содержание

Введение

Вычислительная техника является важнейшим компонентом процесса вычислений и обработки данных. Первыми приспособлениями для вычислений были, вероятно, всем известные счётные палочки, которые и сегодня используются в начальных классах многих школ для обучения счёту. Развиваясь, эти приспособления становились более сложными, например, такими как финикийские глиняные фигурки, также предназначаемые для наглядного представления количества считаемых предметов. Такими приспособлениями, пользовались торговцы и счетоводы того времени.

На первых компьютерах приходилось программировать двоичными машинными кодами. Однако программировать таким образом – достаточно трудоемкая и сложная задача. Для упрощения этой задачи стали появляться языки программирования низкого уровня, которые позволяли задавать машинные команды в более понятном для человека виде.

Язык программирования предназначен для написания компьютерных про-грамм, которые представляют собой набор правил, позволяющих компьютеру выполнить тот или иной вычислительный процесс, организовать управление раз-личными объектами, и т. п. Язык программирования отличается от естественных языков тем, что предназначен для управления электронно-вычислительными ма-шинами, в то время как естественные языки используются, прежде всего, для общения людей между собой. Большинство языков программирования использует специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

Языки низкого уровня, как правило, используют для написания небольших системных программ, драйверов устройств, модулей стыков с нестандартным оборудованием, программирование специализированных микропроцессоров, когда важнейшими требованиями являются компактность, быстродействие и возможность прямого доступа к аппаратным ресурсам.

Программист, создающий алгоритм для компьютера на языке низкого уровня, обращаются непосредственно к ресурсам компьютера: процессору, памяти, периферийным устройствам. Это обеспечивает высокую скорость работы программ, поскольку, в отличие от высокоуровневых, в низкоуровневых языках отсутствуют скрытые фрагменты кода, добавляемые автоматически компилятором во время преобразования исходного текста в бинарный код.

В ходе развития программирования возникла необходимость разработки новых, более совершенных языков программирования, которые бы были схожи с естественными языками и позволяли бы не работать напрямую с машинными командами. Их стали называть языками высокого уровня. Языки высокого уровня ориентированы на описание алгоритмов, поэтому их называют алгоритмическими языками. Преимуществом таких языков является большая наглядность и независимость от конкретного компьютера. Разрабатывать программы на таких языках гораздо проще и ошибок допускается меньше. Значительно сокращается время разработки программы, что особенно важно при работе над большими программными проектами.

В рамках данного дипломного проекта планируется разработать мобильное приложение для обмена текстовыми сообщениями с возможностью передачи фотоизображений.

Разрабатываемое приложение будет использоваться прикладным программистом как готовый шаблон с целью доработки и переработки приложения или каких-то отдельных методов, а также сможет использовать готовые алгоритмы работы с базой данных, отправки сообщений, авторизации и регистрации в своих проектах.

Данная разработка даст возможность пользователю вести безопасное общение с людьми, а также использовать приложение как аналог другим мессенджерам с более простым и интуитивно понятным интерфейсом.

1 Исследовательский раздел

* 1. Анализ технического задания

Темой дипломного проекта является «Разработка мобильного приложения для обмена текстовыми сообщениями». Основной задачей является разработать мобильное приложение со следующим функционалом: регистрация и авторизация пользователя, поиск пользователя, создание диалога, отправка сообщения, отслеживание онлайн-статуса пользователя.

* 1. Поиск и анализ существующих аналогов программного продукта

Существует большое количество аналогов разрабатываемого приложения со своими достоинствами и недостатками. В качестве аналога будет рассмотрено приложение мессенджер “WhatsApp”.

WhatsApp – популярная система обмена мгновенными сообщениями. Позволяет пересылать текст, видео, фото и другие данные через специальные чаты. Из-за своего функционала считается отличной заменой сотовой связи, ибо используется всего лишь интернет-трафик.

К преимущества этого приложения можно отнести:

* WhatsApp является кроссплатформенным приложением, а также у него существует web-версия;
* приложение синхронизирует список контактов с телефонной книгой смартфона;
* сквозное шифрование, гарантирующее возможность прочтения сообщения только по телефонному номеру, на который они были отправлены;
* бесплатные аудио- и видеозвонки, которые можно делать посредством Интернет-соединения;
* отсутствие рекламы и быстрая работа приложения;
* информация о статусе сообщения, предоставляемая двойной галочкой, которая меняет свой цвет.

К недостаткам приложения относятся:

* в приложении отсутствуют стикеры;
* приложение не сильно в вопросах безопасности, посредством сохранения телефонного номера аккаунта на сервера;
* приложение может занимать много места на смартфоне, так как все мультимедиа файлы сохраняются в памяти, а не загружаются на сервера и отображаются.

По сравнению с аналогом разрабатываемое приложение будет иметь преимущества в хранении информации на облачном сервере и недостатки в виде отсутствия аудио- и видеозвонков, отслеживания статуса сообщения, синхронизации контактов, а также разрабатываемое приложение будет поддерживать только операционную систему Android.

Сравнительный анализ показал, что WhatsApp является более богатым по функционалу приложением, но в учебных целях разрабатываемое приложение полностью соответствует минимальным требованиям к приложению-мессенджеру.

* 1. Выбор среды разработки

Рынок мобильных приложений увеличивается фактически в геометрической прогрессии, каждый день появляются новые приложения и все большее количество людей начинает интересоваться, на чем их делают, как они создаются и сложно ли сделать собственный сервис на смартфон для бизнеса. Так же, как и для создания картинок, видеороликов или любых других продуктов, для создания приложений необходима собственная интегрированная среда разработки. Еще несколько лет назад самой популярной средой была бесплатная Eclipse, но сейчас достойную ей конкуренцию составляет сравнительно новая IDE Android Studio. Конечно, существует достаточное количество интегрированных сред разработки, различных фреймворков для разработки под операционную систему Android, но следует остановиться на двух вышеперечисленных кандидатах и рассмотреть их, так как они являются наиболее популярными инструментами по сравнению с аналогами.

Корпорация Google решила вплотную заняться разработкой собственной IDE, заточенной для работы с Android-приложениями. Разработчики в качестве основы для своей новой IDE взяли достаточно востребованную среду Intellij IDEA, доработали ее и внесли немалый список новшеств.

Android Studio поддерживает несколько языков программирования, включая С/С++ и Java, имеет встроенный эмулятор и большую библиотеку со всевозможными шаблонами и компонентами, существенно упрощающими и ускоряющими процесс разработки приложений.

Также в Android Studio можно создавать приложения для самых последних версий Android`a, а созданное приложение можно сразу же проверить на наличие ошибок, протестировать различными инструментами все элементы программы и заранее выявить все возможные проблемы в ее работе. Для тех, кто только начал делать первые шаги в сфере разработки мобильных приложений, существует достаточно удобная документация, которую можно скачать с официального сайта компании.

Благодаря встроенному эмулятору можно проводить тесты производительности и корректности работы разрабатываемых приложений на разных системах, и в случае необходимости проводить оптимизацию.

Вторая среда на рассмотрении – созданная организацией Eclipse Foundation. Бесплатная среда разработки Eclipse долгое время считалась основной, активно развивалась и приобрела огромную армию поклонников среди программистов. Разработчики компании регулярно улучшали Eclipse, выпуская обновления, создавая новые инструменты и внедряя полезные для программистов функции.

В этой среде можно подключать дополнительные модули, позволяющие расшить ее функционал. Например, для разработки проектов на языках С/С++ необходимо подключить модуль C/C++ Development Tools, а для написания программы на Java потребуется Java Development Tools. То есть, в зависимости от предпочтений и конкретных задач, можно подключать необходимые модули и дополнения, расширяя базовый функционал среды.

Для разработчиков из стран постсоветского пространства данная среда достаточно удобна благодаря русифицированному интерфейсу и документации, где можно найти очень много полезной информации. К достоинствам Eclipse можно добавить прекрасную работу даже на машинах со слабой конфигурацией, кроссплатформенность, полную техническую поддержку даже несмотря на то, что среда является бесплатной.

Конфигурацию Eclipse разработчик может формировать под свои потребности и на свое усмотрение, делая среду максимально удобной для себя. В ней прекрасно реализована функция групповой разработки, что по достоинству оценят группы программистов, работающих над одним проектом.

На основании всех вышеперечисленных особенностей и достоинств той или иной среды, можно составить сравнительный анализ кандидатов. Во-первых, хоть Android Studio и Eclipse имеют типичную автоотладку кода Java, но по сравнению с Eclipse в Android Studio отладка происходит немного быстрее.

Во-вторых, интерфейс Eclipse`a довольно громоздкий, как и подавляющее большинство IDE, в то время как в Android Studio рабочий процесс протекает более быстрее и легче.

Кроме того, в сравнении с Android Studio, Eclipse IDE является универсальной средой разработки, которая может быть использована с любым языком и платформой, а не только с Android.

Помимо всего прочего, если необходимо работать над несколькими проектами в Eclipse, то нужно объединить их в рабочую область. В попытке переключиться на другую рабочую область, приходится выбирать путь, после этого среда перезапускается, и это очень неудобно. С Android Studio таких проблем не возникает.

В конце концов, Eclipse – это программное обеспечение на основе Java и более крупная среда IDE по сравнению с Android Studio, поэтому для правильной работы требуется значительно большее количество ОЗУ с высокой скоростью процессора. Несоблюдение этого критерия вызывает сбои. С другой стороны, Android Studio не вызывает большого количества ошибок, и обеспечивает более стабильную производительность. Если на построение проекта в Eclipse уйдет 1-2 минуты, то в Android Studio это можно сделать в течение 30 секунд.

Таким образом, Android Studio заняла лидирующие позиции менее чем за год в качестве основной IDE для разработки приложений под Android и все больше разработчиков делают свой выбор в пользу данной IDE. На основании сравнительного анализа в качестве среды разработки дипломного проекта было выбрано Android Studio.

2 Конструкторский раздел

2.1 Разработка функционально-модульной схемы мобильного приложения

Ключевым компонентом для создания визуального интерфейса в приложении Android является Activity. Activity — это отдельный экран в Android. Это как окно в приложении для рабочего стола, или фрейм в программе на Java. Activity позволяет разместить все компоненты пользовательского интерфейса или виджеты на этом экране.

Приложение может иметь одну или несколько Activity. Например, при создании проекта с пустой Activity в проект по умолчанию добавляется один класс Activity - MainActivity, с которого и начинается работа приложения.

Большинство пользовательских взаимодействий приводит к изменению действующей Activity. Поэтому приложение много раз переключается между Activity за свой жизненный цикл. Необходимо связывать Activity друг с другом, если требуется чтобы одна Activity запускала другую.

На рисунке 2.1, а также на плакате ДП.0902.07.000000.01 ПЛ изображено представление клиентской части приложения в виде модульной схемы.

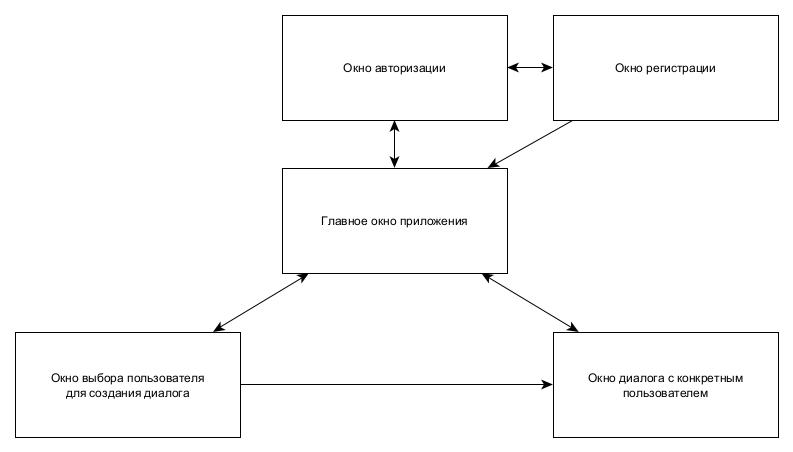


Рисунок 2.1 – Функционально-модульная схема клиентской части приложения

В состав модульной схемы включены следующие элементы:

* окно регистрации;
* окно авторизации;
* главное окно приложения, для просмотра всех диалогов;
* окно диалога с конкретным пользователем;
* окно выбора пользователя для создания диалога.

2.2 Разработка графического интерфейса мобильного приложения

Графический интерфейс приложения состоит из 5 окон: окна регистрации и авторизации, главное окно со списком диалогов, окно выбора пользователя для создания диалога, и непосредственно окно диалога с пользователем.

Окно авторизации содержит такие элементы интерфейса, как приветственную надпись, поля для ввода адреса электронной почты и пароля, кнопку входа в приложение, а также ссылку перехода на окно регистрации. Пример окна авторизации представлен на рисунке 2.2.

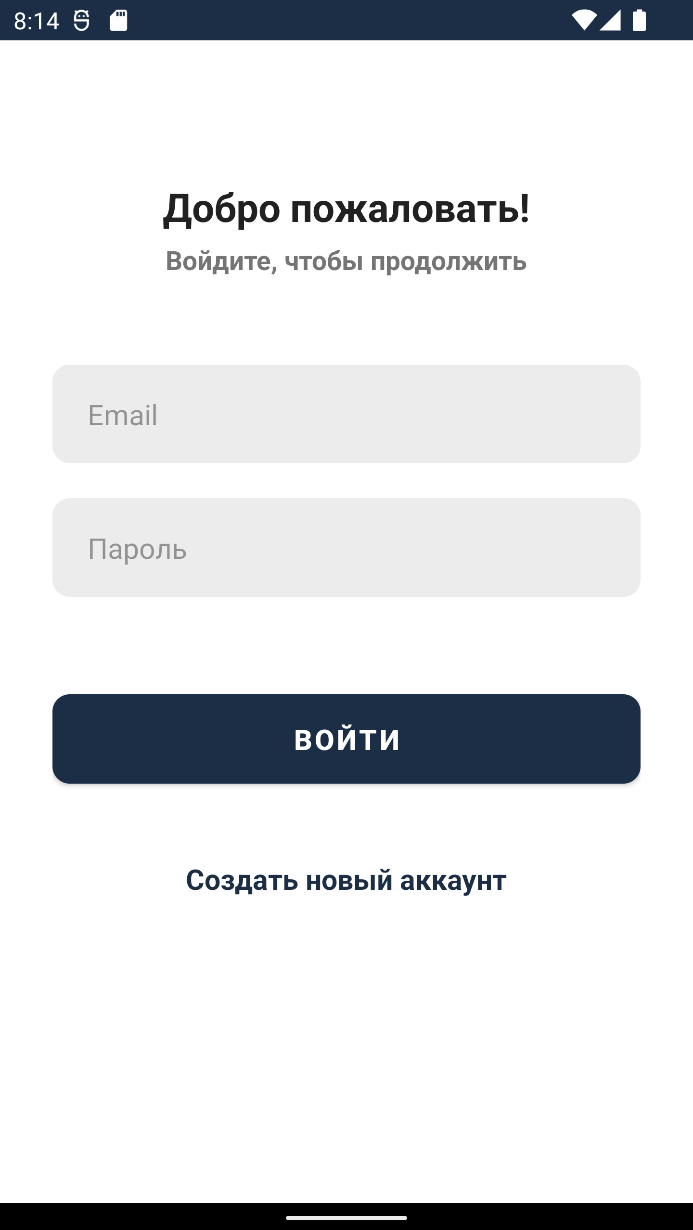


Рисунок 2.2 – Пример окна авторизации

В окне регистрации расположены элементы управления для выбора фотографии профиля пользователя, для ввода логина, адреса электронной почты, пароля и подтверждения пароля, а также элементы управления для регистрации в системе и ссылку перехода в окно авторизации. Пример окна регистрации представлен на рисунке 2.3.



Рисунок 2.3 – Пример окна регистрации

В верхней части главного окна расположены элементы отображения фотографии профиля пользователя, логина и кнопки выхода. В центре окна показан список диалогов, которые содержат в себе информацию в виде фото собеседника, его логин и последнее сообщение. Справа снизу расположена кнопка создания диалога. Пример главного окна представлен на рисунке 2.4.

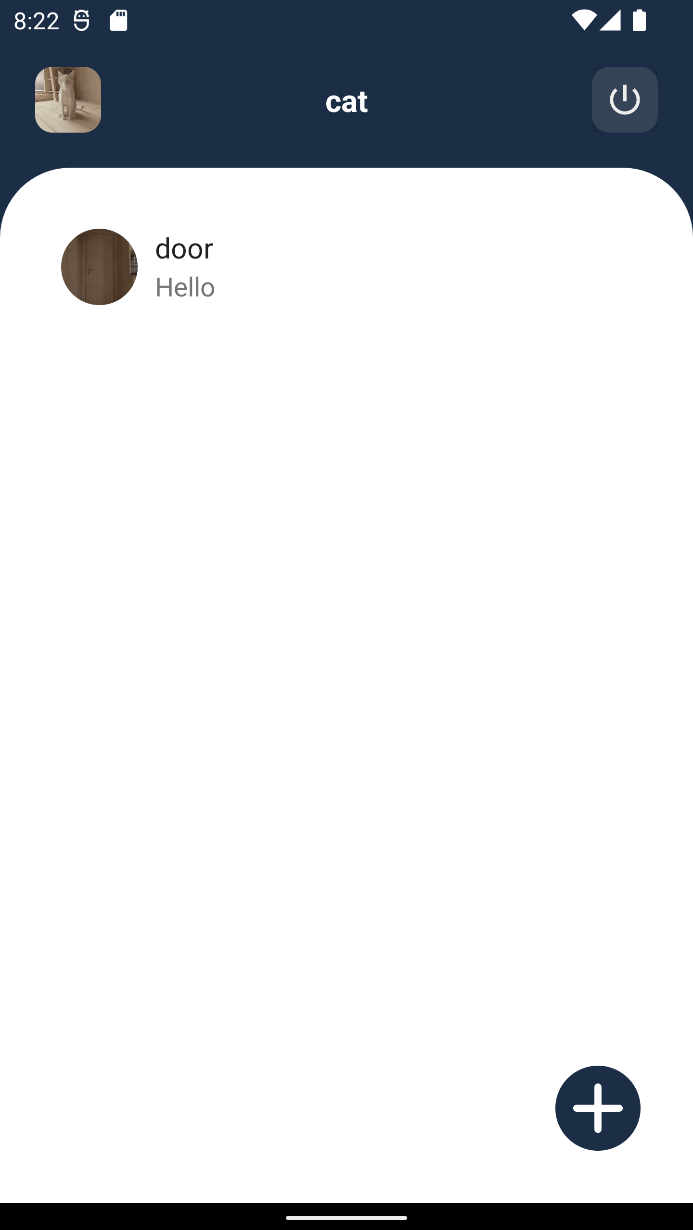


Рисунок 2.4 – Пример главного окна

В верхней части окна создания диалога расположены элементы управления для возврата в главное окно и поле для поиска пользователя по логину. В центре окна показан список пользователей, логин которых соответствует критерию поиска. Информация о пользователе представлена в виде фото профиля и логина аккаунта. Пример окна выбора пользователя для создания диалога представлен на рисунке 2.5.

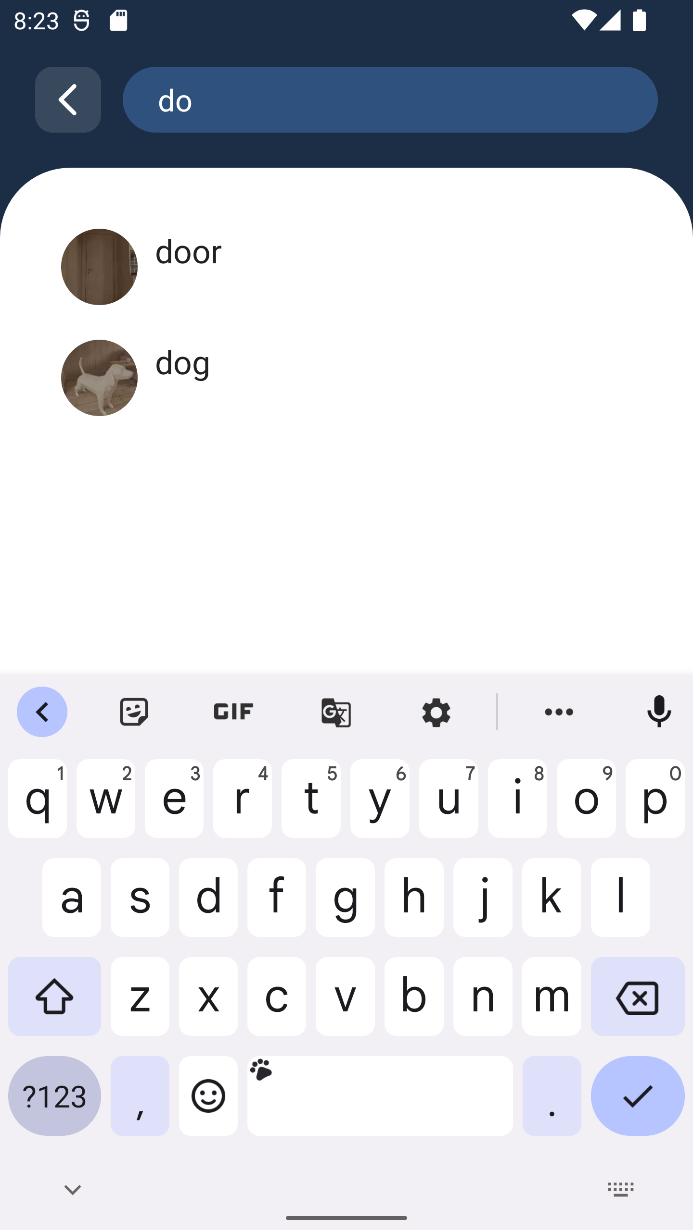


Рисунок 2.5 – Пример окна выбора пользователя для создания диалога

В верхней части окна диалога расположены элементы управления для возврата в главное окно, а также элементы отображения основной информации о собеседнике, содержащей фотографию профиля и логин, а также его онлайн-статус. В центре окна показаны все сообщения, содержащие в себе сам текст сообщения и время отправки. В нижней части окна расположены поле ввода сообщения и кнопка отправки. Пример окна диалога представлен на рисунке 2.6.

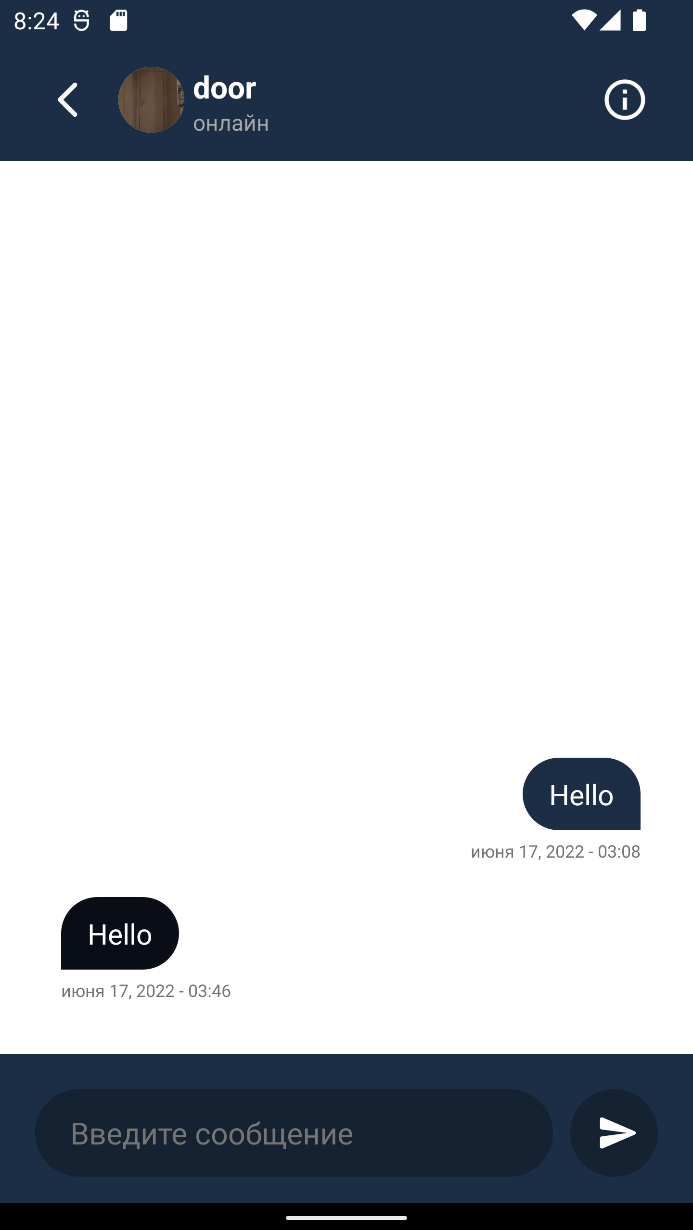


Рисунок 2.6 – Пример окна диалога с пользователем

2.3 Разработка структур данных для обслуживания мобильного приложения

Для обработки действий пользователя на главном окне, существует контроллер – MainActivity. Этот контроллер содержит в себе следующие поля:

* private ActivityMainBinding binding – отвечает за взаимодействие элементов разметки в контроллере;
* private PreferenceManager preferenceManager – содержит в себе данные об авторизированном на устройстве пользователе;
* private List<ChatMessages> conversations – содержит в себе список всех диалогов;
* private RecentConversationsAdapter conversationsAdapter – переменная представляет объект одного недавнего диалога;
* private FirebaseFirestore database – содержит в себе ссылку на базу данных.

Также у класса MainActivity есть ряд методов:

* protected void onCreate – метод инициализирует поля при первой загрузке окна;
* private void init – метод загружает все недавние диалоги авторизированного пользователя в список;
* private void setListeners – метод устанавливает события взаимодействия с элементами управления;
* private void listenConversations – метод отображает все недавние диалоги на экран;
* private void updateToken – метод обновляет токен, отвечающий за авторизованность в приложении;
* private void loadUserDetails – метод загружает на экран данные об авторизованном пользователе;
* private void signOut – метод обрабатывает событие выхода из приложения;
* private void onConversionClicked – метод совершает переход в диалог с выбранным пользователем.

Для обработки действий пользователя на окне регистрации, существует контроллер – SignUpActivity. Этот контроллер содержит в себе следующие поля:

* private ActivitySignUpBinding binding – отвечает за взаимодействие элементов разметки в контроллере;
* private PreferenceManager preferenceManager – содержит в себе данные о регистрируемом пользователе;
* private String encodedImage – хранит в себе выбранное изображение в закодированном виде.

Также у класса SignUpActivity есть ряд методов:

* protected void onCreate – метод инициализирует поля при первой загрузке окна;
* private void setListeners – метод устанавливает события взаимодействия с элементами управления;
* private void signUp – метод регистрирует нового пользователя;
* private String encodeImage – метод кодирует фотографию в строку символов;
* private Boolean isValidSignUpDetails – метод проверяет валидность введенных данных.

Для обработки действий пользователя на окне авторизации, существует контроллер – SignInActivity. Этот контроллер содержит в себе следующие поля:

* private ActivitySignInBinding binding – отвечает за взаимодействие элементов разметки в контроллере;
* private PreferenceManager preferenceManager – содержит в себе данные о регистрируемом пользователе.

Также у класса SignUpActivity есть ряд методов:

* protected void onCreate – метод инициализирует поля при первой загрузке окна;
* private void setListeners – метод устанавливает события взаимодействия с элементами управления;
* private void signIn – метод авторизуется в приложении;
* private Boolean isValidSignInDetails – метод проверяет валидность введенных данных.

Для обработки действий пользователя на окне выбора пользователя для создания диалога, существует контроллер – UsersActivity. Этот контроллер содержит в себе следующие поля:

* private ActivityUsersBinding binding – отвечает за взаимодействие элементов разметки в контроллере;
* private PreferenceManager preferenceManager – содержит в себе данные о регистрируемом пользователе.

Также у класса UsersActivity есть ряд методов:

* protected void onCreate – метод инициализирует поля при первой загрузке окна;
* private void setListeners – метод устанавливает события взаимодействия с элементами управления;
* private void getUsers – метод загружает на экран всех пользователей, зарегистрированных в системе, логин которых соответствует критерию поиска;
* private void onUserClick – метод создает диалог с выбранным пользователем.

Для обработки действий пользователя на окне диалога, существует контроллер – ChatActivity. Этот контроллер содержит в себе следующие поля:

* private ActivityChatBinding binding – отвечает за взаимодействие элементов разметки в контроллере;
* private User receiverUser – содержит в себе информацию о собеседнике;
* private List<ChatMessage> chatMessages – содержит в себе все сообщения в данном диалоге;
* private ChatAdapter chatAdapter – переменная представляет объект одного диалога;
* private PreferenceManager preferenceManager – содержит в себе данные об авторизированном на устройстве пользователе;
* private FirebaseFirestore database – содержит в себе ссылку на базу данных.

Также у класса ChatActivity есть ряд методов:

* protected void onCreate – метод инициализирует поля при первой загрузке окна;
* private void init – метод загружает все сообщения в список;
* private void sendMessage – метод сохраняет сообщение в базу и отправляет его собеседнику;
* private void listenAvailabilityOfReceiver – метод отслеживает онлайн-статус собеседника;
* private void listenMessages – метод отображает все сообщения на экран;
* private void loadReceiverDetails – метод загружает на экран информацию о собеседнике;
* private void setListeners – метод устанавливает события взаимодействия с элементами управления;
* private void addConversion – метод создает таблицу в базе, хранящей в себе информацию о диалоге, в случае если диалога авторизованного пользователя с выбранным не существует;
* private void updateConversion – метод обновляет последнее сообщение в списке диалогов;
* private void checkForConversion – метод обновляет список недавних диалогов пользователя.

Для хранения данных о пользователе в приложении существует класс-модель User. Эта модель содержит в себе следующие поля:

* public String name – хранит в себе логин пользователя;
* public String image – хранит в себе закодированное фото профиля пользователя;
* public String email – хранит в себе адрес электронной почты пользователя;
* public String token – хранит в себе токен пользователя, отвечающий за авторизованность в приложении;
* public String id – хранит в себе идентификатор пользователя.

Для хранения данных о сообщении существует класс-модель ChatMessage. Эта модель содержит в себе следующие поля:

* public String senderID – хранит в себе идентификатор отправителя сообщения;
* public String receiverID – хранит в себе идентификатор получателя сообщения;
* public String message – хранит в себе текст сообщения;
* public String dateTime – хранит в себе время отправки сообщения;
* public String conversionID – хранит в себе идентификатор диалога;
* public String conversionName – хранит в себе имя собеседника;
* public String conversionImage – хранит в себе фото профиля собеседника.

2.4 Разработка алгоритмов и программного кода мобильного приложения

В подразделе 2.3 был выявлен ряд классов и методов, необходимых для реализации бизнес-логики приложения. Данные классы и методы будут проработаны в этом подразделе.

Функция signUp служит для регистрации пользователя в системе. Входных параметров не имеет. Блок-схема алгоритма регистрации представлена на рисунке 2.7.

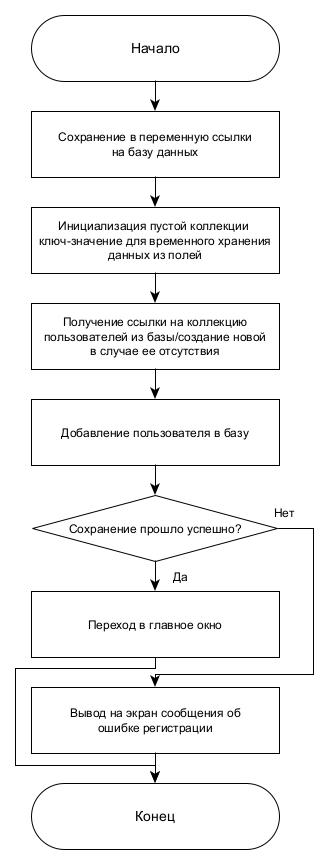


Рисунок 2.7 – Блок-схема функции signUp

Функция signIn служит для авторизации пользователя в системе. Входных параметров не имеет. Блок-схема алгоритма авторизации представлена на рисунке 2.8.

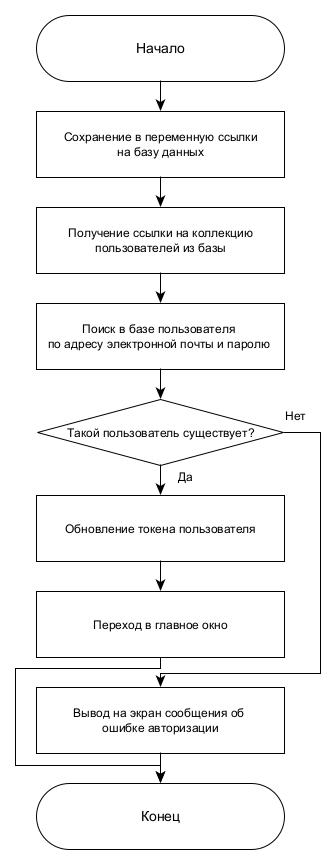


Рисунок 2.8 – Блок-схема алгоритма signIn

Функция signOut служит для выхода пользователя из аккаунта. Входных параметров не имеет. Блок-схема алгоритма выхода из аккаунта представлена на рисунке 2.9.

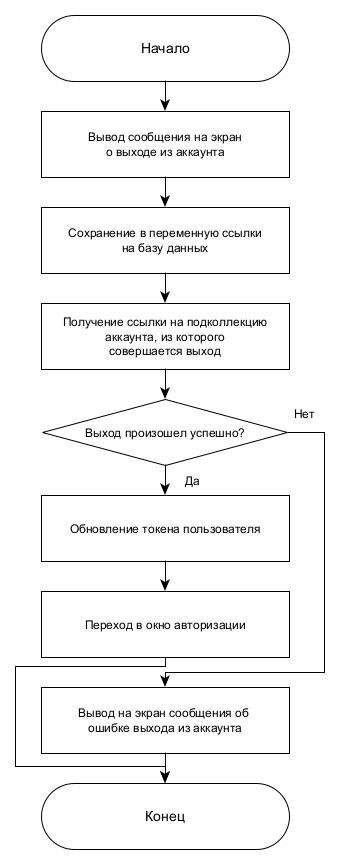


Рисунок 2.9 – Блок-схема алгоритма singOut

Функция getUsers служит для поиска пользователей в системе, логин которых соответствует критерию поиска. Входных параметров не имеет. Блок-схема алгоритма поиска представлена на рисунке 2.10.

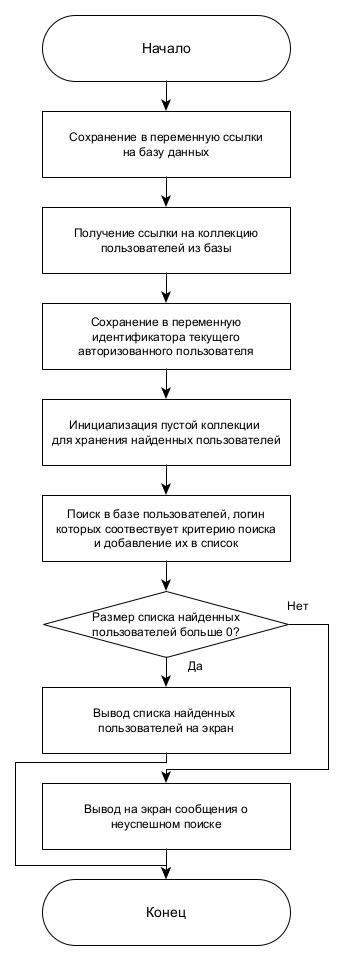


Рисунок 2.10 – Блок-схема алгоритма getUsers

Функция onUserClick служит для создания диалога с выбранным пользователем. В качестве входного параметра выступает класс User, поля которого содержат в себе информацию о выбранном пользователе. Блок-схема алгоритма выбора пользователя представлена на рисунке 2.11.

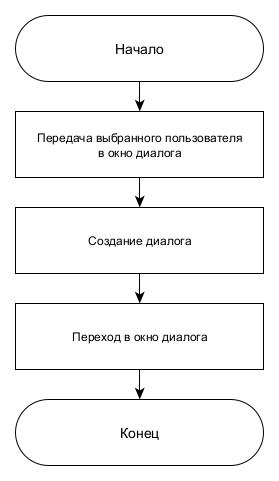


Рисунок 2.11 – Блок-схема алгоритма onUserClick

Функция sendMessage служит для отправки сообщения собеседнику. Входных параметров не имеет. Блок-схема алгоритма отправки сообщения представлена на рисунке 2.12.

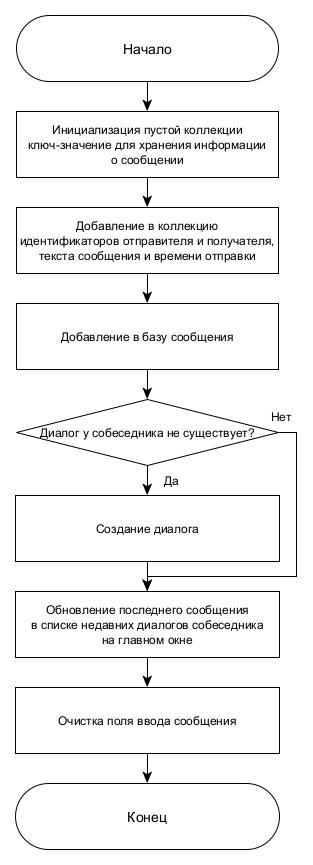


Рисунок 2.12 – Блок-схема алгоритма sendMessage

Метод listenAvailabilityOfReceiver служит для отслеживания онлайн-статуса собеседника. Входных параметров не имеет. Блок-схема алгоритма отслеживания представлена на рисунке 2.13.

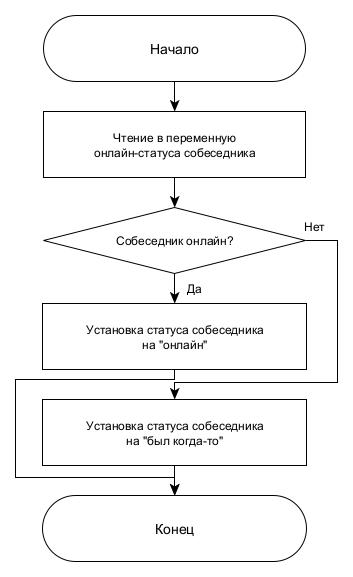


Рисунок 2.13 – Блок-схема алгоритма listenAvailabilityOfReceiver

Функция updateConversion служит для обновления последнего сообщения в списке диалогов на главном окне. Входных параметров не имеет. Блок-схема алгоритма обновления последнего сообщения представлена на рисунке 2.15.

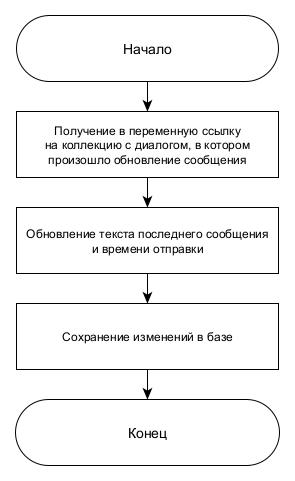


Рисунок 2.15 – Блок-схема алгоритма updateConversion

Функция updateToken служит для обновления токена пользователя. Входных параметров не имеет. Блок-схема алгоритма обновления токена представлена на рисунке 2.16.



Рисунок 2.16 – Блок-схема алгоритма updateToken

2.5 Разработка интерфейса для подключения к серверной части приложения

В качестве хранения информации о пользователях, диалогах и сообщениях была выбрана нереляционная облачная система управления базой данных Firebase. Все данные в базе хранятся в формате json, а также данная СУБД предоставляет удобные методы для чтения, обновления и извлечения данных. Также все данные хранятся в ней в режиме реального времени, что означает, что каждое изменение будет автоматически обновлять подключенных клиентов.

Для работы с базой данных в приложении для начала необходимо получить ссылку на эту базу данных. Пример кода получения ссылки приведен на рисунке 2.17.



Рисунок 2.17 – Получение ссылки базы данных

Далее, чтобы записать в базу необходимую информацию, необходимо обратиться к нужной коллекции по ключу и совершить необходимое действие. Пример обращения к коллекции и записи в базу данных приведен на рисунке 2.18.



Рисунок 2.18 – Получение ссылки на коллекцию и запись в базу

Чтобы считать данные из базы, необходимо обратиться к нужной коллекции по ключу и считать в переменную запись из базы данных по условию. Пример чтения записи из базы в переменную приведен на рисунке 2.19.

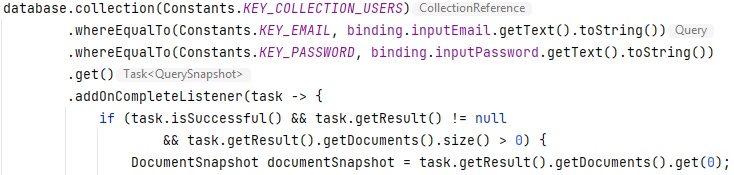


Рисунок 2.19 – Чтение записи из базы в переменную

Чтобы удалить запись из базы данных, необходимо обратиться к нужной коллекции по ключу, задать условие поиска записи, выполнить необходимые преобразования из коллекции базы данных в объект результата запроса и удалить запись с последующим обновлением списка. Пример удаления записи из базы данных приведен на рисунке 2.20.

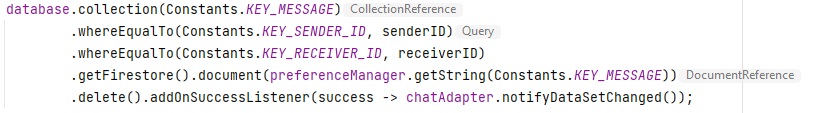


Рисунок 2.20 – Удаление записи из базы

3 Технологический раздел

3.1 Отладка подпрограмм мобильного приложения

Отладка программы – это специальный этап в разработке программы, состоящий в выявлении и устранении программных ошибок, факт существования которых уже установлен.

Программные ошибки, как правило, делятся на три вида.

Первым видом является синтаксическая ошибка. Синтаксические ошибки относят к группе самых простых, так как синтаксис языка, как правило, строго формализован, и ошибки сопровождаются развернутым комментарием с указанием ее местоположения.

Второй вид – семантическая ошибка. Нарушение семантики той или иной конструкции, например передача функции параметров, не соответствующих ее аргументам.

Третий вид – логическая ошибка. Нарушение логики программы, приводящее к неверному результату. Это наиболее трудный для "отлова" тип ошибки, ибо подобного рода ошибки, как правило, кроются в алгоритмах и требуют тщательного анализа и всестороннего тестирования.

В больших и сложных программах логические ошибки и ошибки периода выполнения достаточно трудно отследить и обнаружить. В этих случаях вполне естественным является желание выполнить программу в интерактивном режиме, наблюдая за изменениями значений отдельных переменных или выражений. При этом желательно иметь возможность останавливаться в определенном месте программы и смотреть, что там происходит. Часто бывает полезно изменять значения некоторых переменных при выполнении программы. Это позволит повлиять на ее поведение и увидеть, в какую сторону оно изменилось. В каждой современной системе программирования существует специальное средство отладки программ, которое предоставляет все вышеперечисленные возможности — отладчик.

Существуют две взаимодополняющие технологии отладки. Первая – использование отладчиков. Вторая технология заключается в выводе текущего состояния программы с помощью расположенных в критических точках программы операторов вывода — на экран, принтер, громкоговоритель или в файл.

В данном разделе будет проведён процесс отладки всех функций, которые присутствуют в программном продукте, для удобства восприятия, каждый процесс отладки функции будет описан в отдельной таблице.

В таблице 3.1 приведён процесс отладки функции регистрации.

Таблица 3.1 – Отладка функции регистрации

|  |  |
| --- | --- |
| Предусловие | Запущено приложение |
| Набор | 1 |
| Шаг | Полученный результат |
| Нажать на кнопку «Создать новый аккаунт» | Открылась окно регистрации |
| Заполнить все поля данными | В полях отображаются введенные данные |
| Нажать на кнопку “Зарегистрироваться” | Регистрация аккаунта в системе и его отображение в списке пользователей при создании диалога |
| Набор | 2 |
| Шаг | Полученный результат |
| Нажать на кнопку «Создать новый аккаунт» | Открылось окно регистрации |
| Оставить некоторые поля пустыми | В полях ничего не отображается |
| Нажать на кнопку “Зарегистрироваться” | Предупреждение от том, что не все поля заполнены |
| Набор | 3 |
| Шаг | Полученный результат |
| Нажать на кнопку «Создать новый аккаунт» | Открылось окно регистрации |
| Вести разные пароль и подтверждение пароля | В полях отображаются введенные данные |
| Нажать на кнопку “Зарегистрироваться” | Предупреждение от том, что поля пароль и подтверждение пароля должны совпадать |

В таблице 3.2 приведён процесс отладки функции авторизации.

Таблица 3.2 – Отладка функции авторизации

|  |  |
| --- | --- |
| Предусловие | Запущено приложение |
| Набор | 1 |
| Шаг | Полученный результат |
| Нажать на кнопку «Войти» | Открылось окно авторизации |
| Заполнить все поля данными | В полях отображаются введенные данные |
| Нажать на кнопку «Войти» | Вход в систему |
| Набор | 2 |
| Шаг | Полученный результат |
| Нажать на кнопку «Войти» | Открылось окно авторизации |
| Ввести почту в неправильном формате | В полях отображаются введенные данные |
| Нажать на кнопку «Войти» | Предупреждение о том, что почта введена в неправильном формате |

В таблице 3.3 приведён процесс отладки функции выхода из аккаунта.

Таблица 3.3 – Отладка функции выхода из аккаунта

|  |  |
| --- | --- |
| Предусловие | Запущено приложение |
| Набор | 1 |
| Шаг | Полученный результат |
| Нажать на кнопку «Выйти» | Произведен выход пользователя из аккаунта |

В таблице 3.4 приведён процесс отладки функции поиска пользователя в системе по логину.

Таблица 3.4 – Отладка функции поиска пользователя

|  |  |
| --- | --- |
| Предусловие | Открыто окно выбора пользователя для создания диалога |
| Набор | 1 |
| Шаг | Полученный результат |
| Ввести в поле логин | В поле отображается введенное значение |
| Нажать на кнопку «Поиск» | Отображение списка пользователей с соответствующим логином |
| Набор | 2 |
| Шаг | Полученный результат |
| Оставить поле не заполненным | В поле ничего не отображается |
| Нажать на кнопку «Поиск» | Отображение списка всех пользователей |

В таблице 3.5 приведён процесс отладки функции создания диалога.

Таблица 3.5 – Отладка функции создания диалога

|  |  |
| --- | --- |
| Предусловие | Открыто окно выбора пользователя для создания диалога |
| Набор | 1 |
| Шаг | Полученный результат |
| Выбрать необходимого пользователя | Создание диалога с выбранным пользователем |

В таблице 3.6 приведён процесс отладки функции отправки сообщения.

Таблица 3.6 – Отладка функции отправки сообщения

|  |  |
| --- | --- |
| Предусловие | Открыто окно диалога |
| Набор | 1 |
| Шаг | Полученный результат |
| Ввести в поле сообщение | В поле отображается введенное значение |
| Нажать на кнопку «Отправить» | Сохранение сообщения в базе данных и отображение его в диалоге |

3.2 Разработка инструкции по эксплуатации мобильного приложения

При запуске приложения откроется окно авторизации. Пример окна авторизации представлен на рисунке 3.1.

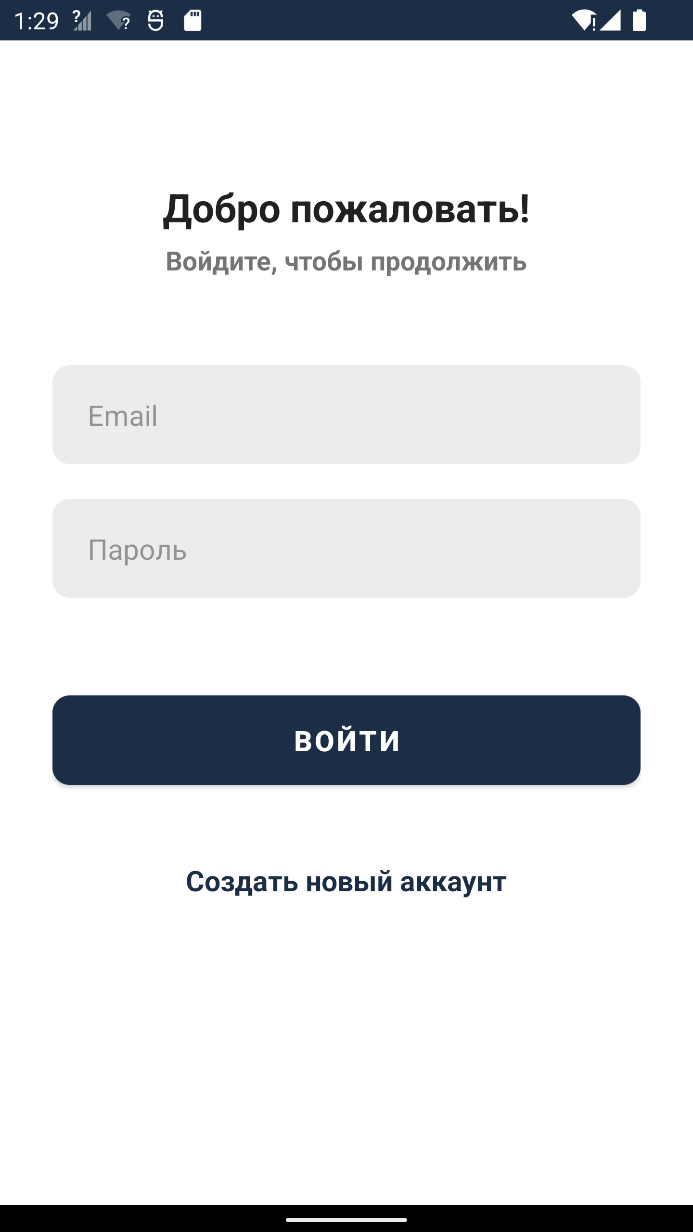


Рисунок 3.1 – Окно авторизации пользователя

Чтобы войти в приложение, пользователь должен ввести корректные почту и пароль. У вводимого адреса электронной почты есть ограничения, заключающиеся в валидировании введенного значения. При попытке ввести почту неправильного формата, пользователь получит сообщение об ошибке, как это показано на рисунке 3.2.

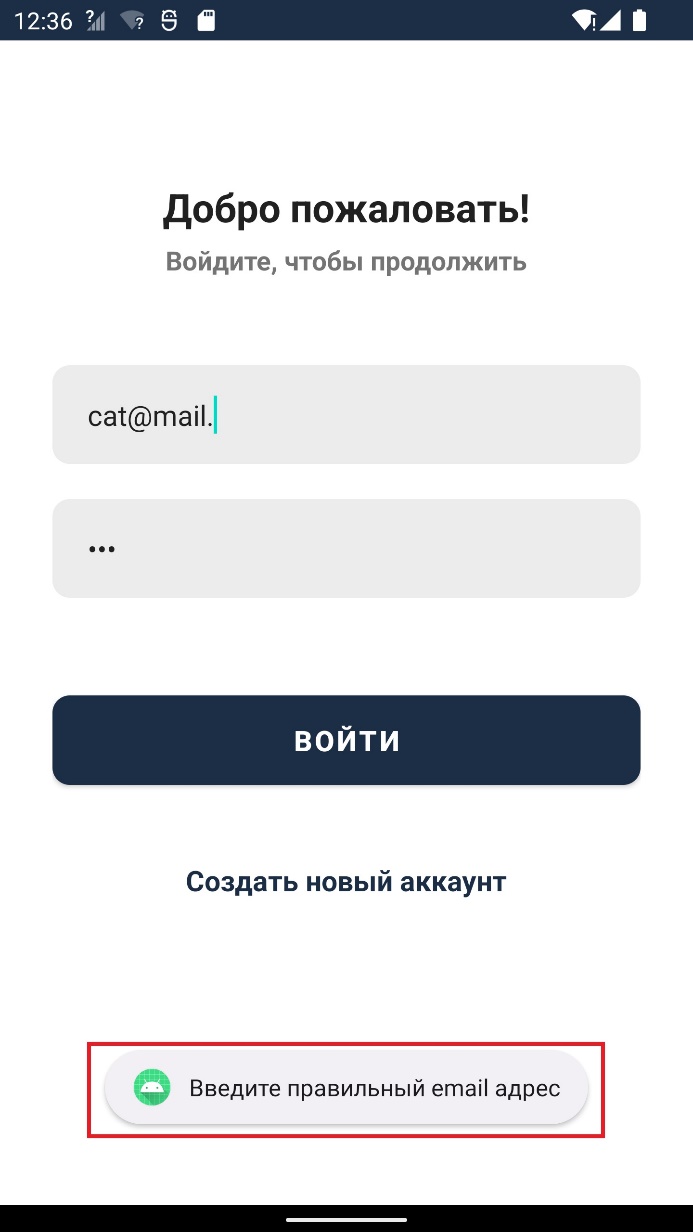


Рисунок 3.2 – Пример работы валидации почты

При вводе правильных данных и нажатии на кнопку «Войти» пользователь перейдет в главное окно приложения. Пример окна авторизации представлен на рисунке 3.3.

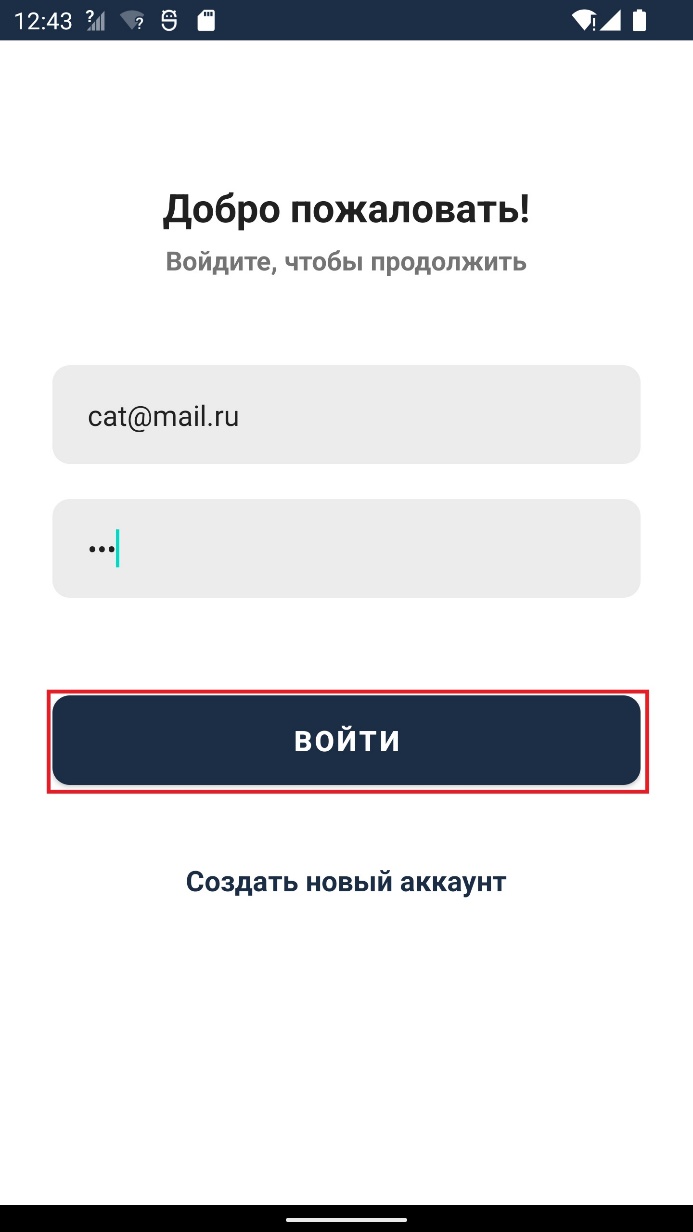


Рисунок 3.3 – Авторизация пользователя

Если аккаунта с такими данными в системе нет, то пользователю необходимо перейти на окно регистрации путем нажатия на ссылку «Создать новый аккаунт». После выполнения вышесказанных действий откроется окно регистрации, как показано на рисунке 3.4.



Рисунок 3.4 – Окно регистрации пользователя

В окне регистрации пользователь должен выбрать фото профиля и ввести корректные данные. У полей почта, пароль и подтверждение пароля есть ограничение, в виде валидатора на ввод. Пользователь обязательно должен ввести одинаковые пароль и подтверждение пароля, иначе появится ошибка регистрации, как показано на рисунке 3.5.

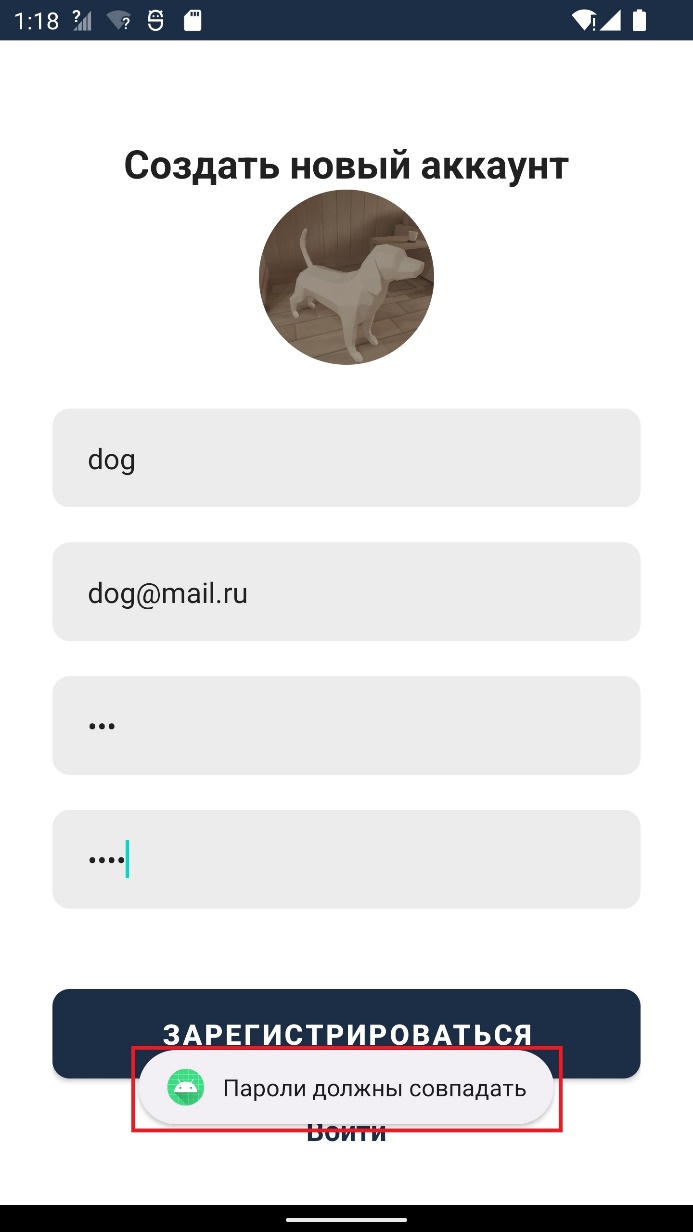


Рисунок 3.5 – Ошибка регистрации

После того, как пользователь введет правильные данные и нажмет кнопку «Зарегистрироваться», он будет зарегистрирован в системе, как показано на рисунке 3.6.

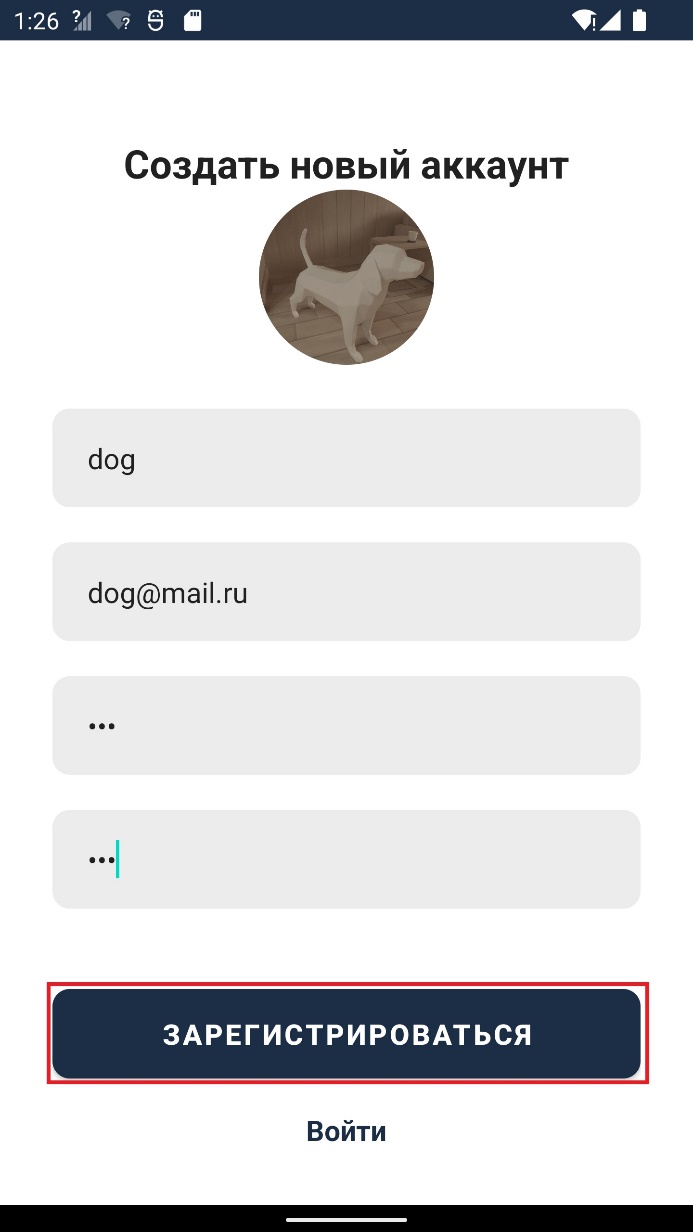


Рисунок 3.6 – Регистрация пользователя

После входа/регистрации в приложение откроется главное окно приложения. Пример главного окна приложения представлен на рисунке 3.7.

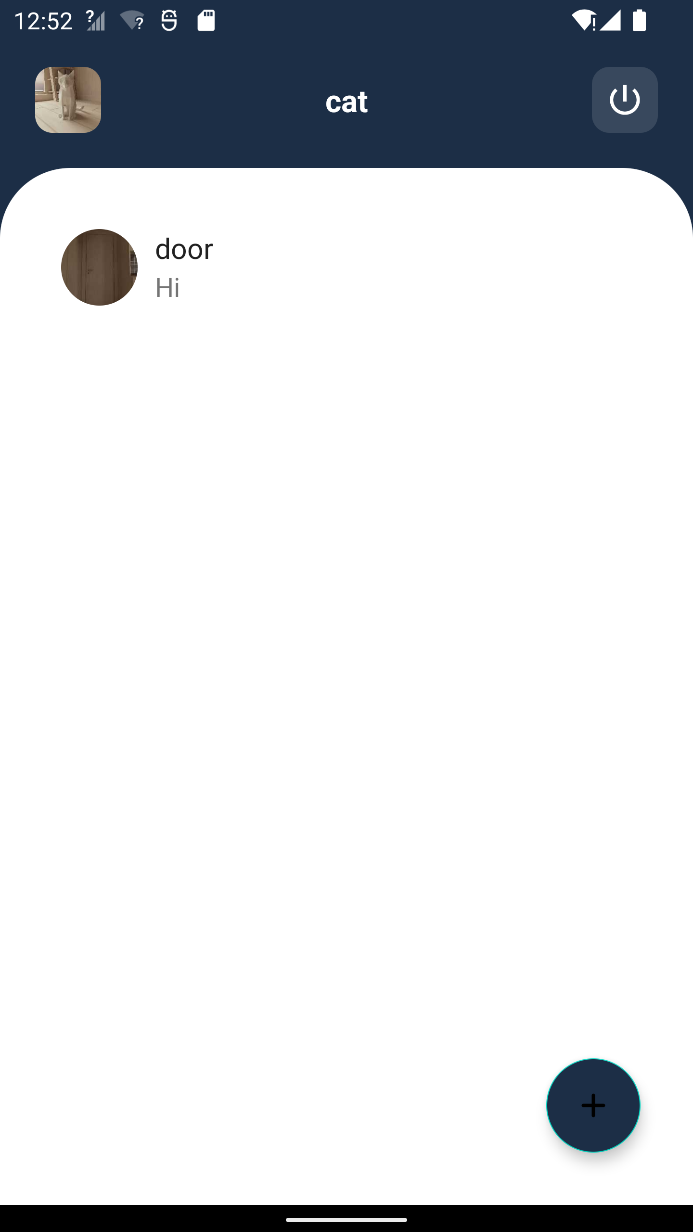


Рисунок 3.7 – Главное окно приложения

Для создания диалога пользователю необходимо нажать на кнопку справа снизу. После проделанных действий будет произведен переход в окно создания диалога со списком всех пользователей, зарегистрированных в системе, как показано на рисунке 3.8.

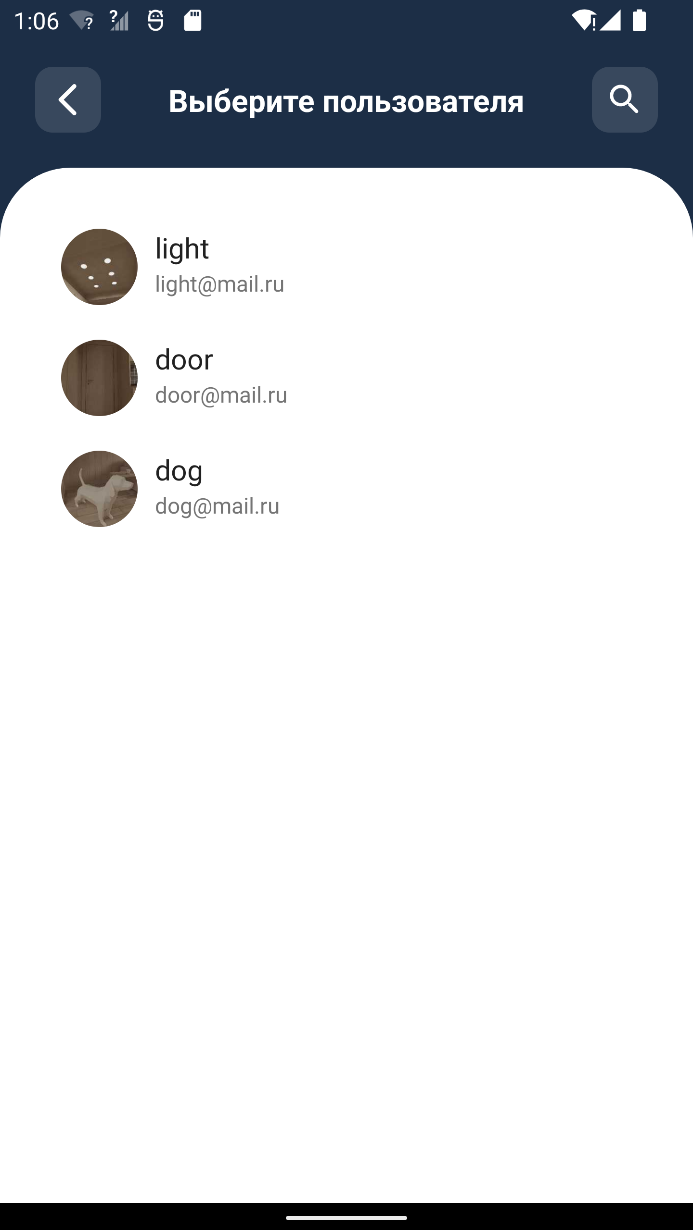


Рисунок 3.8 – Пример окна выбора пользователя для создания диалога

Для поиска по логину, пользователь должен нажать на кнопку поиска в верхней панели в правой части. Список пользователей обновляется по мере ввода логина в поле. Пример поиска пользователя представлен на рисунке 3.9.

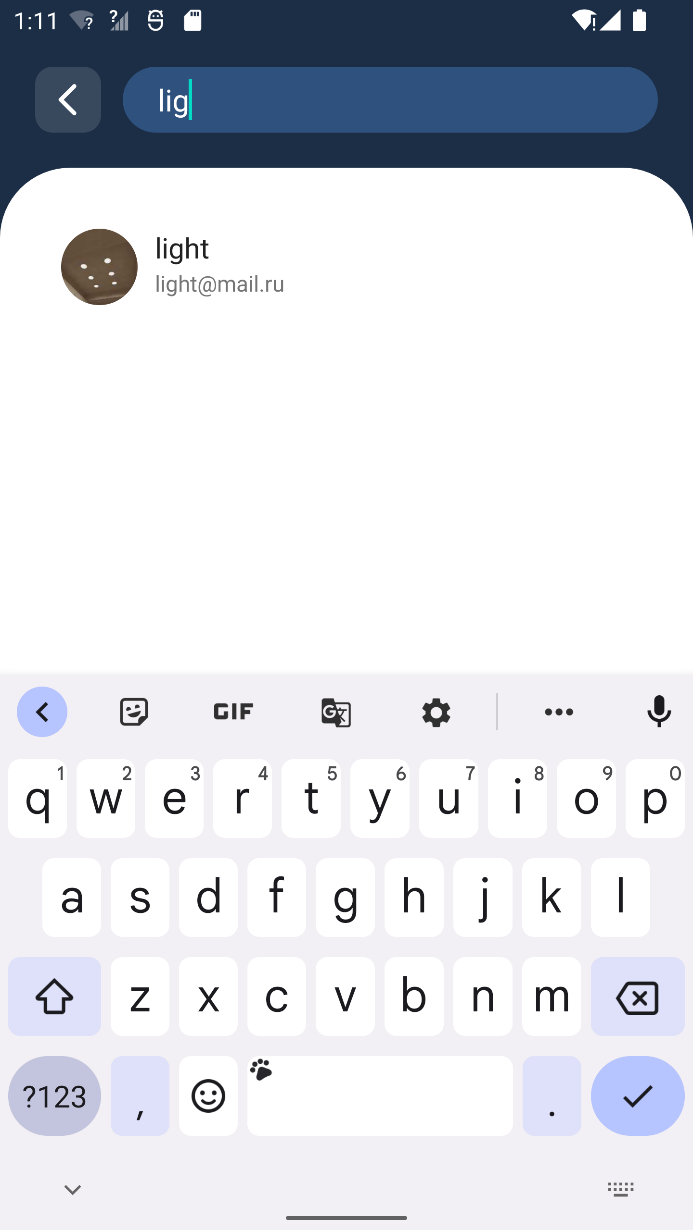


Рисунок 3.9 – Пример поиска пользователя по логину

После выбора пользователя из списка будет создан диалог и осуществится переход на окно диалога. В верхней панели находятся: кнопка возвращения на главное окно приложения, а также фото профиля и логин пользователя. В нижней части экрана находится поле ввода сообщения и кнопка отправки. Чтобы отправить сообщение, пользователю необходимо ввести текст сообщения в поле и нажать на кнопку отправки. Пример окна диалога представлен на рисунке 3.10.

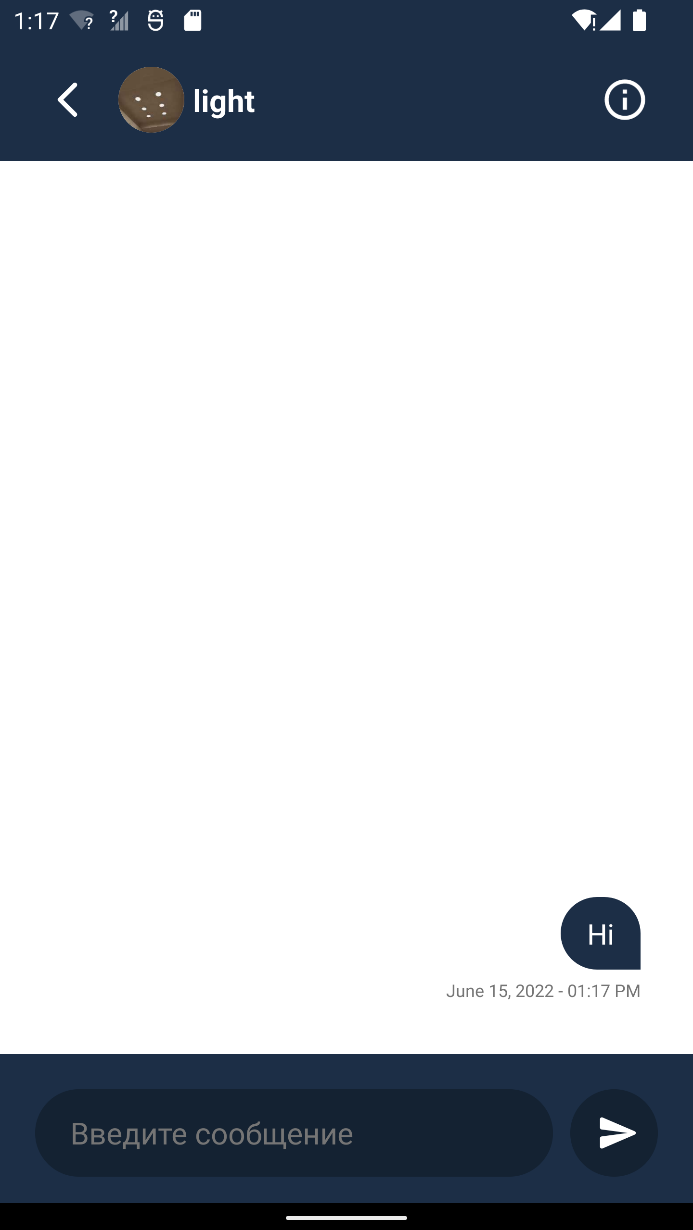


Рисунок 3.10 – Окно диалога с пользователем

4 Технико-экономический раздел

Основой расчета затрат на любой производственный процесс обычно является смета затрат. Смета затрат представляет собой сводный план всех расходов предприятия или организации на рассматриваемый период деятельности. Она определяет общую сумму издержек производства по видам используемых ресурсов, стадиям производственной деятельности, уровням управления предприятием и другим направлениям расходов. В смету включаются затраты основного и вспомогательного производства, связанные с изготовлением и продажей рассматриваемого продукта, а также на содержание административно-управленческого персонала, выполнение различных работ и услуг, в том числе и не входящих в основную производственную деятельность предприятия или организации.

Первым компонентом, входящим в сметный расчет, является материалы. В состав этих затрат принято включать стоимость материалов, которые будут проданы заказчику вместе с программным продуктом.

Совокупные затраты на материалы М*,* руб. рассчитываются по формуле 4.1:

, (4.1)

где *Мi* – стоимость, затраченная на каждый материал, руб.

Все материалы, используемые для изготовления проекта, их количество и стоимость включены, сведены в таблицу 4.1.

Таблица 4.1 –Материалы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование | Цена за единицу, руб., коп. | Количество,  ед. изм. | Стоимость,  руб., коп. |
| 1 | USB-накопитель 1 Гб | 100,00 | 1 | 100,00 |
|  | Итого |  |  | 100,00 |

Стоимость отдельного материала М*i,* руб. рассчитывается по формуле 4.2:

, (4.2)

где Ni – стоимость отдельного материала за штуку, руб.;

Ki – количество отдельного материала, руб.

Подстановкой указанных выше значений в формулу 4.1 получено:

Вторым компонентом, включаемым в сметный расчет, являются затраты на вспомогательные материалы. В состав этих затрат принято включать стоимость расходуемых за период работ покупных инструментов и малоценного хозяйственного инвентаря.

Совокупные затраты на вспомогательные материалы и малоценный инвентарь ВМ*,* руб. рассчитываются по формуле 4.3:

, (4.3)

где *ВМi* – стоимость, затраченная на каждый вспомогательный материал, руб.

Все вспомогательные материалы, используемые в рамках работы непосредственно над проектом, их количество и стоимость включены в таблицу 4.2.

Таблица 4.2 – Вспомогательные материалы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование | Цена за единицу, руб., коп. | Количество,  ед. изм. | Стоимость,  руб., коп. |
| 1 | Бумажный лист формата А4. | 0,8 | 122 | 97,60 |
| 2 | Ватман формата А0, шт. | 160,00 | 2 | 320,00 |
| 3 | Картридж для принтера, шт. | 1490,00 | 0,4 | 596,00 |
|  | Итого |  |  | 1013,60 |

Стоимость отдельного вспомогательного материала *ВМi,* руб. рассчитывается по формуле 4.4:

, (4.4)

где Ni – стоимость отдельного вспомогательного материала за штуку, руб.;

Ki – количество отдельного вспомогательного материала, руб.

Подстановкой указанных выше значений в формулу 4.3 получено:

Третьим компонентом затрат на разработку являются затраты на энергетические ресурсы ЭР, руб, состоящие из затрат на электроэнергию и затрат на топливо, и рассчитываемые по формуле 4.5:

, (4.5)

где Э – затраты на электроэнергию, руб.;

Т – затраты на топливо, руб.

Так как разработка программного продукта не требует затрат на топливо, то стоимость топлива принимается равным нулю.

Затраты на электроэнергию вычисляются по формуле 4.6:

, (4.6)

где W – совокупная потребленная мощность, КВт×ч;

ЦЭ – стоимость одного КВт×ч электроэнергии, руб.; составляет 3,99руб.

Совокупная потребленная мощность *W* рассчитывается по формуле 4.8:

, (4.7)

где Wi – мощность, потребляемая отдельным устройством, КВт.

В таблице 4.3 представлены все возможные устройства, используемые в разработке продукта, потребляющие электроэнергию.

Таблица 4.3 – Используемые устройства

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Наименование | Потребляемая мощность, КВт | Продолжительность эксплуатации в смену, час. | Количество смен в цикле производства (разработки), шт. | Потребленная мощность, КВт×ч |
| 1 | Компьютер | 0,5 | 8 | 6 | 24 |
| 2 | Принтер | 0,34 | 0,5 | 6 | 1,02 |
|  | Итого |  |  |  | 25,02 |

Потребленная мощность *Wi*, кВт×ч для каждого из устройств рассчитывается по формуле 4.8:

, (4.8)

где Pi – мощность устройства, КВт;

ti – продолжительность эксплуатации устройства за одну рабочую смену. час.;

КСi – количество смен, в течение которых использовалось устройство, шт.

Подстановкой значений из таблицы 4.3 в формулу 4.7 получено значение потребленной мощности:

Подстановкой указанных выше значений в формулу 4.6, получаем затраты на электроэнергию:

Подстановкой указанных выше значений в формулу 4.5, получаем затраты на энергетические ресурсы:

Четвертым этапом сметного расчета является расчет амортизации использованного в проекте оборудования.

Амортизация – это процесс постепенного переноса стоимости средств, по мере их износа, на стоимость продукции, которая производится. Амортизация распространяется на основные средства, основные фонды и нематериальные активы. Она исчисляется линейным, нелинейным методом, способом уменьшения остаточной стоимости, способом списания стоимости по сумме лет использования [9].

Амортизационные отчисления – отчисления части стоимости основных средств для возмещения их износа. Амортизационные отчисления включаются в издержки производства и производятся коммерческими организациями на основе установленных норм и балансовой стоимости основных средств.

Общий объем амортизационных отчислений *АМ*, руб., включаемый в расчет по проекту для всех устройств можно вычислить по формуле 4.9:

, (4.9)

где *Аi* – амортизационное отчисление для отдельного основного средства, руб.

Для разработки программного продукта потребовалось оборудование и нематериальные активы, перечисленные в таблице 4.4.

Таблица 4.4 – Информация по расчету амортизационных отчислений

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Наименование основного средства | Изначальная стоимость, руб., коп. | Срок службы, лет | Сумма ежегодного амортиза-ционного отчисления,  руб., коп. | Период использования в проекте, кален. дн. | Сумма амортиза-ционного отчисления, руб., коп. |
| 1 | Офисный стол | 5 399,00 | 4 | 1 349,75 | 9 | 33,28 |
| 2 | Офисное кресло | 3 169,00 | 5 | 633,80 | 9 | 15,63 |
| 3 | Компьютер | 134 499,00 | 5 | 26 899,80 | 9 | 663,28 |
| 4 | Принтер | 6 200,00 | 5 | 1 240,00 | 9 | 30,57 |
| 5 | Windows 11 Pro | 4 900,00 | 2 | 2 450,00 | 9 | 60,41 |
| 7 | Microsoft Office professional 2021 | 59 590,00 | 6 | 9 931,67 | 9 | 244,89 |
|  | Итого |  |  |  |  | 1 048,06 |

Амортизационное отчисление *Аi,* руб., включаемое в расходы по проекту, для каждого из основных средств можно вычислить по формуле 4.10:

, (4.10)

где *АОi*– сумма планируемого амортизационного отчисления для основного средства за год, руб.;

*КДi*– количество календарных дней, в течение которых использовалось основное средство, дн.;

*365* – это количество календарных дней в году, дн.

Для всех основных средств и нематериальных активов рассчитывается сумма годового амортизационного отчисления *АОi*, руб. по формуле 4.11, которое каждый год будет составлять одну и ту же величину для одного и того же устройства.

, (4.11)

где *НСi* – начальная стоимость основного средства, руб.;

*Срi* – срок службы основного средства, лет.

Далее для каждого основного средства рассчитывается период его использования в проекте в календарных днях *КДi*, формула 4.12:

, (4.12)

где *КСi* – это количество рабочих дней (фактически, при односменном режиме работы – количество рабочих смен);

*5* – это количество рабочих дней на наделе;

*7* – это количество календарных дней в неделе.

Подстановкой значений из таблицы 4.4 в формулу 4.9 получено:

Следующим этапом сметного расчета является затраты на оплату труда.

Фонд оплаты труда ЗП, руб., включаемый в затраты по проекту, можно рассчитать по формуле 4.13:

, (4.13)

где ОЗП – основная заработная плата всех работников, участвовавших в проекте, руб.;

ДЗП – дополнительная заработная плата всех работников, участвовавших в проекте, руб.;

Основная заработная плата ОЗП, руб., включаемая в затраты проекта, будет складываться из двух частей, для расчёта можно воспользоваться формулой 4.14:

, (4.14)

где ОкЗП – сумма заработных плат всех рабочих, участвовавших в проекте, оплачиваемых по окладной системе оплаты труда, руб.;

СдЗП – сумма заработных плат всех рабочих, участвовавших в проекте, оплачиваемых по сдельной системе оплаты труда;

Заработная плата всех рабочих ОкЗП, руб., участвовавших в проекте, оплачиваемых по окладной системе оплаты труда, вычисляется по формуле 4.15:

, (4.15)

где ОкЗПi – заработная плата отдельного рабочего, участвовавшего в проекте, оплачиваемого по окладной системе оплаты труда, руб.

В таблице 4.5 сведены данные по работникам, принимавшим участие в проекте, оплачиваемым по окладной системе оплаты труда.

Таблица 4.5 – Заработная плата

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Должность, профессия  работника | Месячный оклад, руб., коп. | Отработано смен, шт. | Заработная плата, руб., коп. |
| 1 | Инженер-программист | 86 000,00 | 6 | 25 294,12 |
|  | Итого |  |  | 25 294,12 |

В рамках проекта заработная плата ОкЗПi, руб., заработанная каждым из работников, оплачиваемых по окладной системе оплаты труда, рассчитывается по формуле 4.16:

, (4.16)

где Оi – оклад работника, руб.,

КСi – количество смен (рабочих дней), которое он работал по проекту, шт., составляет 6 дней;

20,4 – это среднемесячное количество смен (рабочих дней), шт.

Подстановкой указанных выше значений в формулу 4.15 получено:

Заработная плата всех рабочих, оплачиваемых по сдельной системе оплаты труда, , руб., рассчитывается по формуле 4.17:

, (4.17)

где СдЗПi – сумма заработных плат всех рабочих, участвовавших в проекте, оплачиваемых по сдельной системе оплаты труда, руб.

Так как в разработке проекта не участвовали рабочие, оплачиваемые по сдельной системе оплаты труда, сдельная заработная плата равна нулю.

Подстановкой указанных выше значений в формулу 4.14 получено:

Предполагается, что в оплату труда по проекту должна быть заложена в том числе, и дополнительная заработная плата ДЗП, которую необходимо будет выплачивать тем же самым работникам, например, в связи с очередным отпуском, то ее расчет будем осуществлять исходя из того, что ее размер должен составлять какую-то разумную обоснованную долю от основной заработной платы.

Известно, что большинство работников находятся в очередных плановых отпусках в среднем около одного месяца в году, то логично предположить, что доля дополнительной заработной платы ДЗП будет около 1/12 от основной заработной платы, что составляет приблизительно 8-9%. Логично допустить, что кроме отпусков работникам в соответствии с действующим законодательством предстоит делать еще какие-либо выплаты в виде дополнительной заработной платы. По этой причине полученную долю можно несколько увеличить и выбирать ее в размере 9-12%. Таким образом, сумму затрат на дополнительную заработную плату ДЗП, руб., можно рассчитать по формуле 4.18:

, (4.18)

где ОЗП – основная заработная плата всех работников, участвовавших в проекте, руб.;

КДЗП – коэффициент, %.

В рамках проекта КДЗП выбран равным 10%.

Подстановкой указанных выше значений в формулу 4.18 получено:

Подстановкой указанных выше значений в формулу 4.13 получено:

Шестым компонентом, включаемым в сметный расчет, являются страховые взносы.

Социальное страхование – это система социальной защиты, задача которой – обеспечивать реализацию конституционного права экономически активных граждан на материальное обеспечение в старости, в случае болезни, полной или частичной утраты трудоспособности, потери кормильца, безработицы [10].

Уплата страховых взносов является прямой обязанностью организации-работодателя, при этом часть выплат носит персональный характер. Так, например, выплат на лицевой счет каждого из работников в пенсионном фонде зависит от размера его персональной заработной платы.

Общий размер страховых взносов СВ, руб., необходимых для перечисления во все страховые фонды и включаемый в состав затрат при производстве, можно вычислить как некоторый процент от фонда заработной платы, для расчёта можно воспользоваться формулой 4.19:

, (4.19)

где ЗП – фонд оплаты труда по проекту, рассчитанный ранее (руб.), составляет 38 823,53 руб.;

ССВ – ставка (размер налога) страховых взносов, %.

Ставка страховых взносов ССВ на 2022 год составляет 30,2%.

Подстановкой указанных выше значений в формулу 4.19 получено:

Следующим этапом сметного расчета является накладные расходы.

В зависимости от размеров предприятия или организации накладные расходы НР, руб., могут варьироваться в достаточно широком диапазоне. Поэтому в обычной практике в сметных расчетах их принято отражать в процентном отношении к заработной плате основных работников, для расчёта можно воспользоваться формулой 4.20:

, (4.20)

где ЗП – фонд оплаты труда по проекту, руб.;

СНР – размер накладных расходов, %.

Размер накладных расходов СНР составляет 10%.

Подстановкой указанных выше значений в формулу 4.20 получено:

Восьмым этапом сметного расчета является расчет себестоимости.

Себестоимость – это текущие затраты на производство товара или услуги, запуск их в обращение и реализацию [11]. Расчет и анализ себестоимости продукции является важнейшей задачей любого предприятия и входит в систему управленческого учета, так как именно себестоимость лежит в основе большинства управленческих решений.

Себестоимость Сб, руб. программного продукта определяется по формуле 4.21.

*Сб* = *М* + ВМ + ЭР + АМ + *ЗП* + *СВ* +*НР,* (4.21)

где *М* – затраты на материалы руб.;

*ВМ* – затраты на вспомогательные материалы, руб.;

*ЭР* – энергетические затраты, руб.;

*АМ* – затраты на амортизацию, руб.;

*ЗП* – затраты на заработную плату, руб.;

*СВ* – страховые взносы, уплаченные работодателем с фонда оплаты труда, руб.;

*НР* – накладные расходы, возникающие как непредусмотренные всеми предшествующими статьями, руб.;

Подстановкой указанных выше значений в формулу 4.21 получено:

После расчета себестоимости (всех возможных затрат на разработку и производство), к ней добавляется необходимый объем прибыли, он и будет являться девятым продуктом сметного расчета.

Объем прибыли обычно определяется как процент от рассчитанной себестоимости. Объем прибыли П, руб. можно рассчитать по формуле 4.22:

, (4.22)

где Сб – себестоимость производства или разработки, руб.;

СП – процент прибыли, на которую рассчитывает организатор коммерческого предприятия, %.

Процент прибыли СП, на которую рассчитывает организатор коммерческого предприятия составляет 30%.

Подстановкой указанных выше значений в формулу 4.22 получено:

Следующим этапом сметного расчета является налог на добавленную стоимость.

Налог на добавленную стоимость (НДС) – косвенный налог, форма изъятия в бюджет государства части стоимости товара, работы или услуги, которая создаётся на всех стадиях процесса производства товаров, работ и услуг и вносится в бюджет по мере реализации [12].

Для расчета налога на добавленную стоимость НДС, руб. можно воспользоваться формулой 4.23:

, (4.23)

где Сб – себестоимость разработки, руб.;

П – прибыль, руб.;

СНДС – ставка налога на добавленную стоимость, %.

Ставка налога СНДС на добавленную стоимость составляет 20%.

Подстановкой указанных выше значений в формулу 4.23 получено:

Конечная стоимость разработки *С*, руб. рассчитывается как сумма себестоимости, прибыли и НДС, для расчёта можно воспользоваться формулой 4.24:

, (4.24)

где Сб – себестоимость разработки, руб.

П – прибыль. руб.;

НДС – сумма налога на добавленную стоимость, руб.;

Подстановкой указанных выше значений в формулу 4.24 получено:

Исходя из вышеперечисленных расчетов следует, что программный продукт является конкурентоспособным, его стоимость составляет руб.

Данный продукт является востребованным для пользователей, так как функциональность приложения несколько больше, чем у аналогов, а также в нем обеспечена безопасная передача и хранения сообщений и фотографий.

5 Раздел охраны труда

Безопасность жизнедеятельности — наука о комфортном и безопасном взаимодействии человека с техносферой, представляет собой область научных знаний, изучающая опасности, угрожающие человеку и разрабатывающие способы защиты от них в любых условиях обитания человека.

Для комфортной работы в помещении, где находится программист, должны быть определенные показатели температуры и влажности. Именно они создают микроклимат, который влияет на теплообмен с окружающей средой, самочувствие и работоспособность. В теплый период года в комнате, где работает программист, должно быть +20… 25°С, в холодный – +18… 21°С, а в переходный – +17… 21°С. Что касается влажности, то оптимальным показателем является от 40% до 60%. Чтобы организовать комфортные условия, уделяют внимание отопительной системе, вентиляции, кондиционированию воздуха. А также необходимо рационально спланировать режим дня с учетом времени года, чтобы происходило чередование труда с отдыхом.

Программист проводит на своем рабочем месте большую часть рабочего дня, поэтому организация этого пространства влияет и на его трудовую деятельность, и на его здоровье. Рабочая поза сидя вызывает минимальное утомление инженера-программиста. Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, должно быть расположено в зоне легкой досягаемости рабочего пространства. По этой причине важно правильно подобрать стол и другую мебель для работы программиста, позаботиться о микроклимате, вентиляции и хорошем освещении. Далее будут рассмотрены основные нюансы организации рабочего места инженера-программиста.

Подбор подходящей мебели помогает значительно увеличить производительность труда и снижает риск для здоровья программиста. При организации рабочего места важно, чтобы:

* мебель не ограничивала свободу движений в комнате и на рабочем месте, позволяла часто менять положение тела;
* все предметы были размещены оптимально (те, что нужны для работы чаще, должны быть ближе);
* рабочая поза не вызывала быстрого утомления;
* мебель соответствовала антропометрическим параметрам, регулировалась по высоте.

Основной мебелью рабочего места программиста являются кресло и стол. У кресла должна быть немного вогнутая поверхность и незначительный наклон спинки назад. Его высота должна изменяться, а вся конструкция не должна мешать свободе движений корпуса и рук. Желательно, чтобы у кресла имелись подлокотники.

Для комфортной работы стол должен удовлетворять следующим условиям:

* высота стола должна быть выбрана с учетом возможности сидеть свободно, в удобной позе, при необходимости опираясь на подлокотники;
* нижняя часть стола должна быть сконструирована так, чтобы программист мог удобно сидеть, не был вынужден поджимать ноги;
* поверхность стола должна обладать свойствами, исключающими появление бликов в поле зрения программиста;
* конструкция стола должна предусматривать наличие выдвижных ящиков (не менее 3 для хранения документации, листингов, канцелярских принадлежностей);
* высота рабочей поверхности рекомендуется в пределах 680-760мм;
* высота поверхности, на которую устанавливается клавиатура, должна быть около 650мм.

На рисунке 3.11 показан пример размещения основных и периферийных составляющих ПК на рабочем столе программиста., где 1 – сканер, 2 – монитор, 3 – принтер, 4 – поверхность рабочего стола, 5 – клавиатура, 6 – мышь.

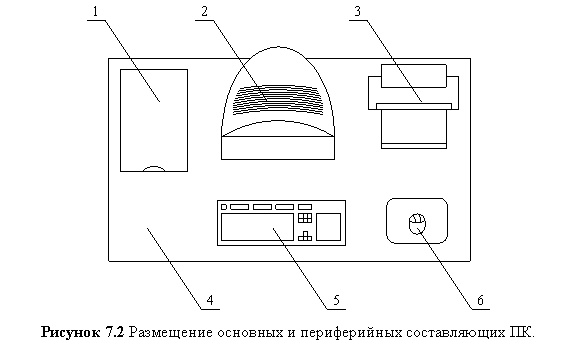


Рисунок 3.11 – Расположение основных и периферийных составляющих ПК.

При работе с вычислительной техникой могут возникнуть различные чрезвычайные ситуации. Наиболее вероятная чрезвычайная ситуация которая может возникнуть при работе с ПЭВМ – пожар, так как в современных ЭВМ очень высокая плотность размещения элементов электронных схем, в непосредственной близости друг от друга располагаются соединительные провода и кабели, при протекании по ним электрического тока выделяется значительное количество теплоты, при этом возможно оплавление изоляции и возникновение возгорания. Возникновение других видов ЧС – маловероятно.

Основным фактором, от которого зависит пожарная безопасность помещения, является автоматическое средство обнаружения пожаров. Оно позволяет оповестить дежурный персонал о пожаре и месте его возникновения. Данная система предназначена для обнаружения начальной стадии пожара и при необходимости включения автоматических систем пожаротушения и дымоудаления. Так как в помещении, в котором разрабатывается ПО, находится вычислительная техника и радиоаппаратура, то следует установить дымовой пожароизвещатель.

Исходя из норм пожарной безопасности, для машинного зала площадью до 100 м2 требуются следующие первичные средства пожаротушения:

* один углекислотный огнетушитель типа ОУ-5 или ОУ-8, с помощью которого можно тушить загорания различных материалов и установок напряжением до 1000 В;
* один порошковый огнетушитель.

При единовременном нахождении на этаже более 10 человек должны быть разработаны и вывешены на видных местах планы (схемы) эвакуации людей в случае пожара. План эвакуации должен включать в себя графическую и текстовую часть. Графическая часть представляет собой чертеж поэтажных планов здания, которые не должны загромождаться второстепенными деталями. Все работники должны пройти инструктаж по плану эвакуации в случае пожара, и должны уметь пользоваться схемой эвакуации и знать значения опознавательных знаков. Также необходимой мерой является инструктаж персонала по электробезопасности. Сюда же следует отнести правильную эксплуатацию техники, указанную в технических паспортах.

Для предотвращения поражения электрическим током предусмотрено наличие провода защитного заземления в электрической розетке, либо наличие заземляющего контура для внешнего заземления.

В случае поражения работника электрическим током, необходимо быстро освободить его от действия электрического тока. Приближаться к человеку, попавшему в зону поражения током, нужно минимальными шажками, чтобы не попасть под воздействие шагового напряжения. Шаговое напряжение – это напряжение, обусловленное электрическим током, протекающим в земле или токопроводящем полу, и равное разности потенциалов между двумя точками поверхности земли (пола), находящимися на расстоянии одного шага человека. Прикасаться к человеку, находящемуся под напряжением, опасно для жизни. Для освобождения пострадавшего от воздействия электрического тока следует воспользоваться сухой одеждой, доской или каким-либо другим предметом, не проводящим электрический ток, избегая при этом прикосновения к металлическим предметам и открытым частям тела. Далее необходимо оказать пострадавшему первую доврачебную помощь. Вызов скорой медицинской помощи обязателен независимо от того в сознании пострадавший или нет.

Заключение

В результате работы над проектом было получено мобильное приложение для обмена текстовыми сообщениями, а также был получен опыт разработки приложений под Android, верстки интерфейса для приложения, а также работы с нереляционной системы управления базами данных Firebase.

Разработанное мобильное приложение позволит прикладным программистам использовать в своих программных продуктах алгоритмы авторизации, регистрации, выхода, отправки сообщений, метод кодировки фотографии, а также алгоритм поиска пользователя в системе. Помимо этого, вся информация о пользователях, диалогах и сообщениях хранится в нереляционной базе данных, посредством чего прикладной программист сможет ознакомиться с основными операциями по работе с такими базами данных.

В современном мире данная разработка даст возможность пользователю вести безопасное общение с людьми, а также использовать приложение как аналог другим мессенджерам с более простым и интуитивно понятным интерфейсом.

В дальнейшем данное приложение можно модифицировать, добавив в него настройку профиля пользователя, возможность совершать аудио и видеозвонки, отправку кода подтверждения аккаунта на почту в целях избегания регистрации огромного количества пользователей с несуществующей почтой, а также отправку голосовых и видеосообщений.

Список использованных источников

1. <https://pandia.ru/text/77/412/193.php>
2. <https://kitapp.pro/android-studio-vs-eclipse-preimushhestvo-kazhdoj-iz-sred/>
3. <https://scienceforum.ru/2019/article/2018010389>
4. <https://ipadstory.ru/plyusy-i-minusy-whatsapp-dostoinstva-i-nedostatki.html>
5. <https://w-hatsapp.ru/preimushestva-plusy-minusy/>
6. <https://code.tutsplus.com/ru/tutorials/getting-started-with-cloud-firestore-for-android--cms-30382>
7. <https://firebase.google.com/docs/reference/android/com/google/firebase/firestore/FirebaseFirestore>
8. <https://russianblogs.com/article/62142250186/>
9. <https://aws.amazon.com/ru/nosql/>
10. <https://infourok.ru/user/3482220/blog/bezopasnost-zhiznedeyatelnosti-128391.html>
11. <https://vplate.ru/programmist/rabochee-mesto/>

Приложение А

Код класса MainActivity

package com.chat.tutorial.v1.activities;

import androidx.appcompat.app.AppCompatActivity;

import android.annotation.SuppressLint;

import android.content.Intent;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.os.Bundle;

import android.util.Base64;

import android.view.View;

import android.widget.Toast;

import com.chat.tutorial.v1.adapters.RecentConversationsAdapter;

import com.chat.tutorial.v1.databinding.ActivityMainBinding;

import com.chat.tutorial.v1.listeners.ConversionListener;

import com.chat.tutorial.v1.models.ChatMessage;

import com.chat.tutorial.v1.models.User;

import com.chat.tutorial.v1.utilities.Constants;

import com.chat.tutorial.v1.utilities.PreferenceManager;

import com.google.firebase.firestore.DocumentChange;

import com.google.firebase.firestore.DocumentReference;

import com.google.firebase.firestore.EventListener;

import com.google.firebase.firestore.FieldValue;

import com.google.firebase.firestore.FirebaseFirestore;

import com.google.firebase.firestore.QuerySnapshot;

import com.google.firebase.messaging.FirebaseMessaging;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Objects;

public class MainActivity extends BaseActivity implements ConversionListener {

private ActivityMainBinding binding;

private PreferenceManager preferenceManager;

private List<ChatMessage> conversations;

private RecentConversationsAdapter conversationsAdapter;

private FirebaseFirestore database;

private Boolean isReceiverAvailable;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

binding = ActivityMainBinding.inflate(getLayoutInflater());

setContentView(binding.getRoot());

preferenceManager = new PreferenceManager(getApplicationContext());

init();

loadUserDetails();

getToken();

setListeners();

listenConversations();

}

private void init() {

conversations = new ArrayList<>();

conversationsAdapter = new RecentConversationsAdapter(conversations, this);

binding.conversationRecyclerView.setAdapter(conversationsAdapter);

database = FirebaseFirestore.getInstance();

}

private void setListeners() {

binding.imageSignOut.setOnClickListener(v -> signOut());

binding.fabNewChat.setOnClickListener(v ->

startActivity(new Intent(getApplicationContext(), UsersActivity.class)));

}

private void listenConversations() {

database.collection(Constants.KEY\_COLLECTIONS\_CONVERSATIONS)

.whereEqualTo(Constants.KEY\_SENDER\_ID, preferenceManager.getString(Constants.KEY\_USER\_ID))

.addSnapshotListener(eventLister);

database.collection(Constants.KEY\_COLLECTIONS\_CONVERSATIONS)

.whereEqualTo(Constants.KEY\_RECEIVER\_ID, preferenceManager.getString(Constants.KEY\_USER\_ID))

.addSnapshotListener(eventLister);

}

@SuppressLint("NotifyDataSetChanged")

private final EventListener<QuerySnapshot> eventLister = (value, error) -> {

if (error != null) {

return;

}

if (value != null) {

for (DocumentChange documentChange : value.getDocumentChanges()) {

if (documentChange.getType() == DocumentChange.Type.ADDED) {

String senderID = documentChange.getDocument().getString(Constants.KEY\_SENDER\_ID);

String receiverID = documentChange.getDocument().getString(Constants.KEY\_RECEIVER\_ID);

ChatMessage chatMessage = new ChatMessage();

chatMessage.senderID = senderID;

chatMessage.receiverID = receiverID;

if (preferenceManager.getString(Constants.KEY\_USER\_ID).equals(senderID)) {

chatMessage.conversionID = documentChange.getDocument().getString(Constants.KEY\_RECEIVER\_ID);

chatMessage.conversionName = documentChange.getDocument().getString(Constants.KEY\_RECEIVER\_NAME);

chatMessage.conversionImage = documentChange.getDocument().getString(Constants.KEY\_RECEIVER\_IMAGE);

} else {

chatMessage.conversionID = documentChange.getDocument().getString(Constants.KEY\_SENDER\_ID);

chatMessage.conversionName = documentChange.getDocument().getString(Constants.KEY\_SENDER\_NAME);

chatMessage.conversionImage = documentChange.getDocument().getString(Constants.KEY\_SENDER\_IMAGE);

}

chatMessage.message = documentChange.getDocument().getString(Constants.KEY\_LAST\_MESSAGE);

chatMessage.dateObject = documentChange.getDocument().getDate(Constants.KEY\_TIMESTAMP);

conversations.add(chatMessage);

} else if (documentChange.getType() == DocumentChange.Type.MODIFIED) {

for (int i = 0; i < conversations.size(); i++) {

String senderID = documentChange.getDocument().getString(Constants.KEY\_SENDER\_ID);

String receiverID = documentChange.getDocument().getString(Constants.KEY\_RECEIVER\_ID);

if (conversations.get(i).senderID.equals(senderID) && conversations.get(i).receiverID.equals(receiverID)) {

conversations.get(i).message = documentChange.getDocument().getString(Constants.KEY\_LAST\_MESSAGE);

conversations.get(i).dateObject = documentChange.getDocument().getDate(Constants.KEY\_TIMESTAMP);

break;

}

}

}

}

Collections.sort(conversations, (obj1, obj2) -> obj2.dateObject.compareTo(obj1.dateObject));

conversationsAdapter.notifyDataSetChanged();

binding.conversationRecyclerView.smoothScrollToPosition(0);

binding.conversationRecyclerView.setVisibility(View.VISIBLE);

binding.progressBar.setVisibility(View.GONE);

}

};

private void getToken() {

FirebaseMessaging.getInstance().getToken().addOnSuccessListener(this::updateToken);

}

private void updateToken(String token) {

FirebaseFirestore database = FirebaseFirestore.getInstance();

DocumentReference documentReference =

database.collection(Constants.KEY\_COLLECTION\_USERS).document(

preferenceManager.getString(Constants.KEY\_USER\_ID)

);

documentReference.update(Constants.KEY\_FCM\_TOKEN, token);

// .addOnSuccessListener(unused -> showToast("Токен успешно обновлен"))

// .addOnFailureListener(e -> showToast("Не получилось обновить токен"));

}

private void loadUserDetails() {

binding.textName.setText(preferenceManager.getString(Constants.KEY\_NAME));

byte[] bytes = Base64.decode(preferenceManager.getString(Constants.KEY\_IMAGE), Base64.DEFAULT);

Bitmap bitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);

binding.imageProfile.setImageBitmap(bitmap);

}

private void showToast(String message) {

Toast.makeText(getApplicationContext(), message, Toast.LENGTH\_SHORT).show();

}

private void signOut() {

showToast("Выход...");

FirebaseFirestore database = FirebaseFirestore.getInstance();

DocumentReference documentReference =

database.collection(Constants.KEY\_COLLECTION\_USERS).document(

preferenceManager.getString(Constants.KEY\_USER\_ID)

);

Map<String, Object> updates = new HashMap<>();

updates.put(Constants.KEY\_FCM\_TOKEN, FieldValue.delete());

documentReference.update(updates)

.addOnSuccessListener(unused -> {

preferenceManager.clear();

startActivity(new Intent(getApplicationContext(), SignInActivity.class));

finish();

})

.addOnFailureListener(e -> showToast("Не получилось выйти"));

}

@Override

public void onConversionClicked(User user) {

Intent intent = new Intent(getApplicationContext(), ChatActivity.class);

intent.putExtra(Constants.KEY\_USER, user);

startActivity(intent);

}

}

Код класса SignUpActivity

package com.chat.tutorial.v1.activities;

import android.content.Intent;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.net.Uri;

import android.os.Bundle;

import android.provider.MediaStore;

import android.util.Base64;

import android.util.Patterns;

import android.view.View;

import android.widget.Toast;

import androidx.activity.result.ActivityResultLauncher;

import androidx.activity.result.contract.ActivityResultContracts;

import androidx.appcompat.app.AppCompatActivity;

import com.chat.tutorial.v1.databinding.ActivitySignUpBinding;

import com.chat.tutorial.v1.utilities.Constants;

import com.chat.tutorial.v1.utilities.PreferenceManager;

import com.google.firebase.firestore.FirebaseFirestore;

import java.io.ByteArrayOutputStream;

import java.io.FileNotFoundException;

import java.io.InputStream;

import java.util.HashMap;

import java.util.Map;

public class SignUpActivity extends AppCompatActivity {

private ActivitySignUpBinding binding;

private PreferenceManager preferenceManager;

private String encodedImage;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

binding = ActivitySignUpBinding.inflate(getLayoutInflater());

setContentView(binding.getRoot());

preferenceManager = new PreferenceManager(getApplicationContext());

setListeners();

}

private void setListeners() {

binding.textSignIn.setOnClickListener(view -> onBackPressed());

binding.buttonSignUp.setOnClickListener(v -> {

if (isValidSignUpDetails()) {

signUp();

}

});

binding.layoutImage.setOnClickListener(v -> {

Intent intent = new Intent(Intent.ACTION\_PICK, MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI);

intent.addFlags(Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION);

pickImage.launch(intent);

});

}

private void showToast(String message) {

Toast.makeText(getApplicationContext(), message, Toast.LENGTH\_SHORT).show();

}

private void signUp() {

loading(true);

FirebaseFirestore database = FirebaseFirestore.getInstance();

Map<String, Object> user = new HashMap<>();

user.put(Constants.KEY\_NAME, binding.inputName.getText().toString());

user.put(Constants.KEY\_EMAIL, binding.inputEmail.getText().toString());

user.put(Constants.KEY\_PASSWORD, binding.inputPassword.getText().toString());

user.put(Constants.KEY\_IMAGE, encodedImage);

database.collection(Constants.KEY\_COLLECTION\_USERS)

.add(user)

.addOnSuccessListener(documentReference -> {

loading(false);

preferenceManager.putBoolean(Constants.KEY\_IS\_SIGNED\_IN, true);

preferenceManager.putString(Constants.KEY\_USER\_ID, documentReference.getId());

preferenceManager.putString(Constants.KEY\_NAME, binding.inputName.getText().toString());

preferenceManager.putString(Constants.KEY\_IMAGE, encodedImage);

Intent intent = new Intent(getApplicationContext(), MainActivity.class);

intent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK | Intent.FLAG\_ACTIVITY\_CLEAR\_TASK);

startActivity(intent);

})

.addOnFailureListener(exception -> {

loading(false);

showToast(exception.getMessage());

});

}

private String encodeImage(Bitmap bitmap) {

int previewWidth = 150;

int previewHeight = bitmap.getHeight() \* previewWidth / bitmap.getWidth();

Bitmap previewBitmap = Bitmap.createScaledBitmap(bitmap, previewWidth, previewHeight, false);

ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

previewBitmap.compress(Bitmap.CompressFormat.JPEG, 50, byteArrayOutputStream);

byte[] bytes = byteArrayOutputStream.toByteArray();

return Base64.encodeToString(bytes, Base64.DEFAULT);

}

private final ActivityResultLauncher<Intent> pickImage = registerForActivityResult(

new ActivityResultContracts.StartActivityForResult(),

result -> {

if (result.getResultCode() == RESULT\_OK) {

if (result.getData() != null) {

Uri imageUri = result.getData().getData();

try {

InputStream inputStream = getContentResolver().openInputStream(imageUri);

Bitmap bitmap = BitmapFactory.decodeStream(inputStream);

binding.imageProfile.setImageBitmap(bitmap);

binding.textAddImage.setVisibility(View.GONE);

encodedImage = encodeImage(bitmap);

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

}

}

);

private Boolean isValidSignUpDetails() {

// if (encodedImage == null) {

// showToast("Выберите фото профиля");

// return false;

// } else

if (binding.inputName.getText().toString().trim().isEmpty()) {

showToast("Введите свое имя");

return false;

} else if (binding.inputEmail.getText().toString().trim().isEmpty()) {

showToast("Введите свой email адрес");

return false;

} else if (!Patterns.EMAIL\_ADDRESS.matcher(binding.inputEmail.getText().toString()).matches()) {

showToast("Введите правильный email адрес");

return false;

} else if (binding.inputPassword.getText().toString().trim().isEmpty()) {

showToast("Введите пароль");

return false;

} else if (binding.inputConfirmPassword.getText().toString().trim().isEmpty()) {

showToast("Подтвердите свой пароль");

return false;

} else if (!binding.inputPassword.getText().toString().equals(binding.inputConfirmPassword.getText().toString())) {

showToast("Пароли должны совпадать");

return false;

} else {

return true;

}

}

private void loading(Boolean isLoading) {

if (isLoading) {

binding.buttonSignUp.setVisibility(View.INVISIBLE);

binding.progressBar.setVisibility(View.VISIBLE);

} else {

binding.buttonSignUp.setVisibility(View.VISIBLE);

binding.progressBar.setVisibility(View.INVISIBLE);

}

}

}

Код класса SignInActivity

package com.chat.tutorial.v1.activities;

import android.content.Intent;

import android.os.Bundle;

import android.util.Patterns;

import android.view.View;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import com.chat.tutorial.v1.databinding.ActivitySignInBinding;

import com.chat.tutorial.v1.utilities.Constants;

import com.chat.tutorial.v1.utilities.PreferenceManager;

import com.google.firebase.firestore.DocumentSnapshot;

import com.google.firebase.firestore.FirebaseFirestore;

public class SignInActivity extends AppCompatActivity {

private ActivitySignInBinding binding;

private PreferenceManager preferenceManager;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

preferenceManager = new PreferenceManager(getApplicationContext());

if (preferenceManager.getBoolean(Constants.KEY\_IS\_SIGNED\_IN)) {

Intent intent = new Intent(getApplicationContext(), MainActivity.class);

startActivity(intent);

finish();

}

binding = ActivitySignInBinding.inflate(getLayoutInflater());

setContentView(binding.getRoot());

setListeners();

}

private void setListeners() {

binding.textCreateNewAccount.setOnClickListener(v ->

startActivity(new Intent(getApplicationContext(), SignUpActivity.class)));

binding.buttonSignIn.setOnClickListener(v -> {

if (isValidSignInDetails()) {

signIn();

}

});

}

private void signIn() {

loading(true);

FirebaseFirestore database = FirebaseFirestore.getInstance();

database.collection(Constants.KEY\_COLLECTION\_USERS)

.whereEqualTo(Constants.KEY\_EMAIL, binding.inputEmail.getText().toString())

.whereEqualTo(Constants.KEY\_PASSWORD, binding.inputPassword.getText().toString())

.get()

.addOnCompleteListener(task -> {

if (task.isSuccessful() && task.getResult() != null

&& task.getResult().getDocuments().size() > 0) {

DocumentSnapshot documentSnapshot = task.getResult().getDocuments().get(0);

preferenceManager.putBoolean(Constants.KEY\_IS\_SIGNED\_IN, true);

preferenceManager.putString(Constants.KEY\_USER\_ID, documentSnapshot.getId());

preferenceManager.putString(Constants.KEY\_NAME, documentSnapshot.getString(Constants.KEY\_NAME));

preferenceManager.putString(Constants.KEY\_IMAGE, documentSnapshot.getString(Constants.KEY\_IMAGE));

Intent intent = new Intent(getApplicationContext(), MainActivity.class);

intent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK | Intent.FLAG\_ACTIVITY\_CLEAR\_TASK);

startActivity(intent);

} else {

loading(false);

showToast("Не удалось выполнить вход.");

}

});

}

private void loading(Boolean isLoading) {

if (isLoading) {

binding.buttonSignIn.setVisibility(View.INVISIBLE);

binding.progressBar.setVisibility(View.VISIBLE);

} else {

binding.buttonSignIn.setVisibility(View.VISIBLE);

binding.progressBar.setVisibility(View.INVISIBLE);

}

}

private void showToast(String message) {

Toast.makeText(getApplicationContext(), message, Toast.LENGTH\_LONG).show();

}

private Boolean isValidSignInDetails() {

if (binding.inputEmail.getText().toString().trim().isEmpty()) {

showToast("Введите свой email адрес");

return false;

} else if (!Patterns.EMAIL\_ADDRESS.matcher(binding.inputEmail.getText().toString()).matches()) {

showToast("Введите правильный email адрес");

return false;

} else if (binding.inputPassword.getText().toString().trim().isEmpty()) {

showToast("Введите пароль");

return false;

} else {

return true;

}

}

}

Код класса UsersActivity

package com.chat.tutorial.v1.activities;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.os.Bundle;

import android.text.Editable;

import android.text.TextWatcher;

import android.view.View;

import com.chat.tutorial.v1.listeners.UserListener;

import com.chat.tutorial.v1.adapters.UserAdapter;

import com.chat.tutorial.v1.databinding.ActivityUsersBinding;

import com.chat.tutorial.v1.models.User;

import com.chat.tutorial.v1.utilities.Constants;

import com.chat.tutorial.v1.utilities.PreferenceManager;

import com.google.firebase.firestore.FirebaseFirestore;

import com.google.firebase.firestore.QueryDocumentSnapshot;

import java.util.ArrayList;

public class UsersActivity extends BaseActivity implements UserListener {

private ActivityUsersBinding binding;

private PreferenceManager preferenceManager;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

binding = ActivityUsersBinding.inflate(getLayoutInflater());

setContentView(binding.getRoot());

preferenceManager = new PreferenceManager(getApplicationContext());

setListeners();

getUsers();

}

private void setListeners() {

binding.imageBack.setOnClickListener(v -> onBackPressed());

binding.imageSearch.setOnClickListener(v -> {

searchByName(true);

});

binding.searchUser.addTextChangedListener(new TextWatcher() {

@Override

public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {

}

@Override

public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {

getUsers();

}

@Override

public void afterTextChanged(Editable editable) {

}

});

}

private void getUsers() {

loading(true);

FirebaseFirestore database = FirebaseFirestore.getInstance();

database.collection(Constants.KEY\_COLLECTION\_USERS)

.get()

.addOnCompleteListener(task -> {

loading(false);

String currentUserId = preferenceManager.getString(Constants.KEY\_USER\_ID);

if (task.isSuccessful() && task.getResult() != null) {

ArrayList<User> users = new ArrayList<>();

for (QueryDocumentSnapshot queryDocumentSnapshot : task.getResult()) {

if (currentUserId.equals(queryDocumentSnapshot.getId())) continue;

if (queryDocumentSnapshot.getString(Constants.KEY\_NAME).contains(binding.searchUser.getText().toString())) {

User user = new User();

user.name = queryDocumentSnapshot.getString(Constants.KEY\_NAME);

user.email = queryDocumentSnapshot.getString(Constants.KEY\_EMAIL);

user.image = queryDocumentSnapshot.getString(Constants.KEY\_IMAGE);

user.token = queryDocumentSnapshot.getString(Constants.KEY\_FCM\_TOKEN);

user.id = queryDocumentSnapshot.getId();

users.add(user);

}

}

if (users.size() > 0) {

UserAdapter userAdapter = new UserAdapter(users, this);

binding.usersRecyclerView.setAdapter(userAdapter);

binding.usersRecyclerView.setVisibility(View.VISIBLE);

binding.textErrorMessage.setVisibility(View.GONE);

} else {

showErrorMessage();

// binding.usersRecyclerView.setAdapter(new UserAdapter(null, this));

binding.usersRecyclerView.setVisibility(View.GONE);

}

} else showErrorMessage();

});

}

private void searchByName(Boolean byName) {

if (byName) {

binding.imageSearch.setVisibility(View.INVISIBLE);

binding.searchUser.setVisibility(View.VISIBLE);

} else {

binding.imageSearch.setVisibility(View.VISIBLE);

binding.searchUser.setVisibility(View.INVISIBLE);

}

}

private void showErrorMessage() {

binding.textErrorMessage.setText(String.format("%s", "Пользователей нет"));

binding.textErrorMessage.setVisibility(View.VISIBLE);

}

private void loading(Boolean isLoading) {

if (isLoading) {

binding.progressBar.setVisibility(View.VISIBLE);

} else {

binding.progressBar.setVisibility(View.INVISIBLE);

}

}

@Override

public void onUserClick(User user) {

Intent intent = new Intent(getApplicationContext(), ChatActivity.class);

intent.putExtra(Constants.KEY\_USER, user);

startActivity(intent);

finish();

}

}

Код класса ChatActivity

package com.chat.tutorial.v1.activities;

import android.annotation.SuppressLint;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.os.Bundle;

import android.util.Base64;

import android.view.View;

import com.chat.tutorial.v1.adapters.ChatAdapter;

import com.chat.tutorial.v1.databinding.ActivityChatBinding;

import com.chat.tutorial.v1.models.ChatMessage;

import com.chat.tutorial.v1.models.User;

import com.chat.tutorial.v1.utilities.Constants;

import com.chat.tutorial.v1.utilities.PreferenceManager;

import com.google.android.gms.tasks.OnCompleteListener;

import com.google.firebase.firestore.DocumentChange;

import com.google.firebase.firestore.DocumentReference;

import com.google.firebase.firestore.DocumentSnapshot;

import com.google.firebase.firestore.EventListener;

import com.google.firebase.firestore.FirebaseFirestore;

import com.google.firebase.firestore.QuerySnapshot;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Locale;

import java.util.Map;

import java.util.Objects;

public class ChatActivity extends BaseActivity {

private ActivityChatBinding binding;

private User receiverUser;

private List<ChatMessage> chatMessages;

private ChatAdapter chatAdapter;

private PreferenceManager preferenceManager;

private FirebaseFirestore database;

private String conversionID = null;

private Boolean isReceiverAvailable = false;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

binding = ActivityChatBinding.inflate(getLayoutInflater());

setContentView(binding.getRoot());

setListeners();

loadReceiverDetails();

init();

listenMessages();

}

private void init() {

preferenceManager = new PreferenceManager(getApplicationContext());

chatMessages = new ArrayList<>();

chatAdapter = new ChatAdapter(

chatMessages,

getBitmapFromEncodedString(receiverUser.image),

preferenceManager.getString(Constants.KEY\_USER\_ID)

);

binding.chatRecyclerView.setAdapter(chatAdapter);

database = FirebaseFirestore.getInstance();

}

private void sendMessage() {

Map<String, Object> message = new HashMap<>();

message.put(Constants.KEY\_SENDER\_ID, preferenceManager.getString(Constants.KEY\_USER\_ID));

message.put(Constants.KEY\_RECEIVER\_ID, receiverUser.id);

message.put(Constants.KEY\_MESSAGE, binding.inputMessage.getText().toString().trim());

message.put(Constants.KEY\_TIMESTAMP, new Date());

database.collection(Constants.KEY\_COLLECTION\_CHAT).add(message);

if (conversionID != null) {

updateConversion(binding.inputMessage.getText().toString());

} else {

HashMap<String, Object> conversion = new HashMap<>();

conversion.put(Constants.KEY\_SENDER\_ID, preferenceManager.getString(Constants.KEY\_USER\_ID));

conversion.put(Constants.KEY\_SENDER\_NAME, preferenceManager.getString(Constants.KEY\_NAME));

conversion.put(Constants.KEY\_SENDER\_IMAGE, preferenceManager.getString(Constants.KEY\_IMAGE));

conversion.put(Constants.KEY\_RECEIVER\_ID, receiverUser.id);

conversion.put(Constants.KEY\_RECEIVER\_NAME, receiverUser.name);

conversion.put(Constants.KEY\_RECEIVER\_IMAGE, receiverUser.image);

conversion.put(Constants.KEY\_LAST\_MESSAGE, binding.inputMessage.getText().toString().trim());

conversion.put(Constants.KEY\_TIMESTAMP, new Date());

addConversion(conversion);

}

binding.inputMessage.setText(null);

}

private void listenAvailabilityOfReceiver() {

database.collection(Constants.KEY\_COLLECTION\_USERS)

.document(receiverUser.id)

.addSnapshotListener(ChatActivity.this, (value, error) -> {

if (error != null) return;

if (value != null) {

if (value.getLong(Constants.KEY\_AVAILABILITY) != null) {

int availability = Objects.requireNonNull(

value.getLong(Constants.KEY\_AVAILABILITY)

).intValue();

isReceiverAvailable = availability == 1;

}

}

if (isReceiverAvailable) {

binding.availability.setText("онлайн");

} else {

binding.availability.setText("был когда-то");

}

});

}

private void listenMessages() {

database.collection(Constants.KEY\_COLLECTION\_CHAT)

.whereEqualTo(Constants.KEY\_SENDER\_ID, preferenceManager.getString(Constants.KEY\_USER\_ID))

.whereEqualTo(Constants.KEY\_RECEIVER\_ID, receiverUser.id)

.addSnapshotListener(eventListener);

database.collection(Constants.KEY\_COLLECTION\_CHAT)

.whereEqualTo(Constants.KEY\_SENDER\_ID, receiverUser.id)

.whereEqualTo(Constants.KEY\_RECEIVER\_ID, preferenceManager.getString(Constants.KEY\_USER\_ID))

.addSnapshotListener(eventListener);

}

private final EventListener<QuerySnapshot> eventListener = (value, error) -> {

if (error != null) {

return;

}

if (value != null) {

int count = chatMessages.size();

for (DocumentChange documentChange : value.getDocumentChanges()) {

if (documentChange.getType() == DocumentChange.Type.ADDED) {

ChatMessage chatMessage = new ChatMessage();

chatMessage.senderID = documentChange.getDocument().getString(Constants.KEY\_SENDER\_ID);

chatMessage.receiverID = documentChange.getDocument().getString(Constants.KEY\_RECEIVER\_ID);

chatMessage.message = documentChange.getDocument().getString(Constants.KEY\_MESSAGE);

chatMessage.dateTime = getReadableDateTime(documentChange.getDocument().getDate(Constants.KEY\_TIMESTAMP));

chatMessage.dateObject = documentChange.getDocument().getDate(Constants.KEY\_TIMESTAMP);

chatMessages.add(chatMessage);

}

}

Collections.sort(chatMessages, (obj1, obj2) -> obj1.dateObject.compareTo(obj2.dateObject));

if (count == 0) {

chatAdapter.notifyDataSetChanged();

} else {

chatAdapter.notifyItemRangeInserted(chatMessages.size(), chatMessages.size());

binding.chatRecyclerView.smoothScrollToPosition(chatMessages.size() - 1);

}

binding.chatRecyclerView.setVisibility(View.VISIBLE);

}

binding.progressBar.setVisibility(View.GONE);

if (conversionID == null) {

checkForConversion();

}

};

private Bitmap getBitmapFromEncodedString(String encodedImage) {

byte[] bytes = Base64.decode(encodedImage, Base64.DEFAULT);

return BitmapFactory.decodeByteArray(bytes, 0, bytes.length);

}

private void loadReceiverDetails() {

receiverUser = (User) getIntent().getSerializableExtra(Constants.KEY\_USER);

binding.textName.setText(receiverUser.name);

byte[] bytes = Base64.decode(receiverUser.image, Base64.DEFAULT);

Bitmap bitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);

binding.imageProfile.setImageBitmap(bitmap);

}

private void setListeners() {

binding.imageBack.setOnClickListener(v -> onBackPressed());

binding.layoutSend.setOnClickListener(v -> {

if (!binding.inputMessage.getText().toString().trim().isEmpty()) {

sendMessage();

}

});

// binding.chatRecyclerView.setOnLongClickListener(deleteMessage());

// deleteConversion();

}

private String getReadableDateTime(Date date) {

return new SimpleDateFormat("MMMM dd, yyyy - hh:mm", new Locale("ru")).format(date);

}

private void addConversion(HashMap<String, Object> conversion) {

database.collection(Constants.KEY\_COLLECTIONS\_CONVERSATIONS)

.add(conversion)

.addOnSuccessListener(documentReference -> conversionID = documentReference.getId());

}

private void updateConversion(String message) {

DocumentReference documentReference =

database.collection(Constants.KEY\_COLLECTIONS\_CONVERSATIONS).document(conversionID);

documentReference.update(

Constants.KEY\_LAST\_MESSAGE, message,

Constants.KEY\_TIMESTAMP, new Date()

);

// documentReference.delete();

}

@SuppressLint("NotifyDataSetChanged")

private void deleteMessage(String senderID, String receiverID) {

database.collection(Constants.KEY\_MESSAGE)

.whereEqualTo(Constants.KEY\_SENDER\_ID, senderID)

.whereEqualTo(Constants.KEY\_RECEIVER\_ID, receiverID)

.getFirestore().document(preferenceManager.getString(Constants.KEY\_MESSAGE))

.delete().addOnSuccessListener(success -> chatAdapter.notifyDataSetChanged());

}

private void checkForConversion() {

if (chatMessages.size() != 0) {

checkForConversionRemotely(

preferenceManager.getString(Constants.KEY\_USER\_ID),

receiverUser.id

);

checkForConversionRemotely(

receiverUser.id,

preferenceManager.getString(Constants.KEY\_USER\_ID)

);

}

}

private void checkForConversionRemotely(String senderID, String receiverID) {

database.collection(Constants.KEY\_COLLECTIONS\_CONVERSATIONS)

.whereEqualTo(Constants.KEY\_SENDER\_ID, senderID)

.whereEqualTo(Constants.KEY\_RECEIVER\_ID, receiverID)

.get()

.addOnCompleteListener(conversionOnCompleteListener);

}

private final OnCompleteListener<QuerySnapshot> conversionOnCompleteListener = task -> {

if (task.isSuccessful() && task.getResult() != null && task.getResult().getDocuments().size() > 0) {

DocumentSnapshot documentSnapshot = task.getResult().getDocuments().get(0);

conversionID = documentSnapshot.getId();

}

};

@Override

protected void onResume() {

super.onResume();

listenAvailabilityOfReceiver();

}

}

Код класса BaseActivity

package com.chat.tutorial.v1.activities;

import android.os.Bundle;

import androidx.annotation.Nullable;

import androidx.appcompat.app.AppCompatActivity;

import com.chat.tutorial.v1.utilities.Constants;

import com.chat.tutorial.v1.utilities.PreferenceManager;

import com.google.firebase.firestore.DocumentReference;

import com.google.firebase.firestore.FirebaseFirestore;

public class BaseActivity extends AppCompatActivity {

private DocumentReference documentReference;

@Override

protected void onCreate(@Nullable Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

PreferenceManager preferenceManager = new PreferenceManager(getApplicationContext());

FirebaseFirestore database = FirebaseFirestore.getInstance();

documentReference = database.collection(Constants.KEY\_COLLECTION\_USERS)

.document(preferenceManager.getString(Constants.KEY\_USER\_ID));

}

@Override

protected void onPause() {

super.onPause();

documentReference.update(Constants.KEY\_AVAILABILITY, 0);

}

@Override

protected void onResume() {

super.onResume();

documentReference.update(Constants.KEY\_AVAILABILITY, 1);

}

}

Код класса User

package com.chat.tutorial.v1.models;

import java.io.Serializable;

public class User implements Serializable {

public String name;

public String image;

public String email;

public String token;

public String id;

}

Код класса ChatMessage

package com.chat.tutorial.v1.models;

import java.util.Date;

public class ChatMessage {

public String senderID;

public String receiverID;

public String message;

public String dateTime;

public Date dateObject;

public String conversionID;

public String conversionName;

public String conversionImage;

}

Код класса UserAdapter

package com.chat.tutorial.v1.adapters;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.util.Base64;

import android.view.LayoutInflater;

import android.view.ViewGroup;

import androidx.annotation.NonNull;

import androidx.recyclerview.widget.RecyclerView;

import com.chat.tutorial.v1.listeners.UserListener;

import com.chat.tutorial.v1.databinding.ItemContainerUserBinding;

import com.chat.tutorial.v1.models.User;

import java.util.List;

public class UserAdapter extends RecyclerView.Adapter<UserAdapter.UserViewHolder> {

private final List<User> users;

private final UserListener userListener;

public UserAdapter(List<User> users, UserListener userListener) {

this.users = users;

this.userListener = userListener;

}

@NonNull

@Override

public UserViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

ItemContainerUserBinding itemContainerUserBinding = ItemContainerUserBinding.inflate(

LayoutInflater.from(parent.getContext()),

parent,

false

);

return new UserViewHolder(itemContainerUserBinding);

}

@Override

public void onBindViewHolder(@NonNull UserViewHolder holder, int position) {

holder.setUserData(users.get(position));

}

@Override

public int getItemCount() {

return users.size();

}

class UserViewHolder extends RecyclerView.ViewHolder {

ItemContainerUserBinding binding;

public UserViewHolder(ItemContainerUserBinding itemContainerUserBinding) {

super(itemContainerUserBinding.getRoot());

binding = itemContainerUserBinding;

}

void setUserData(User user) {

binding.textName.setText(user.name);

binding.imageProfile.setImageBitmap(getUserImage(user.image));

binding.getRoot().setOnClickListener(v -> userListener.onUserClick(user));

}

}

private Bitmap getUserImage(String encodedImage) {

byte[] bytes = Base64.decode(encodedImage, Base64.DEFAULT);

return BitmapFactory.decodeByteArray(bytes, 0, bytes.length);

// return null;

}

}

Код класса ChatAdapter

package com.chat.tutorial.v1.adapters;

import android.graphics.Bitmap;

import android.view.LayoutInflater;

import android.view.ViewGroup;

import androidx.annotation.NonNull;

import androidx.recyclerview.widget.RecyclerView;

import com.chat.tutorial.v1.databinding.ItemContainerReceivedMessageBinding;

import com.chat.tutorial.v1.databinding.ItemContainerSentMessageBinding;

import com.chat.tutorial.v1.models.ChatMessage;

import java.util.List;

public class ChatAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {

public ChatAdapter(List<ChatMessage> chatMessages, Bitmap receiverProfileImage, String senderID) {

this.chatMessages = chatMessages;

this.receiverProfileImage = receiverProfileImage;

this.senderID = senderID;

}

private final List<ChatMessage> chatMessages;

private final Bitmap receiverProfileImage;

private final String senderID;

public static final int VIEW\_TYPE\_SENT = 1;

public static final int VIEW\_TYPE\_RECIEVED = 2;

@NonNull

@Override

public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

if (viewType == VIEW\_TYPE\_SENT) {

return new SentMessageHolder(

ItemContainerSentMessageBinding.inflate(

LayoutInflater.from(parent.getContext()),

parent,

false

)

);

} else {

return new ReceiverMessageViewHolder(

ItemContainerReceivedMessageBinding.inflate(

LayoutInflater.from(parent.getContext()),

parent,

false

)

);

}

}

@Override

public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {

if (getItemViewType(position) == VIEW\_TYPE\_SENT) {

((SentMessageHolder) holder).setData(chatMessages.get(position));

} else {

((ReceiverMessageViewHolder) holder).setData(chatMessages.get(position), receiverProfileImage);

}

}

@Override

public int getItemCount() {

return chatMessages.size();

}

@Override

public int getItemViewType(int position) {

if (chatMessages.get(position).senderID.equals(senderID)) {

return VIEW\_TYPE\_SENT;

} else {

return VIEW\_TYPE\_RECIEVED;

}

}

static class SentMessageHolder extends RecyclerView.ViewHolder {

private final ItemContainerSentMessageBinding binding;

public SentMessageHolder(ItemContainerSentMessageBinding itemContainerSentMessageBinding) {

super(itemContainerSentMessageBinding.getRoot());

binding = itemContainerSentMessageBinding;

}

void setData(ChatMessage chatMessage) {

binding.textMessage.setText(chatMessage.message);

binding.textDateTime.setText(chatMessage.dateTime);

}

}

static class ReceiverMessageViewHolder extends RecyclerView.ViewHolder {

private final ItemContainerReceivedMessageBinding binding;

public ReceiverMessageViewHolder(ItemContainerReceivedMessageBinding itemContainerReceivedMessageBinding) {

super(itemContainerReceivedMessageBinding.getRoot());

binding = itemContainerReceivedMessageBinding;

}

void setData(ChatMessage chatMessage, Bitmap receivedProfileImage) {

binding.textMessage.setText(chatMessage.message);

binding.textDateTime.setText(chatMessage.dateTime);

// binding.imageProfile.setImageBitmap(receivedProfileImage);

}

}

}

Код класса RecentConversationsAdapter

package com.chat.tutorial.v1.adapters;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.util.Base64;

import android.view.LayoutInflater;

import android.view.ViewGroup;

import androidx.annotation.NonNull;

import androidx.recyclerview.widget.RecyclerView;

import com.chat.tutorial.v1.databinding.ItemContainerRecentConversionBinding;

import com.chat.tutorial.v1.listeners.ConversionListener;

import com.chat.tutorial.v1.models.ChatMessage;

import com.chat.tutorial.v1.models.User;

import java.util.List;

public class RecentConversationsAdapter extends RecyclerView.Adapter<RecentConversationsAdapter.ConversionViewHolder> {

private final List<ChatMessage> chatMessages;

private final ConversionListener conversionListener;

public RecentConversationsAdapter(List<ChatMessage> chatMessages, ConversionListener conversionListener) {

this.chatMessages = chatMessages;

this.conversionListener = conversionListener;

}

@NonNull

@Override

public ConversionViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

return new ConversionViewHolder(

ItemContainerRecentConversionBinding.inflate(

LayoutInflater.from(parent.getContext()),

parent,

false

)

);

}

@Override

public void onBindViewHolder(@NonNull ConversionViewHolder holder, int position) {

holder.setData(chatMessages.get(position));

}

@Override

public int getItemCount() {

return chatMessages.size();

}

class ConversionViewHolder extends RecyclerView.ViewHolder {

ItemContainerRecentConversionBinding binding;

ConversionViewHolder(ItemContainerRecentConversionBinding itemContainerRecentConversionBinding) {

super(itemContainerRecentConversionBinding.getRoot());

binding = itemContainerRecentConversionBinding;

}

void setData(ChatMessage chatMessage) {

binding.imageProfile.setImageBitmap(getConversionImage(chatMessage.conversionImage));

binding.textName.setText(chatMessage.conversionName);

binding.textRecentMessage.setText(chatMessage.message);

binding.getRoot().setOnClickListener(v -> {

User user = new User();

user.id = chatMessage.conversionID;

user.name = chatMessage.conversionName;

user.image = chatMessage.conversionImage;

conversionListener.onConversionClicked(user);

});

}

}

private Bitmap getConversionImage(String encodedImage) {

byte[] bytes = Base64.decode(encodedImage, Base64.DEFAULT);

return BitmapFactory.decodeByteArray(bytes, 0, bytes.length);

}

}