записи данной категории необходимо нажать на соответствующую категорию.

По нажатию на запись из поиска можно перейти в соответствующее взаимодействие.

Экран возможных контактов

Функционал данного раздела описан в пункте 2.1.3. Интерфейс данного раздела представлен на рисунке 35.

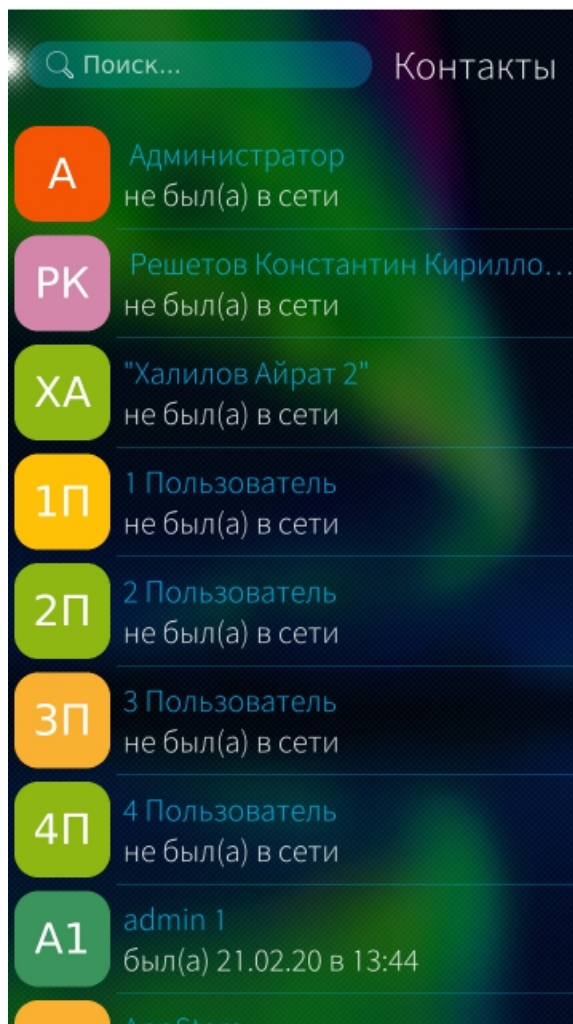


Рисунок 35 - Экран «Контакты»

Поддерживается поиск по данному разделу. Чтобы осуществить поиск, необходимо ввести искомый запрос в поле «Поиск...». По нажатию на запись можно перейти на экран взаимодействия с выбранным пользователем.

Экран взаимодействия с пользователем или группой пользователей

Функционал данного окна описан в пункте 2.1.7. Интерфейс данного окна представлен на рисунке 36.

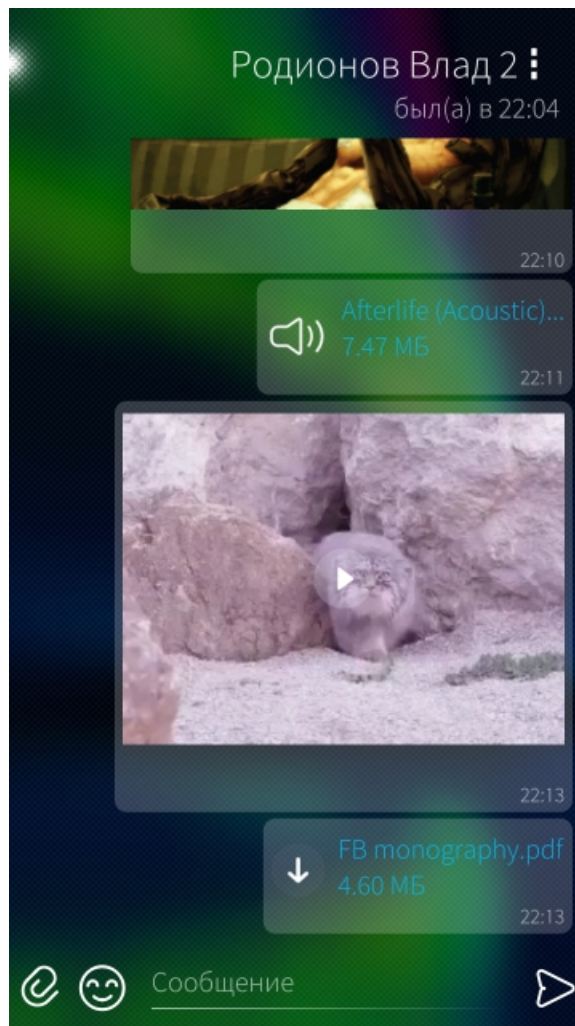


Рисунок 36 - Экран взаимодействия

Чтобы отправить собеседнику текстовое сообщение, необходимо ввести текст в поле «Сообщение» и нажать на клавишу ввода, или на значок стрелки.

Поддерживаются текстовые сообщения, пользовательские файлы, которые можно загрузить в локальную файловую систему, аудиофайлы и видеофайлы, которые можно воспроизвести по нажатию на запись после загрузки файла, а также изображения, которые можно просмотреть по нажатию на запись после загрузки файла.

Чтобы отправить какой-либо файл, необходимо нажать на значок скрепки, после чего использовать открывшееся диалоговое меню, изображённое на рисунке 37.

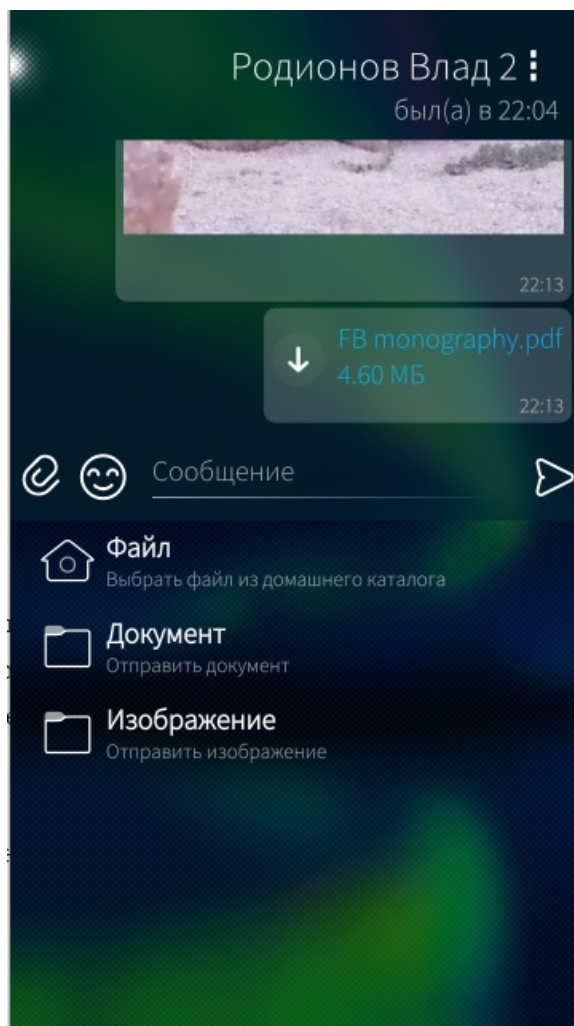


Рисунок 37 - Диалоговое меню отправки файла

После выбора файла к нему можно прикрепить текстовый комментарий.

Также поддерживается отправка и приём миниатюрных изображений «стикеров». Для отправки такого изображения следует нажать на значок смайла, после чего выбрать нужный «стикер» из выпадающего меню, представленного на рисунке 38.

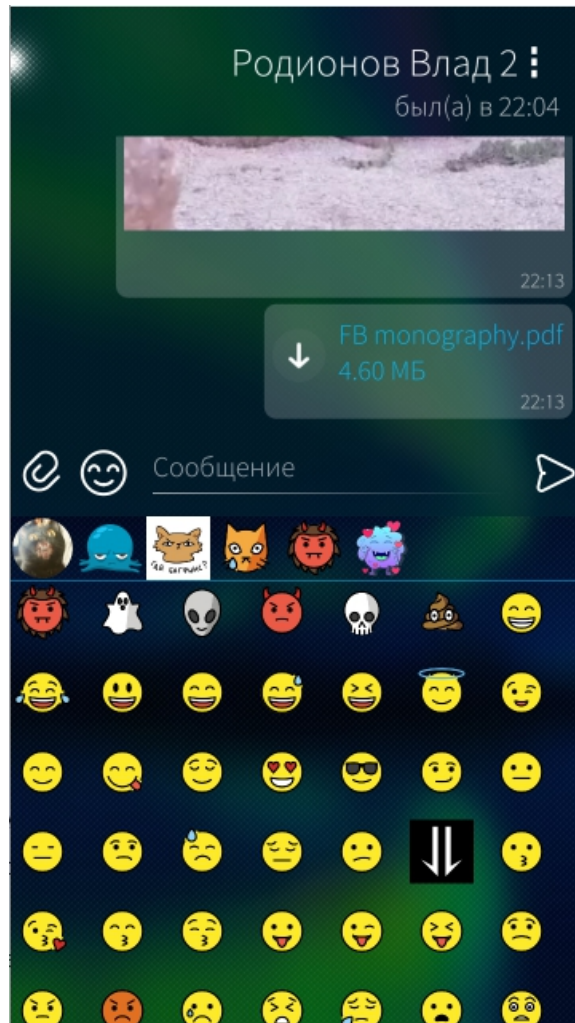


Рисунок 38 - Диалоговое меню отправки "стикера"

У записей есть контекстное меню, выдвигаемое по долгому нажатию на запись. Оно продемонстрировано на рисунке 39 и предоставляет дополнительный функционал управления записями.

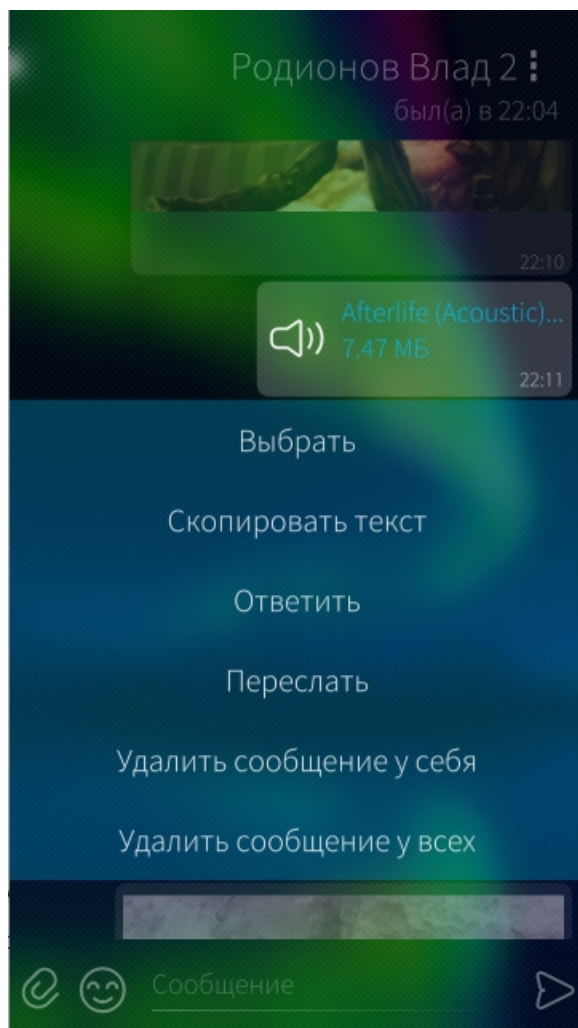


Рисунок 39 - Контекстное меню записи в окне взаимодействия

По нажатию на заголовок страницы можно перейти на окно свойств взаимодействия.

Экран свойств взаимодействия

Функционал данного элемента описан в пункте 2.1.8. Интерфейс данного окна для личного взаимодействия представлен на рисунке 40, а для группового – на рисунке 41.

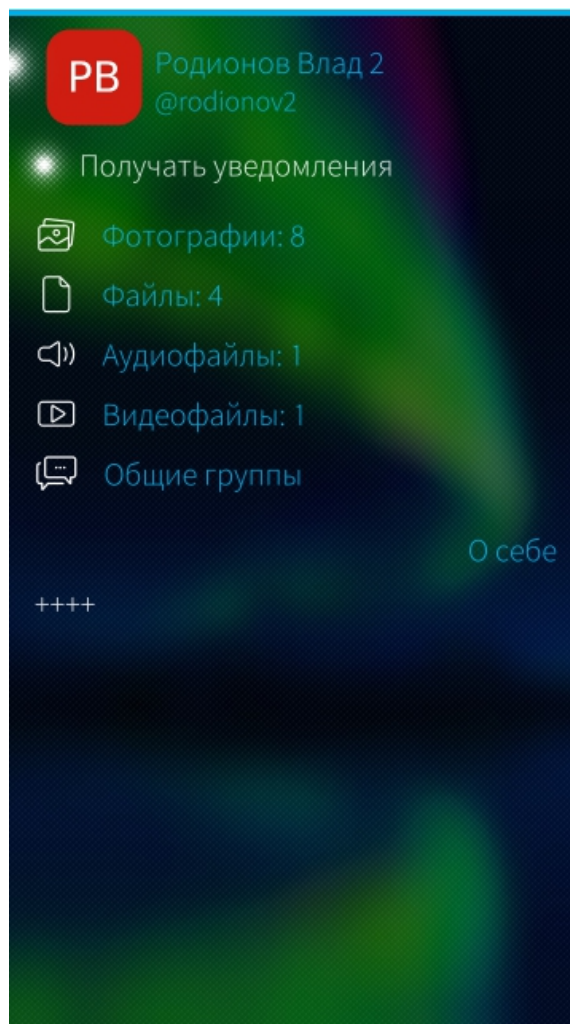


Рисунок 40 - Экран свойств взаимодействия с пользователем

Для просмотра файлов различного типа в данном взаимодействии следует нажать на соответствующий пункт меню. Пункт «Общие группы» откроет список групп, в которых пользователь состоит вместе с выбранным аккаунтом.

Нажатие на «Получать уведомления» позволит включить или отключить оповещение о новых сообщениях от данного пользователя.

Также с помощью вытягиваемого сверху меню можно очистить историю взаимодействия.

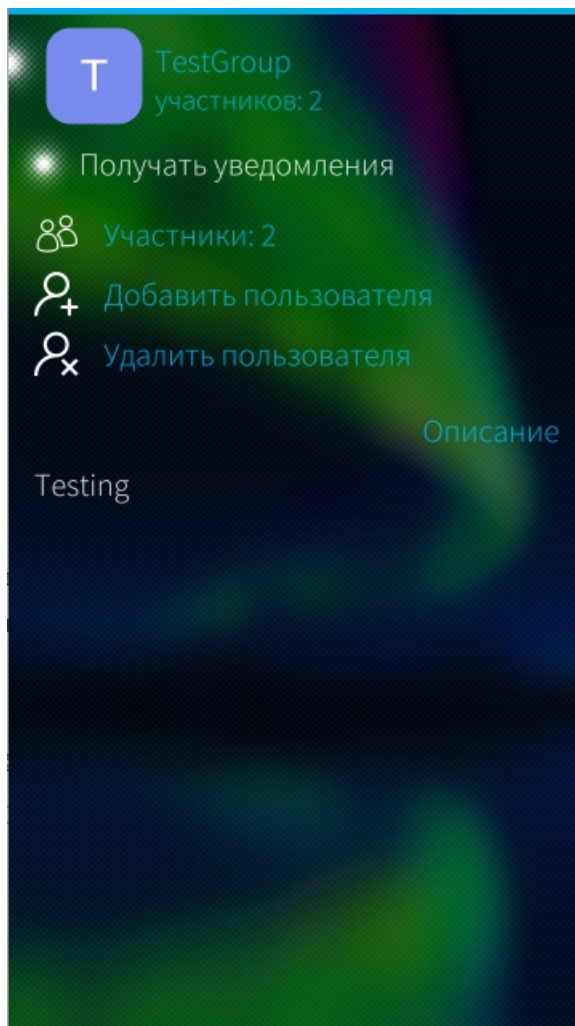


Рисунок 41 - Экран свойств взаимодействия между группой пользователей

Пункт «Получать уведомления» аналогичен пункту из экрана свойств личной беседы.

По нажатию на пункт «Участники: ...» можно открыть список всех пользователей, участвующих в данном взаимодействии.

Если пользователь является администратором или создателем группы, то ему доступна возможность добавления и удаления участников с помощью соответствующих кнопок, по нажатию на которые произойдёт переход на страницу, аналогичную странице выбора участников при создании группы.

С помощью вытягиваемого сверху меню можно покинуть группу, а также, если пользователь является создателем или администратором, отредактировать настройки группы или удалить её.

Вывод

Инструкция для пользователя с описанием пользовательского интерфейса, которая приводится в данной главе, позволит пользователю легко освоить функциональные возможности приложения. Даны описания всех основных экранов и методы взаимодействия с ними, включая описание возможных внештатных ситуаций и реакции системы на них.

Разработанный интерфейс является простым и интуитивно понятным для пользователя.

4 БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

Для нормальной работы пользователя с программным обеспечением должны быть соблюдены общие рекомендации для обеспечения безопасной работы с ПЭВМ.

Данные рекомендации включают в себя следующее:

- ПЭВМ должны соответствовать требованиям санитарных правил, и каждый их тип подлежит санитарно-эпидемиологической экспертизе с оценкой в испытательных лабораториях, аккредитованных в установленном порядке;

- допустимые уровни звукового давления и уровней звука, создаваемых ПЭВМ, не должны превышать установленных норм;

- временные допустимые уровни электромагнитных полей, создаваемых ПЭВМ, не должны превышать установленных норм;

- концентрации вредных веществ, выделяемых ПЭВМ в воздух помещений, не должны превышать предельно допустимых концентраций, установленных для атмосферного воздуха;

- мощность экспозиционной дозы мягкого рентгеновского излучения в любой точке на расстоянии 0,05 м от экрана и корпуса ВДТ (на электронно-лучевой трубке) при любых положениях регулировочных устройств не должна превышать 1 мкЗв/час (100 мкР/час);

- документация на проектирование, изготовление и эксплуатацию ПЭВМ не должна противоречить требованиям санитарных правил.

Также необходимо чтобы используемое ПЭВМ, поддерживало возможность поворота корпуса и его фиксации в установленном положении. Расцветка ПЭВМ должна состоять из спокойных цветов, не отвлекающих внимание.

Если планируется использование программного обеспечения на рабочем месте, то необходимо соблюдать требования к помещению и требования к организации рабочего места.

Помещение, в котром планируется взаимодействие с разработанным продуктом, его размер (площадь, объем) должно соответствовать количеству полностью оснащенных рабочих мест сотрудников. Кроме того, должны быть учтены соответствующие нормы температуры, освещения, чистоты воздуха, обеспечена изоляция, от производственных шумов и т.п. На одного работающего сотрудника, согласно санитарным нормам труда СН 245-71, должно быть помещение объемом не менее 15 м³, площадью не менее 4,5 м².

Основные требования к организации рабочего места и его оборудованию прописаны в ГОСТ 12.2.032 78, а именно, высота рабочей поверхности стола для сотрудника должна регулироваться в пределах 680-800 мм; при отсутствии таковой возможности высота рабочей поверхности стола должна составлять 725 мм.

Рабочий стол должен иметь следующие параметры: ширина соответствует 800, 1200 или 1400 мм, глубина – 800 или 1000 мм при нерегулируемой высоте, которая равна 725 мм. Кроме того, пространство для ног высотой в рабочей части должно быть не менее 600 мм, шириной – не менее 500 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм.

Рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также по расстоянию спинки до переднего края сиденья.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной выпускной работы была изучена концепция систем мгновенного обмена данными, получены навыки реализации программного обеспечения для мобильной платформы «Аврора».

Результатами решения поставленной задачи является разработанное и протестированное приложение, обеспечивающее взаимодействие между пользователями в режиме реального времени.

Благодаря используемым решениям приложение соответствует выдвигаемым требованиям, обладает удобным и интуитивным графическим интерфейсом, отказоустойчиво и безопасно, а также может быть модифицировано и масштабировано для добавления нового функционала.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вакуленко С. П., Шапаренко Ю. М. ОБЗОР ПРОГРАММ ОЦЕНКИ РАБОТОСПОСОБНОСТИ ОПЕРАЦИОННЫХ СИСТЕМ И ТЕСТИРОВАНИЯ КОМПЬЮТЕРОВ //WORLD SCIENCE: PROBLEMS AND INNOVATIONS. – 2019. – С. 72-76.
2. Манукова Е. Ю., Захарова М. В. Использование сервисов мгновенного обмена сообщениями в современной массовой коммуникации //Современная филология. – 2017. – С. 85-88.
3. Ortiz S. Instant Messaging: No Longer Just Chat //Computer. – 2001. – №. 3. – С. 12-15.
4. Werry C. C. Internet relay chat //Computer-mediated communication: Linguistic, social and cross-cultural perspectives. – 1996. – С. 47-63.
5. Gregori M. E., Cámara J. P., Bada G. A. A jabber-based multi-agent system platform //Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. – 2006. – С. 1282-1284.
6. Umair A. et al. User relationship classification of Facebook messenger mobile data using weka //International Conference on Network and System Security. – Springer, Cham, 2018. – С. 337-348.
7. Harwit E. WeChat: Social and political development of China's dominant messaging app //Chinese Journal of Communication. – 2017. – Т. 10. – №. 3. – С. 312-327.
8. Akbari A., Gabdulhakov R. Platform surveillance and resistance in Iran and Russia: the case of Telegram //Surveillance & Society. – 2019. – Т. 17. – №. 1/2. – С. 223-231.
9. Eunjeong C. Kakaotalk, a mobile social platform pioneer //SERI Quarterly. – 2013. – Т. 6. – №. 1. – С. 63.
10. Уйгурова С. А. ИНФОРМАТИЗАЦИЯ ОБЩЕСТВА НА ПРИМЕРЕ ИСПОЛЬЗОВАНИЯ МЕССЕНДЖЕРА WHATSAPP //СТУДЕНТ ГОДА 2019. – 2019. – С. 297-303.

11. Новиков Б. А., Горшкова Е. А. Основы технологий баз данных. Учебное пособие — М.: «ДМК Пресс», 2019. — С. 165-233.

12. Д. Куроуз, К. Росс. Компьютерные сети. Нисходящий подход. — М.: Издательство «Э», 2016. — С. 912.


```

{
    Q_UNUSED(engine);
    Q_UNUSED(scriptEngine);

    return Instance();
}

static DbManager *Instance()
{
    if (!sm_pInstance)
        sm_pInstance = new DbManager();
    return sm_pInstance;
}

/**
 * @brief Destructor
 *
 * Close the db connection
 */
~DbManager();

Q_INVOKABLE bool isOpen() const;
Q_INVOKABLE bool clear();
Q_INVOKABLE bool upsertChat(QJSValue model);
Q_INVOKABLE bool removeChat(const QString& roomId);
Q_INVOKABLE bool upsertMessage(QJSValue model);
Q_INVOKABLE bool removeMessage(const QString& anyId);
Q_INVOKABLE bool removeMessageInRoom(const QString&
roomId, const QString& anyId);
Q_INVOKABLE int getAllChats(QJSValue jsCallback);
Q_INVOKABLE int getPinnedChats(QJSValue jsCallback);
Q_INVOKABLE int getMessages(const QString& roomId,
QJSValue jsCallback, int limit = -1, int offset = 0);
Q_INVOKABLE bool countMessagesInChats(QJSValue
jsCallback);
Q_INVOKABLE int getMessagesCount(const QString&
roomId);
Q_INVOKABLE int getUnsentMessages(QJSValue
jsCallback);
Q_INVOKABLE int search(const QString& text, QJSValue
jsCallback);
Q_INVOKABLE bool updateMediaInfo(const QString&
roomId, const QString& mediaInfo);
Q_INVOKABLE QString getMediaInfo(const QString&
roomId);
Q_INVOKABLE bool setDraft(const QString& roomId, const
QString& draft);
Q_INVOKABLE QString getDraft(const QString& roomId);
Q_INVOKABLE int getChatCount();
Q_INVOKABLE bool addDelayedCommand(const QString&
cmd);

```

```

    Q_INVOKABLE int getDelayedCommands(QJSValue
jsCallback);
    Q_INVOKABLE bool removeDelayedCommand(int id);

    bool upsertMessage(const QString& serverId,
        const QString& localId,
        const QString& roomId,
        const QVariant& timestamp,
        const QString& content,
        const QString& messageModel);
    bool upsertServiceMessage(const QString& serverId,
        const QString& roomId,
        const QVariant& timestamp,
        const QString& content,
        const QString& messageModel);

private:
    bool _init();
};

#endif // DBMANAGER_H

```

Файл «dbmanager.cpp»

```
#include "dbmanager.h"

#define BEGIN_TRANSACTION(db) if (!db.transaction())
{ qWarning() << "BEGIN TRANSACTION failed:" <<
db.lastError().text(); return false; }
#define COMMIT_TRANSACTION(db) if (!db.commit())
{ qWarning() << "COMMIT TRANSACTION failed:" <<
db.lastError().text(); return false; }
#define EXEC_QUERY_CATCH_ERROR(q) if (!q.exec())
{ qWarning() << "QSqlQuery error:" <<
q.lastError().text(); return false; }
#define EXEC_QUERY_ROLLBACK_ERROR(db, q) if (!q.exec())
{ qWarning() << "QSqlQuery error:" <<
q.lastError().text(); db.rollback(); return false; }

#define JSON_STRINGIFY(v) v.engine()-
>evaluate(QString("JSON.stringify")).call(QJSValueList { v
}).toString();
#define JSON_PARSE(e, v) e-
>evaluate(QString("JSON.parse")).call(QJSValueList { v });

DbManager* DbManager::sm_pInstance;

DbManager::DbManager(QObject *parent) : QObject(parent)
{
    m_db = QSqlDatabase::addDatabase("QSQLITE");

    m_db.setDatabaseName(QStandardPaths::writableLocation(QSta
ndardPaths::DataLocation)+DATABASE_NAME);
    qDebug() << "database name:" << m_db.databaseName();

    if (!m_db.open()) {
        qWarning() << "Database: connection error: " <<
m_db.lastError().text();
        exit(-1);
        return;
    }

    qDebug() << "Database: connection ok";

    _init();
}

DbManager::~DbManager()
{
    qDebug() << "Close database ...";
    if (m_db.isOpen()) {
        QSqlQuery query(m_db);
```

```

        query.prepare("SELECT count(*) FROM Messages WHERE
serverId IS NULL OR serverId = '' OR serverId =
'undefined'");
        if (!query.exec()) { qWarning() << "QSqlQuery
error:" << query.lastError().text(); return; }

        while (query.next()) {
            qDebug() << "Unsent messages =" <<
query.value(0).toInt();
        }

        m_db.close();
        qDebug() << " ... closed";
    } else {
        qDebug() << " ... was not open";
    }
}

bool DbManager::isOpen() const
{
    return m_db.isOpen();
}

bool DbManager::_init()
{
    QSqlQuery query(m_db);

    int dbver = -1;
    if (query.exec("PRAGMA user_version")) {
        if (query.next()) {
            bool ok;
            int v = query.value(0).toInt(&ok);
            if (ok)
                dbver = v;
        }
    }

    if (dbver != DATABASE_VERSION) {
        qDebug() << "Current DB version:" << dbver <<
"Actual DB version:" << DATABASE_VERSION << "-- recreate
database!";

        query.prepare("PRAGMA encoding='UTF-16'");
        EXEC_QUERY_CATCH_ERROR(query);

        query.prepare("DROP TABLE IF EXISTS Messages");
        EXEC_QUERY_CATCH_ERROR(query);

        query.prepare("DROP TABLE IF EXISTS Chats");
        EXEC_QUERY_CATCH_ERROR(query);

        query.prepare("CREATE TABLE IF NOT EXISTS Chats ("

```

```

        "id TEXT PRIMARY KEY,"
        "type INT NOT NULL,"
        "name TEXT NOT NULL,"
        "chatModel TEXT NOT NULL,"
        "timestamp BIGINT,"
        "pinned BIGINT,"
        "mediaInfo TEXT ALLOW NULL DEFAULT
NULL,"
        "draft TEXT ALLOW NULL DEFAULT
NULL)");
    EXEC_QUERY_CATCH_ERROR(query);

    query.prepare("CREATE TABLE IF NOT EXISTS Messages
("
        "localId TEXT,"
        "serverId TEXT,"
        "chatId TEXT NOT NULL,"
        "content TEXT,"
        "timestamp BIGINT NOT NULL,"
        "messageModel TEXT,"
        "PRIMARY KEY (chatId, localId,
serverId)");
    EXEC_QUERY_CATCH_ERROR(query);

    query.prepare("CREATE TABLE IF NOT EXISTS
DelayedCommands ("
        "id INTEGER PRIMARY KEY
AUTOINCREMENT,"
        "command TEXT)");
    EXEC_QUERY_CATCH_ERROR(query);

    query.prepare(QString("PRAGMA user_version =
%1").arg(DATABASE_VERSION));
    EXEC_QUERY_CATCH_ERROR(query);
}

return true;
}

bool DbManager::clear()
{
    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);
    query.prepare("DELETE FROM Messages");
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

    query.prepare("DELETE FROM Chats");
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

    COMMIT_TRANSACTION(m_db);
}

```

```

        return true;
    }

bool DbManager::upsertChat(QJSValue model)
{
    QString roomId = model.property("roomId").toString();
    int type = model.property("chatType").toInt();
    QString name = model.property("chatName").toString();
    bool ok;
    qint64 timestamp =
model.property("lastMessageTimestamp").toString().toLongLo
ng();
    qint64 pinned =
model.property("pinned").toString().toLongLong(&ok);
    if (ok && pinned && pinned > timestamp) {
        timestamp = pinned;
    }
    QString chatModel = JSON_STRINGIFY(model);

    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);

    query.prepare("UPDATE Chats SET type = :type, name
= :name, chatModel = :model, timestamp = :ts, pinned =
:pinned WHERE id = :id");
    query.bindValue(":type", type);
    query.bindValue(":name", name);
    query.bindValue(":model", chatModel);
    query.bindValue(":ts", timestamp);
    query.bindValue(":pinned", pinned);
    query.bindValue(":id", roomId);
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

    if (query.numRowsAffected() == 0)
    {
        query.prepare("INSERT OR IGNORE INTO Chats VALUES
(:id, :type, :name, :model, :ts, :pinned, null, null)");
        query.bindValue(":id", roomId);
        query.bindValue(":type", type);
        query.bindValue(":name", name);
        query.bindValue(":model", chatModel);
        query.bindValue(":ts", timestamp);
        query.bindValue(":pinned", pinned);
        EXEC_QUERY_ROLLBACK_ERROR(m_db, query);
    }

    COMMIT_TRANSACTION(m_db);

    return query.numRowsAffected() > 0;
}

```

```

bool DbManager::removeChat(const QString& roomId)
{
    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);

    query.prepare("DELETE FROM Messages WHERE chatId =
:id");
    query.bindValue(":id", roomId);
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

    query.prepare("DELETE FROM Chats WHERE id = :id");
    query.bindValue(":id", roomId);
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

    COMMIT_TRANSACTION(m_db);

    return true;
}

bool DbManager::upsertMessage(
    const QString& serverId,
    const QString& localId,
    const QString& roomId,
    const QVariant& timestamp,
    const QString& content,
    const QString& messageModel)
{
    //qDebug() << "roomId=" << roomId << "serverId=" <<
serverId << "localId=" << localId;

    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);

    query.prepare("UPDATE Messages SET serverId =
:serverId, localId = :localId, content = :content,
timestamp = :ts, messageModel = :model"
        " WHERE chatId = :roomId AND localId
= :localId");

        //" WHERE (serverId = :serverId OR
serverId IS NULL OR serverId = '' OR serverId =
'undefined') AND localId = :localId"); // А такой вариант
не работает, нужно еще искать где ошибка
    query.bindValue(":serverId", serverId);
    query.bindValue(":localId", localId);
    query.bindValue(":content", content);
    query.bindValue(":ts", timestamp);
    query.bindValue(":model", messageModel);
    query.bindValue(":roomId", roomId);
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);
}

```

```

        //qDebug() << "upsertMessage: update" <<
        query.numRowsAffected();

        if (query.numRowsAffected() == 0)
        {
            query.prepare("INSERT OR IGNORE INTO Messages
(localId, serverId, chatId, content, timestamp,
messageModel) VALUES (:localId, :serverId, :roomId,
:content, :ts, :model)");
            query.bindValue(":localId", localId);
            query.bindValue(":serverId", serverId);
            query.bindValue(":roomId", roomId);
            query.bindValue(":content", content);
            query.bindValue(":ts", timestamp);
            query.bindValue(":model", messageModel);
            EXEC_QUERY_ROLLBACK_ERROR(m_db, query);
            //qDebug() << "upsertMessage: insert" <<
            query.numRowsAffected();
        }

        COMMIT_TRANSACTION(m_db);

        return query.numRowsAffected() > 0;
    }

bool DbManager::upsertServiceMessage(
    const QString& serverId,
    const QString& roomId,
    const QVariant& timestamp,
    const QString& content,
    const QString& messageModel)
{
    //qDebug() << "roomId=" << roomId << "serverId=" <<
    serverId << "localId=" << localId;

    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);

    query.prepare("UPDATE Messages SET localId = NULL,
content = :content, timestamp = :ts, messageModel =
:model"
                " WHERE chatId = :roomId AND serverId
= :serverId");
    query.bindValue(":serverId", serverId);
    query.bindValue(":content", content);
    query.bindValue(":ts", timestamp);
    query.bindValue(":model", messageModel);
    query.bindValue(":roomId", roomId);
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);
    //qDebug() << "upsertMessage: update" <<
    query.numRowsAffected();

```

```

        if (query.numRowsAffected() == 0)
        {
            query.prepare("INSERT OR IGNORE INTO Messages
(localId, serverId, chatId, content, timestamp,
messageModel) VALUES (NULL, :serverId, :roomId,
:content, :ts, :model)");
            query.bindValue(":serverId", serverId);
            query.bindValue(":roomId", roomId);
            query.bindValue(":content", content);
            query.bindValue(":ts", timestamp);
            query.bindValue(":model", messageModel);
            EXEC_QUERY_ROLLBACK_ERROR(m_db, query);
            //qDebug() << "upsertMessage: insert" <<
query.numRowsAffected();
        }

        COMMIT_TRANSACTION(m_db);

        return query.numRowsAffected() > 0;
    }

    // Модель уже содержит в себе часть извлеченных из
сообщения данных и
    // позволяет не выполнять некоторые "тяжелые" операции по
несколько раз
    // Получить модель из пришедшего от сервера сообщения
можно используя
    // функцию toMessageModel() из Utils.js
bool DbManager::upsertMessage(QJSValue model)
{
    QString serverId =
model.property("messageID").toString();
    QString localId =
model.property("localID").toString();
    QString roomId = model.property("roomId").toString();
    QVariant timestamp =
model.property("date").toString();
    QString content =
model.property("content").toString();
    QString messageModel = JSON_STRINGIFY(model);

    int messageType =
model.property("messageType").toInt();

    if(messageType == 6)
        return upsertServiceMessage(serverId, roomId,
timestamp, content, messageModel);
    else
        return upsertMessage(serverId, localId, roomId,
timestamp, content, messageModel);
}

```

```

bool DbManager::removeMessage(const QString& anyId)
{
    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);

    query.prepare("DELETE FROM Messages WHERE localId =
:id OR serverId = :id");
    query.bindValue(":id", anyId);
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

    COMMIT_TRANSACTION(m_db);

    return true;
}

bool DbManager::removeMessageInRoom(const QString& roomID,
const QString& anyId)
{
    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);

    query.prepare("DELETE FROM Messages WHERE (localId
= :id OR serverId = :id) AND chatId = :roomID");
    query.bindValue(":id", anyId);
    query.bindValue(":roomID", roomID);
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

    COMMIT_TRANSACTION(m_db);

    return true;
}

int DbManager::getChatCount()
{
    QSqlQuery query(m_db);

    query.prepare("SELECT count(*) FROM Chats");
    EXEC_QUERY_CATCH_ERROR(query);

    if (query.next()) {
        bool ok;
        int v = query.value(0).toInt(&ok);
        if (ok)
            return v;
    }

    return -1;
}

```

```

// ВАЖНО! Чаты из БД должны приходить отсортированные
таким образом, чтобы прикрепленные были среди первых
int DbManager::getAllChats(QJSValue jsCallback)
{
    if (!jsCallback.isCallable())
        return -1;

    QSqlQuery query(m_db);

    query.prepare("SELECT chatModel FROM Chats ORDER BY
pinned DESC, timestamp DESC, name ASC");
    EXEC_QUERY_CATCH_ERROR(query);

    int result = 0;
    while (query.next()) {
        auto val = JSON_PARSE(jsCallback.engine(),
QJSValue(query.value(0).toString()));
        jsCallback.call(QJSValueList { val });
        ++result;
    }

    return result;
}

int DbManager::getPinnedChats(QJSValue jsCallback)
{
    if (!jsCallback.isCallable())
        return -1;

    QSqlQuery query(m_db);

    query.prepare("SELECT chatModel FROM Chats WHERE
pinned > 0 ORDER BY pinned DESC, name ASC");
    EXEC_QUERY_CATCH_ERROR(query);

    int result = 0;
    while (query.next()) {
        auto val = JSON_PARSE(jsCallback.engine(),
QJSValue(query.value(0).toString()));
        jsCallback.call(QJSValueList { val });
        ++result;
    }

    return result;
}

// ВАЖНО: Сообщения возвращаются от более свежих к более
старым.
int DbManager::getMessages(const QString& roomId, QJSValue
jsCallback, int limit, int offset)
{
    if (!jsCallback.isCallable())

```

```

        return -1;

        QSqlQuery query(m_db);

        QString l = (limit > 0) ? QString(" LIMIT
%1").arg(limit) : QString("");
        QString o = (offset > 0) ? QString(" OFFSET
%1").arg(offset) : QString("");
        QString queryText("SELECT messageModel FROM Messages
WHERE chatId = :roomID ORDER BY timestamp DESC");
        queryText.append(l);
        queryText.append(o);

        query.prepare(queryText);
        query.bindValue(":roomID", roomID);
        EXEC_QUERY_CATCH_ERROR(query);

        int result = 0;
        while (query.next()) {
            auto val = JSON_PARSE(jsCallback.engine(),
QJSValue(query.value(0).toString()));
            jsCallback.call(QJSValueList { val });
        }
        ++result;

        return result;
}

bool DbManager::countMessagesInChats(QJSValue jsCallback)
{
    if (!jsCallback.isCallable())
        return false;

    QSqlQuery query(m_db);

    query.prepare("SELECT chatId, count(*) FROM Messages
GROUP BY chatId");
    EXEC_QUERY_CATCH_ERROR(query);

    while (query.next()) {
        auto roomID = QJSValue(query.value(0).toString());
        auto count = QJSValue(query.value(1).toInt());
        jsCallback.call(QJSValueList { roomID, count });
    }

    return true;
}

int DbManager::getMessagesCount(const QString& roomID)
{
    QSqlQuery query(m_db);

```

```

        query.prepare("SELECT count(*) FROM Messages WHERE
chatId = :id");
        query.bindValue(":id", roomId);

        if (query.exec()) {
            if (query.next())
                return query.value(0).toInt();
        } else
            qWarning() << "QSqlQuery error:" <<
query.lastError().text();

        return -1;
    }

int DbManager::getUnsentMessages(QJSValue jsCallback)
{
    if (!jsCallback.isCallable())
        return -1;

    QSqlQuery query(m_db);

    query.prepare("SELECT messageModel, chatId, serverId,
localId FROM Messages"
                  " WHERE serverId IS NULL OR serverId =
'' OR serverId = 'undefined' ORDER BY timestamp ASC");
    EXEC_QUERY_CATCH_ERROR(query);

    int result = 0;
    while (query.next()) {
        qDebug() << "roomId=" << query.value(1).toString()
<< "serverId=" << query.value(2).toString() << "localId="
<< query.value(3).toString();

        auto val = JSON_PARSE(jsCallback.engine(),
QJSValue(query.value(0).toString()));
        jsCallback.call(QJSValueList { val });
        QThread::msleep(1000); // Гарантируем отличие
времени отправки сообщений, если оно будет устанавливаться
сервером
    }
    ++result;

    return result;
}

int DbManager::search(const QString& text, QJSValue
jsCallback)
{
    if (!jsCallback.isCallable())
        return -1;

    QString t("%"+text+"%");

```

```

t.replace(' ', '%');

QDebug() << "search" << t;

 QSqlQuery query(m_db);

    query.prepare("SELECT messageModel FROM Messages WHERE
content LIKE :text ORDER BY chatId ASC, timestamp ASC");
    query.bindValue(":text", t);
    EXEC_QUERY_CATCH_ERROR(query);

    int result = 0;
    while (query.next()) {
        auto val = JSON_PARSE(jsCallback.engine(),
QJSValue(query.value(0).toString()));
        jsCallback.call(QJSValueList { val });
    }
    ++result;

    return result;
}

bool DbManager::updateMediaInfo(const QString& roomId,
const QString& mediaInfo)
{
    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);

    query.prepare("UPDATE Chats SET mediaInfo = :media
WHERE id = :id");
    query.bindValue(":media", mediaInfo);
    query.bindValue(":id", roomId);
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

    COMMIT_TRANSACTION(m_db);

    return query.numRowsAffected() > 0;
}

QString DbManager::getMediaInfo(const QString& roomId)
{
    QSqlQuery query(m_db);

    query.prepare("SELECT mediaInfo FROM Chats WHERE id
= :id");
    query.bindValue(":id", roomId);
    if (query.exec()) {
        if (query.next())
            if (!query.value(0).isNull())
                return query.value(0).toString();
    } else

```

```

        qWarning() << "QSqlQuery error:" <<
query.lastError().text();

        return
QString("{\\"chatCount\\":0,\\"audioCount\\":0,\\"videoCount\\":
0,\\"photoCount\\":0,\\"filesCount\\":0}");
}

bool DbManager::setDraft(const QString& roomId, const
QString& draft)
{
    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);

    query.prepare("UPDATE Chats SET draft = :draft WHERE
id = :id");
    query.bindValue(":draft", draft);
    query.bindValue(":id", roomId);
    EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

    COMMIT_TRANSACTION(m_db);

    return query.numRowsAffected() > 0;
}

QString DbManager::getDraft(const QString& roomId)
{
    QSqlQuery query(m_db);

    query.prepare("SELECT draft FROM Chats WHERE id =
:id");
    query.bindValue(":id", roomId);

    if (query.exec()) {
        if (query.next())
            return query.value(0).toString();
    } else
        qWarning() << "QSqlQuery error:" <<
query.lastError().text();

    return QString("");
}

bool DbManager::addDelayedCommand(const QString& cmd)
{
    BEGIN_TRANSACTION(m_db);

    QSqlQuery query(m_db);

    query.prepare("INSERT INTO DelayedCommands (command)
VALUES (:cmd)");

```

```

        query.bindValue(":cmd", cmd);
        EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

        COMMIT_TRANSACTION(m_db);

        return true;
    }

    int DbManager::getDelayedCommands(QJSValue jsCallback)
    {
        if (!jsCallback.isCallable())
            return -1;

        QSqlQuery query(m_db);

        query.prepare("SELECT id, cmd FROM DelayedCommands
ORDER BY id ASC");
        EXEC_QUERY_CATCH_ERROR(query);

        int result = 0;
        while (query.next()) {
            jsCallback.call(QJSValueList
{ QJSValue(query.value(0).toInt()),
QJSValue(query.value(1).toString()) });
        }
        ++result;

        return result;
    }

    bool DbManager::removeDelayedCommand(int id)
    {
        BEGIN_TRANSACTION(m_db);

        QSqlQuery query(m_db);

        query.prepare("DELETE FROM DelayedCommands WHERE id
= :id");
        query.bindValue(":id", id);
        EXEC_QUERY_ROLLBACK_ERROR(m_db, query);

        COMMIT_TRANSACTION(m_db);

        return true;
    }

```

Приложение В Листинг класса RestApi

Файл «restapi.h»

```
#ifndef RESTAPI_H
#define RESTAPI_H

#include <QObject>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QDebug>
#include <QJsonObject>
#include <QJsonArray>
#include <QJsonDocument>
#include <QJsonValue>
#include <QJsonParseError>

class RestApi : public QObject
{
    Q_OBJECT

public:
    explicit RestApi(QObject *parent = Q_NULLPTR);
    virtual ~RestApi();

    Q_PROPERTY(QString token READ getToken WRITE setToken
NOTIFY changeToken)

    inline QString getToken() const { return m_token; }
    inline void setToken(const QString& token) { m_token =
token; emit changeToken(token); }

Q_SIGNALS:
    // Формируется при изменении значения токена
    void changeToken(const QString& token);
    // Формируется в случае возникновения сетевой ошибки
    void networkFailed(int networkErrorCode, const
QString& networkErrorText);

protected:
    void get(const QString& url);
    void post(const QString& url, const QString& content);
    void put(const QString& url, const QString& content);
    void del(const QString& url);

    // Вызывается при успешном получении ответа от сервера
    virtual void finished(const QUrl& url, int httpStatus,
const QByteArray& replyData) = 0;

private slots:
```

```

        // Обработчик сигнала завершения запроса от
        QNetworkAccessManager
        void requestFinished(QNetworkReply* reply);

protected:
    // Token аутентификации, передаваемый на сервер с
    каждым запросом
    QString m_token;

private:
    QNetworkAccessManager* m_manager;
};

#endif // RESTAPI_H

```

Файл «restapi.cpp»

```

#include "restapi.h"

RestApi::RestApi(QObject *parent) : QObject(parent)
{
    m_manager = new QNetworkAccessManager();
    connect(m_manager, SIGNAL(finished(QNetworkReply*)),
this, SLOT(requestFinished(QNetworkReply*)));
}

RestApi::~RestApi()
{
    m_manager->deleteLater();
}

void RestApi::get(const QString& url)
{
    QUrl _url(url);
    QNetworkRequest request(_url);

    request.setRawHeader(QByteArray("access-token"),
m_token.toLocal8Bit());

    qDebug() << "----- GET:" << _url;

    QNetworkReply* reply = m_manager->get(request);
    reply->setParent(m_manager); // delete the reply with
the m_manager
}

void RestApi::post(const QString& url, const QString&
content)
{
    QUrl _url(url);
    QNetworkRequest request(_url);
}

```

```

        request.setHeader(QNetworkRequest::ContentTypeHeader,
"application/json; charset=utf-8");
        if (!m_token.isEmpty())
            request.setRawHeader(QByteArray("access-token"),
m_token.toLocal8Bit());

        qDebug() << "----- POST:" << _url;

        QNetworkReply* reply = m_manager->post(request,
content.toUtf8());
        reply->setParent(m_manager); // delete the reply with
the m_manager
    }

void RestApi::put(const QString &url, const QString
&content) {
    QUrl _url(url);
    QNetworkRequest request(_url);

    request.setHeader(QNetworkRequest::ContentTypeHeader,
"application/json; charset=utf-8");
    if (!m_token.isEmpty())
        request.setRawHeader(QByteArray("access-token"),
m_token.toLocal8Bit());

    qDebug() << "----- PUT:" << _url;

    QNetworkReply* reply = m_manager->put(request,
content.toUtf8());
    reply->setParent(m_manager); // delete the reply with
the m_manager
}

void RestApi::del(const QString &url) {
    QUrl _url(url);
    QNetworkRequest request(_url);

    request.setRawHeader(QByteArray("access-token"),
m_token.toLocal8Bit());

    qDebug() << "----- DELETE:" << _url;

    QNetworkReply* reply = m_manager-
>deleteResource(request);
    reply->setParent(m_manager); // delete the reply with
the m_manager
}

void RestApi::requestFinished(QNetworkReply* reply)
{

```

```

        int httpStatus = reply-
>attribute(QNetworkRequest::HttpStatusCodeAttribute).toInt
();

        //qDebug() << "----- Finished:" << reply->url()
<< httpStatus;
        if (reply->error() > 0) {
            //qDebug() << "-----          :" << reply-
>error() << reply->errorString();
            emit networkFailed(reply->error(), reply-
>errorString());
        } else {
            QByteArray replyData = reply->readAll();
            finished(reply->url(), httpStatus, replyData);
        }

        reply->deleteLater();
    }

```

Приложение С Листинг класса «QMLRestApi»

Файл «qmlrestapi.h»

```
#ifndef UCALLROOMSAPI_H
#define UCALLROOMSAPI_H

#include <QObject>
#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>
#include "restapi.h"

#define STRINGIFY(x) #x
#define TOSTRING(x) STRINGIFY(x)

class UcallRestApi : public RestApi
{
    Q_OBJECT

public:
    explicit UcallRestApi(QObject *parent = Q_NULLPTR);

    Q_PROPERTY(QString baseUrl READ getBaseUrl WRITE
setBaseUrl NOTIFY changeBaseUrl)

    inline QString getBaseUrl() const { return
m_baseUrl; }
    inline void setBaseUrl(const QString& baseUrl) { if
(m_baseUrl != baseUrl) { m_baseUrl = baseUrl; emit
changeBaseUrl(baseUrl); } }

Q_SIGNALS:
    // Формируется при изменении базового адреса
    void changeBaseUrl(const QString& baseUrl);
    // Формируется по окончании извлечения и
предварительно обработки данных от сервера
    void fetchComplete();
    // Формируется, если сервер вернул ошибку внутри
ответа на запрос
    void apiError(int code);
    // Формируется, если не удалось представить ответ
сервера в виде JSON объекта
    void jsonParseFailed();
    // Формируется по окончании разбора данных, полученных
от сервера
    void parseComplete();

protected:
    // Вызывается после успешного получения данных от
сервера и перед передачей их в обработчик сигнала
requestSuccess
```

```
        virtual void finished(const QUrl& url, int httpStatus,
const QByteArray& replyData);
        // Вызывается после успешного анализа заголовка ответа
и предназначен для разбора содержимого ответа
        virtual void parseData(const QUrl& url, const
QJsonObject& data) = 0;
        // Проверяет корректность JSON элемента
        bool isValidElement(QJsonObject source, const QString&
name, QJsonValue::Type type);
```

```
protected:
    QString m_baseUrl;
};
```

```
#endif // UCALLROOMSAPI_H
```

Файл «qmlrestapi.cpp»

```
#include "qmlrestapi.h"
```

```
QmlRestApi::QmlRestApi(QObject *parent) : RestApi(parent)
{
}
```

```
QmlRestApi::~~QmlRestApi()
{
}
```

```
void QmlRestApi::finished(const QUrl& url, int httpStatus,
const QByteArray& replyData)
{
    emit requestSuccess(url.toString(), httpStatus,
QString(replyData));
}
```

Приложение D Листинг класса «HttpMultipart»

Файл «httpmultipart.h»

```
#ifndef HTTPMULTIPART_H
#define HTTPMULTIPART_H

#include <QObject>
#include <QFileInfo>
#include <QStandardPaths>
#include <QMimeDatabase>
#include <QHttpMultiPart>
#include <QNetworkReply>
#include <QEventLoop>
#include <QDebug>
#include <QByteArray>
#include <QJsonArray>
#include <QJsonObject>
#include <QJsonValue>
#include <QJsonDocument>

class HttpMultipart : public QObject
{
    Q_OBJECT

public:
    explicit HttpMultipart(QObject *parent = Q_NULLPTR);
    virtual ~HttpMultipart();

    Q_INVOKABLE void request(const QString& url, const
QString& fileName, const QString &token);
    Q_INVOKABLE void resetAvatar(const QString& url, const
QString &token);
    Q_INVOKABLE void createGroup(const QString& url, const
QString &name, const QString &description, const QString
&usersList, const QString &adminList, int readOnly, const
QString &token);

    Q_SIGNALS:
        void progress(qint64 done, qint64 total);
        void error(int code, const QString& errorString);
        void finished(const QString& response);

public slots:

private slots:
    void uploadProgress(qint64 done, qint64 total);
    void uploadFinished();

private:
    QNetworkAccessManager* m_manager;
    QNetworkReply* m_reply;
```

```

        QFile* m_file;
    };

#endif // HTTPMULTIPART_H
Файл «httpmultipart.cpp»

#include "httpmultipart.h"

HttpMultipart::HttpMultipart(QObject *parent) :
    QObject(parent)
{
}

HttpMultipart::~HttpMultipart()
{
}

void HttpMultipart::request(const QString& url, const
    QString& fileName, const QString &token)
{
    QString nameOnServer(fileName);
    int lastSlash = nameOnServer.lastIndexOf("/") + 1;
    nameOnServer.remove(0, lastSlash);

    QMimeDatabase db;
    QMimeType type = db.mimeTypeForFile(fileName);

    QHttpMultiPart *multiPart = new
    QHttpMultiPart(QHttpMultiPart::FormDataType);
    QHttpPart imagePart;

    imagePart.setHeader(QNetworkRequest::ContentTypeHeader,
        type.name());

    imagePart.setHeader(QNetworkRequest::ContentDispositionHeader,
        QVariant("form-data; name=\"file\"; filename=\""+nameOnServer+"\""));

    QHttpPart textPart;

    textPart.setHeader(QNetworkRequest::ContentDispositionHeader,
        QVariant("form-data; name=\"name\""));
    textPart.setBody(nameOnServer.toUtf8());

    QFile *file = new QFile(fileName);
    file->open(QIODevice::ReadOnly);
    imagePart.setBodyDevice(file);
    file->setParent(multiPart); // we cannot delete the
    file now, so delete it with the multiPart

    multiPart->append(textPart);
    multiPart->append(imagePart);
}

```

```

        QUrl _url(url);
        QNetworkRequest request(_url);
        request.setRawHeader(QByteArray("access-token"),
QByteArray(token.toLocal8Bit()));
        m_manager = new QNetworkAccessManager();
        m_reply = m_manager->post(request, multiPart);
        multiPart->setParent(m_reply); // delete the multiPart
with the reply

        connect(m_reply, SIGNAL(finished()), this,
SLOT(uploadFinished()));
        connect(m_reply, SIGNAL(uploadProgress(qint64,
qint64)), this, SLOT(uploadProgress(qint64, qint64)));
    }

void HttpMultipart::resetAvatar(const QString& url, const
QString &token)
{
    QHttpMultipart *multiPart = new
QHttpMultipart(QHttpMultipart::FormDataType);
    QUrl _url(url);
    QNetworkRequest request(_url);

    request.setRawHeader(QByteArray("access-token"),
QByteArray(token.toLocal8Bit()));
    m_manager = new QNetworkAccessManager();
    m_reply = m_manager->post(request, multiPart);
    multiPart->setParent(m_reply);

    connect(m_reply, SIGNAL(finished()), this,
SLOT(uploadFinished()));
}

void HttpMultipart::createGroup(const QString& url, const
QString &name, const QString &description, const QString
&usersList, const QString& adminList, int readOnly, const
QString &token)
{
    QHttpMultipart *multiPart = new
QHttpMultipart(QHttpMultipart::FormDataType);
    QHttpPart part;

    part.setHeader(QNetworkRequest::ContentDispositionHeader,
QVariant("form-data; name=\"name\""));
    part.setBody(name.toUtf8());
    multiPart->append(part);

    if (description != "")
    {
        part.setHeader(QNetworkRequest::ContentDispositionHeader,
QVariant("form-data; name=\"description\""));
    }

```

```

        part.setBody(description.toUtf8());
        multiPart->append(part);
    }

    QFile *file = new
    QFile(QStandardPaths::writableLocation(QStandardPaths::TempLocation)+"/avatar.jpg");
    if(file->open(QIODevice::ReadOnly))
    {
        QMimeDatabase db;
        QMimeType type =
        db.mimeTypeForFile(QStandardPaths::writableLocation(QStandardPaths::TempLocation)+"/avatar.jpg");
        QHttpPart imagePart;

        imagePart.setHeader(QNetworkRequest::ContentTypeHeader,
        type.name());

        imagePart.setHeader(QNetworkRequest::ContentDispositionHeader,
        QVariant("form-data; name=\"file\"; filename=\"avatar.jpg\""));
        imagePart.setBodyDevice(file);
        file->setParent(multiPart);
        multiPart->append(imagePart);
    }

    part.setHeader(QNetworkRequest::ContentDispositionHeader,
    QVariant("form-data; name=\"users\""));
    part.setBody(usersList.toUtf8());
    multiPart->append(part);

    if (adminList != "")
    {
        part.setHeader(QNetworkRequest::ContentDispositionHeader,
        QVariant("form-data; name=\"administrators\""));
        part.setBody(adminList.toUtf8());
        multiPart->append(part);
    }

    part.setHeader(QNetworkRequest::ContentDispositionHeader,
    QVariant("form-data; name=\"readOnly\""));
    //part.setBody(QString::number(readOnly).toUtf8());
    part.setBody(QByteArray::number(readOnly));
    multiPart->append(part);

    QUrl _url(url);
    QNetworkRequest request(_url);
    request.setRawHeader(QByteArray("access-token"),
    QByteArray(token.toLocal8Bit()));

```

```

    m_manager = new QNetworkAccessManager();
    m_reply = m_manager->post(request, multiPart);
    multiPart->setParent(m_reply);

    connect(m_reply, SIGNAL(finished()), this,
    SLOT(uploadFinished()));
    connect(m_reply, SIGNAL(uploadProgress(qint64,
    qint64)), this, SLOT(uploadProgress(qint64, qint64)));
}

void HttpMultipart::uploadProgress(qint64 done, qint64
total)
{
    //qDebug() << "-----Upload part-----" <<
done << "/" << total;
    emit progress(done, total);
}

void HttpMultipart::uploadFinished()
{
    int httpStatus = m_reply-
>attribute(QNetworkRequest::HttpStatusCodeAttribute).toInt
();

    //qDebug() << "-----Finished-----" <<
httpStatus;

    if (m_reply->error() > 0)
    {
        emit error(httpStatus, m_reply->errorString());
    }
    else
    {
        QString data = QString(m_reply->readAll());
        //qDebug() << "Upload finished" << data;
        emit finished(data);
    }

    m_reply->deleteLater();
    m_manager->deleteLater();
}

```

Приложение С Презентация

Клиент обмена текстовыми и медиа сообщениями для мобильной платформы «Аврора»

Автор работы: студент группы 16BB2 Родионов В.С.

Научный руководитель: к.т.н., доцент кафедры ВТ Дубравин А.В.

Пенза, 2020

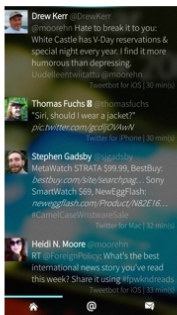
Цель и объект работы

Цель работы: разработка клиента системы мгновенного обмена данными для мобильной платформы «Аврора»

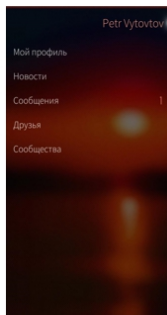
Объектом разработки является приложение, обеспечивающее следующий функционал: авторизация пользователя в системе, организация взаимодействия между пользователями или группой пользователей в виде обеспечения возможности передачи данных различного типа.

Необходимо обеспечить пользователю удобный графический интерфейс приложения, хранение личной информации пользователя в локальной базе данных, а также обеспечить механизмы сетевого взаимодействия в рамках архитектуры «клиент-сервер»

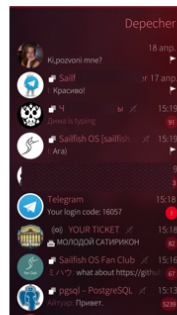
Клиенты IMS-сетей для платформы «Аврора»



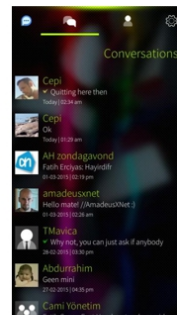
Tweetian



Kat



Depecher



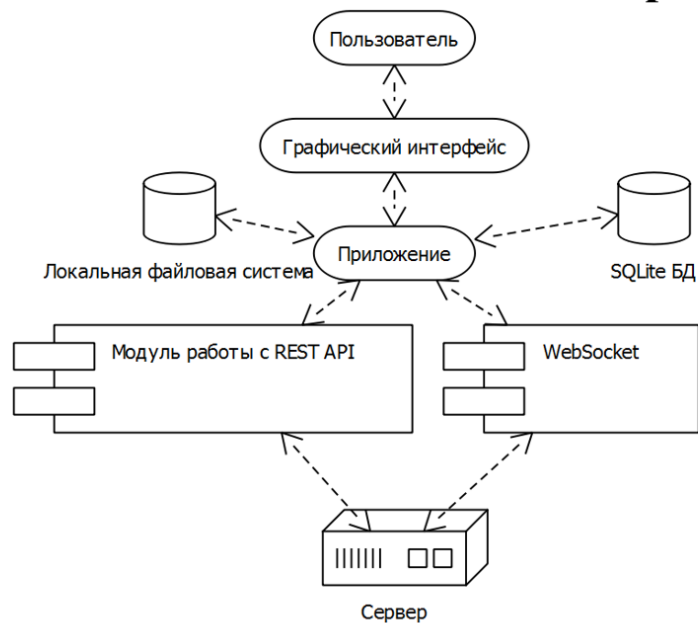
Whatsup

Стек технологий



Дополнительные инструменты: SQLite, REST, HTTP, WebSocket

Схема взаимодействия компонентов приложения



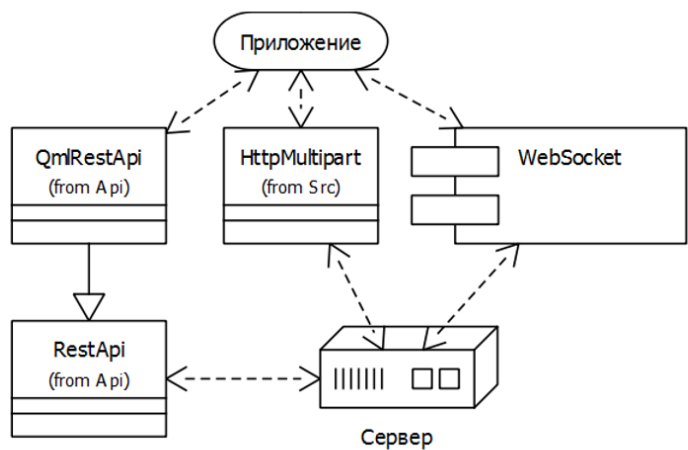
Графический интерфейс пользователя



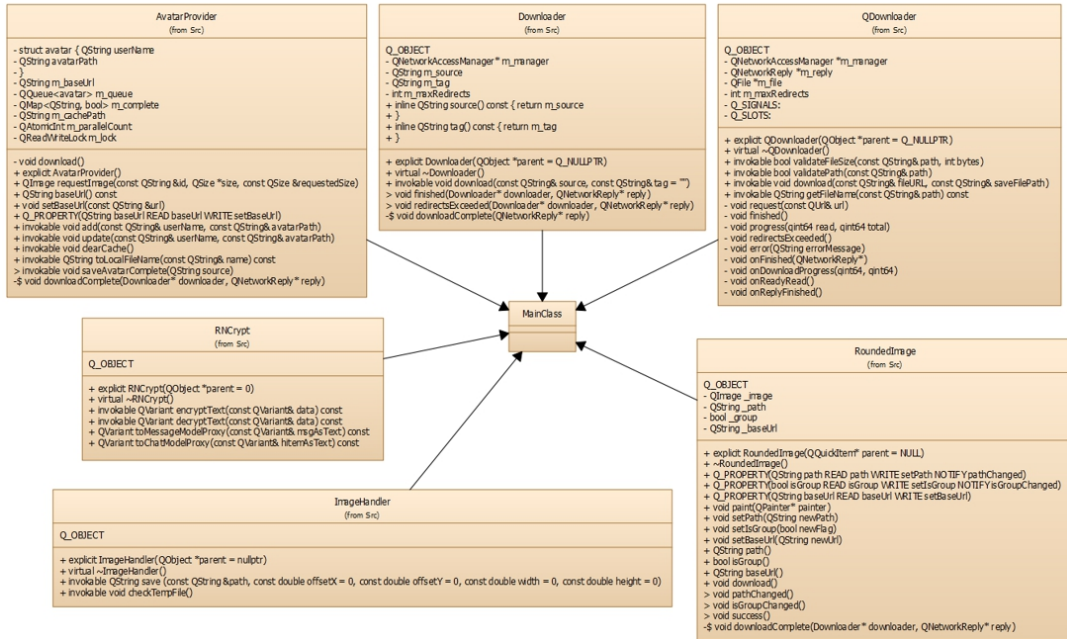
Структура локальной базы данных

Chats		Messages	
id	TEXT	LocalId	TEXT
Type	INTEGER	ServerId	TEXT
Name	TEXT	ChatId	TEXT
ChatModel	TEXT	Content	TEXT
Timestamp	INTEGER	Timestamp	INTEGER
Pinned	INTEGER	MessageModel	TEXT
MediaInfo	TEXT		
Draft	TEXT		

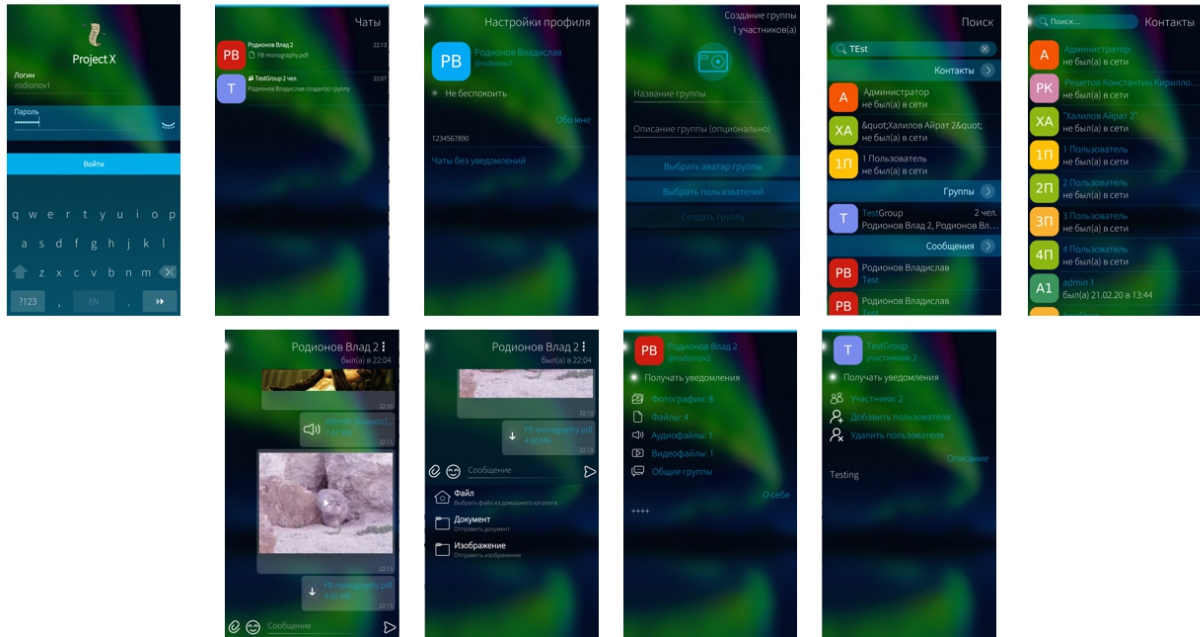
Сетевое взаимодействие



C++ классы



Работа приложения



Результаты работы

- 1) Проведено аналитическое исследование предметной области, изучены уже реализованные системы мгновенного обмена данных для целевой платформы, а также технические средства разработки программного обеспечения для платформы «Аврора».
- 2) Разработан удобный и интуитивно понятный графический интерфейс для конечного приложения.
- 3) Разработана локальная база данных для приложения с использованием СУБД SQLite и средства взаимодействия с ней.
- 4) Разработаны механизмы сетевого взаимодействия посредством протоколов TCP/IP и WebSocket.
- 5) Разработано и отлажено мобильное приложение, которое предоставляет функционал взаимодействия между пользователями или группой пользователей общей системы в виде передачи данных различного типа, включая текстовые данные и медиа данные.

Спасибо за внимание!