**РЕФЕРАТ**

Курсовая работа 30 стр., 4 ч., 17 рис., 3 источника, 5 приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ ОБЩЕНИЯ “ECO”

Целью данной работы является работка мобильного приложения для коммуникации или просто мессенджера на основе популярной платформы Android, проектирование, разработка и внедрение в приложение базы данных.

Объектом исследования в данной работе является мобильное клиент-серверное приложение.

Предметом исследования являются архитектура Android-приложения, алгоритмы разработки приложения, облачная база данных Firebase и IDE Android Studio.

Методы исследования: моделирование и описание.

Практическая ценность работы состоит в приобретении новых навыков разработки мобильных приложений и возможности запуска бета-версии этого приложения для его тестирования и использования.

СОДЕРЖАНИЕ

[Введение 5](#_Toc90391452)

[1 Чистая архитектура Android-приложений 6](#_Toc90391453)

[2 Базы данных NoSQL 8](#_Toc90391454)

[2.1 Firebase Realtime Database 9](#_Toc90391455)

[2.1.1 Ключевые особенности 9](#_Toc90391456)

[2.1.2 Применение 10](#_Toc90391457)

[2.1.3 Структурирование и хранение данных 11](#_Toc90391458)

[2.1.4 Безопасность 13](#_Toc90391459)

[2.1.4.1 Правила безопасности 13](#_Toc90391460)

[2.1.4.2 Аутентификация 14](#_Toc90391461)

[2.1.4.3 Авторизация 14](#_Toc90391462)

[2.1.4.4 Валидация данных 15](#_Toc90391463)

[2.1.4.5 Определение индексов базы данных 16](#_Toc90391464)

[2.2 Логика построения БД чата в Firebase 16](#_Toc90391465)

[2.3 Консоль Firebase 18](#_Toc90391466)

[3 Архитектура приложения Eco 21](#_Toc90391467)

[3.1 MessengerFragment 22](#_Toc90391468)

[3.2 UsersFragment 23](#_Toc90391469)

[3.3 ProfileFragment 23](#_Toc90391470)

[3.4 ChatFragment 23](#_Toc90391471)

[3.5 CreateChatFragment 24](#_Toc90391472)

[3.6 AddUserFragment 24](#_Toc90391473)

[4 Разработка приложения 25](#_Toc90391474)

[4.1 Диаграмма классов 25](#_Toc90391475)

[4.2 Интерфейс приложения 27](#_Toc90391476)

[Заключение 29](#_Toc90391477)

[Список использованных источников 30](#_Toc90391478)

[Приложение А Главная активность приложения 31](#_Toc90391479)

[Приложение Б Фрагменты 33](#_Toc90391480)

[Приложение В Адаптеры 51](#_Toc90391481)

[Приложение Г Модели данных 62](#_Toc90391482)

[Приложение Д Класс Database 64](#_Toc90391483)

# ВВЕДЕНИЕ

Актуальность данной работы обусловлена широким использованием в обществе приложений разной направленности: для работы социальных и государственных служб, игровых приложений, приложений для широкого общения и для взаимодействия в пределах служебного круга задач. Умение построить правильную, хорошо поддерживаемую архитектуру мобильного приложения является очень востребованной компетенцией молодого специалиста.

Целью данной работы является разработка мобильного приложения для коммуникации или просто мессенджера на основе популярной платформы Android, проектирование, разработка и внедрение в приложение базы данных.

Для реализации поставленной цели предполагается решить следующие задачи:

* изучить работу с облачной базой данных Google Firebase;
* спроектировать базу данных;
* изучить работу с IDE Android Studio и языком Kotlin;
* разработать проект приложения и само приложение.

Объектом исследования в данной работе является мобильное клиент-серверное приложение.

Предметом исследования являются архитектура Android-приложения, алгоритмы разработки приложения, облачная база данных Firebase и IDE Android Studio.

Методы исследования: моделирование и описание.

Практическая ценность работы состоит в приобретении новых навыков разработки мобильных приложений и возможности запуска бета-версии этого приложения для его тестирования и использования.

# 1 Чистая архитектура Android-приложений

Архитектура Android-приложений обычно не бывает слишком сложной, поэтому, возможно, схему из Clean Architecture можно упростить без потери качества. И эту схему в любом случае нужно как-то адаптировать для конкретного использования в Android.

Наиболее критичной с точки зрения тестирования приложения является бизнес-логика или бизнес-правила, определяющие суть работы приложения. И они должны быть в первую очередь независимы от других элементов и протестированы.

Чтобы достичь независимости и возможности тестирования, предлагается разбить приложение на 3 ключевых слоя:

* слой данных (Data Layer);
* слой бизнес-логики (Domain Layer);
* слой представления (Presentation Layer).

При этом чтобы обеспечить максимальную независимость этих слоев, на каждом из них используется своя модель данных, которая конвертируется при взаимодействии между слоями.

Схема этих слоев выглядит следующим образом (рисунок 1).

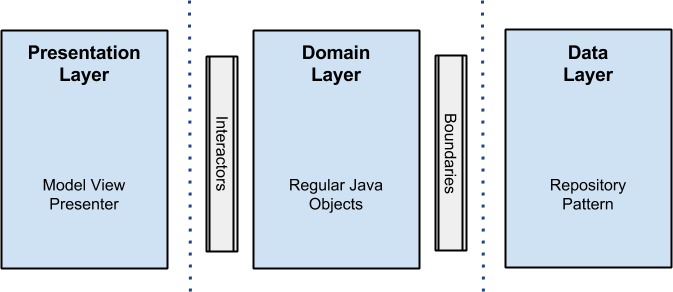


Рисунок 1 – Слоистая архитектура Андроид-приложения

Основная идея всех архитектурных паттернов сохраняется – мы разбиваем код на модули на несколько слоев, среди которых обязательно есть слой, содержащий логику приложения, и слой, отвечающий за представление данных и работу с UI, к этим двум основным слоям добавляется слой для работы с данными. Это позволяет упростить клиент-серверное взаимодействие и избавляет бизнес-логику приложения от необходимости работать с получением данных из различных источников.

Разумеется, приложение – это в первую очередь взаимодействие с пользователем. Поэтому нам нужен специальный слой, который будет отвечать за логику отображения данных на экране, за взаимодействие с пользователем и за другие процессы, связанные с UI. Этот слой не должен содержать логику приложения, не связанную с UI.

Именно этот слой привязывается к экранам и помогает организовать взаимодействие со слоем бизнес-логики и работу с данными. Этот слой может быть реализован с использованием любого предпочитаемого паттерна, к примеру, MVC, MVP, MVVM и других.

# 2 Базы данных NoSQL

Базы данных NoSQL (нереляционные базы данных) – специально созданы для определённых моделей данных и обладают гибкими схемами, что позволяет разрабатывать современные приложения. Базы данных NoSQL получили широкое распространение в связи с простотой разработки, функциональностью и производительностью при любых масштабах.

Как работает база данных NoSQL? Базы данных NoSQL используют разнообразные модели данных для доступа к данным и управления ими. Базы данных таких типов оптимизированы для приложений, которые работают с большим объёмом данных, нуждаются в низкой задержке и гибких моделях данных. Всё это достигается путём смягчения жёстких требований к непротиворечивости данных, характерных для других типов БД.

Для чего можно использовать базы данных NoSQL? Базы данных NoSQL хорошо подходят для многих современных приложений, например, мобильных, игровых, интернет-приложений, когда требуются гибкие масштабируемые базы данных с высокой производительностью и широкими функциональными возможностями, способные обеспечивать максимальное удобство использования.

Рассмотрим преимущества баз данных NoSQL.

* *Гибкость*. Как правило, базы данных NoSQL предлагают гибкие схемы, что позволяет осуществлять разработку быстрее и обеспечивает возможность поэтапной реализации. Благодаря использованию гибких моделей данных БД NoSQL хорошо подходят для частично структурированных и неструктурированных данных.
* *Масштабируемость.* Базы данных NoSQL рассчитаны на масштабирование с использованием распределённых кластеров аппаратного обеспечения, а не путём добавления дорогих надёжных серверов. Некоторые поставщики облачных услуг проводят эти операции в фоновом режиме, обеспечивая полностью управляемый сервис.
* *Высокая производительность.* Базы данных NoSQL оптимизированы для конкретных моделей данных и шаблонов доступа, что позволяет достичь более высокой производительности по сравнению с реляционными базами данных.
* *Широкие функциональные возможности.* Базы данных NoSQL предоставляют API и типы данных с широкой функциональностью которые специально разработаны для соответствующих моделей данных.

## **2.1 Firebase Realtime Database**

База данных Firebase Realtime – это облачная NoSQL база данных компании Google. Данные хранятся в формате JSON и синхронизируются в режиме реального времени с каждым подключенным клиентом. Поддержаны особенности интеграции с приложениями под операционные системы Android и iOS, реализовано API для приложений на JavaScript, Java, Objective-C и Node.js. Также возможно работать напрямую с базой данных в стиле REST из ряда JavaScript-фреймворков, включая AngularJS, React, Vue.js, Ember.js и Backbone.js. Предусмотрено API для шифрования данных. При создании кроссплатформенного приложения с помощью SDK для iOS, Android и JavaScript, все клиенты совместно используют один экземпляр базы данных в реальном времени и автоматически получают обновления с самыми новыми данными.

## **2.1.1 Ключевые особенности**

Работа в реальном времени. Вместо типичных HTTP-запросов база данных Firebase Realtime использует синхронизацию данных – каждый раз, когда данные меняются, любое подключенное устройство получает это обновление в течение миллисекунд, что позволяет обеспечивать совместный и захватывающий опыт, не думая о сетевом коде.

Оффлайн работа. Приложения Firebase остаются отзывчивыми даже в автономном режиме, поскольку SDK базы данных Firebase Realtime сохраняет ваши данные на диск. После восстановления соединения клиентское устройство получает любые пропущенные изменения, синхронизируя его с текущим состоянием сервера.

Доступность с устройств клиентов. Доступ к базе данных Firebase Realtime можно получить непосредственно с мобильного устройства или веб-браузера; нет необходимости в сервере приложений. Безопасность и проверка данных доступны через правила безопасности баз данных Firebase Realtime, основанные на выражениях правила, которые выполняются при чтении или записи данных.

Масштабирование по нескольким базам данных. С помощью базы данных Firebase Realtime можно масштабировать потребности приложений в данных, разбивая данные на несколько экземпляров базы данных в одном проекте Firebase. Есть возможность оптимизировать аутентификацию с помощью Firebase Authentication для вашего проекта и проверять подлинность пользователей во всех экземплярах базы данных. Управлять доступом к данным в каждой базе данных с помощью пользовательских правил базы данных Firebase Realtime для каждого экземпляра базы данных.

## **2.1.2 Применение**

База данных Firebase Realtime позволяет создавать многофункциональные приложения для совместной работы, обеспечивая безопасный доступ к базе данных непосредственно из кода на стороне клиента. Данные сохраняются локально, и даже в автономном режиме события в реальном времени продолжают срабатывать, предоставляя пользователю полноценный отклик. Когда устройства восстанавливает соединение, база данных реального времени синхронизирует локальные изменения данных с удалёнными обновлениями, которые произошли, когда клиент находился в автономном режиме, автоматически объединяя любые изменения.

База данных реального времени предоставляет гибкий язык правил на основе выражений, называемый правилами безопасности баз данных в реальном времени Firebase, для определения того, как ваши данные должны быть структурированы и когда данные могут считываться или записываться. При интеграции с аутентификацией Firebase разработчики могут определить, кто имеет доступ к каким данным и как они могут получить к ним доступ.

База данных реального времени является базой данных NoSQL, и поэтому имеет иные виды оптимизации и функциональность по сравнению с реляционной базой данных. API-интерфейс базы данных реального времени позволяет выполнять только те операции, которые могут быть выполнены быстро. Это позволяет вам производить обработку данных в реальном времени, обслуживая миллионы пользователей без ущерба для скорости отклика. В связи с этим важно продумать то, как пользователя должны получать доступ к вашим данным, и затем соответствующим образом структурировать их.

## **2.1.3 Структурирование и хранение данных**

Все данные базы данных Firebase Realtime хранятся в виде объектов JSON. Вы можете думать о базе данных как о дереве JSON, размещённом в облаке. В отличие от базы данных SQL, здесь нет таблиц или записей. Когда вы добавляете данные в дерево JSON, оно становится узлом в существующей структуре JSON со связанным ключом. Хотя база данных использует дерево JSON, данные, хранящиеся в базе данных, могут быть представлены в виде определённых собственных типов, которые соответствуют доступным типам JSON, что обеспечивает возможность написать более поддерживаемый код. Например, рассмотрим приложение чата, которое позволяет пользователям сохранять базовый профиль и список контактов. Профиль пользователя находится, например, в “/users/$uid”. У пользователя “alovelace” может быть запись в базе данных, которая выглядит, как показано ниже (рисунок 2).



Рисунок 2 – Запись в базе данных

База данных Firebase Realtime позволяет вкладывать данные глубиной до 32 уровней, однако не стоит думать, что так структура выглядит по умолчанию. Таким образом, когда вы выбираете данные в каком-либо месте в вашей базе данных, вы также получаете все его дочерние узлы. Кроме того, когда вы предоставляете кому-то доступ для чтения или записи в узле вашей базы данных, вы также предоставляете им доступ ко всем данным в этом узле. Поэтому на практике лучше сохранять структуру данных как можно более плоской. В качестве примера того, почему вложенные данные являются плохими, рассмотрим следующую структуру с множественными вложениями (рисунок 3):

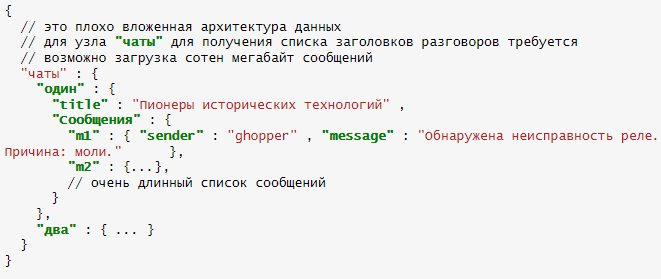


Рисунок 3 – Плохая структура данных

При таком вложенном дизайне итерация по данным становится проблематичной. Например, для перечисления заголовков разговоров в чате требуется, чтобы всё дерево “чаты”, включая всех участников и сообщения, было загружено на клиент.

## **2.1.4 Безопасность**

## **2.1.4.1 Правила безопасности**

Правила базы данных Firebase Realtime определяют, кто имеет права на чтение и запись в вашу базу данных, как структурированы ваши данные и какие индексы существуют. Эти правила располагаются на серверах Firebase и применяются автоматически в любое время. Каждый запрос на чтение и запись будет выполнен, только удовлетворяет правилам. По умолчанию правила не разрешают кому-либо доступ к вашей базе данных. Это необходимо для защиты базы данных от злоумышленников до тех пор, пока у вас не будет времени настроить свои правила или настроить аутентификацию

Правила базы данных реального времени имеют JavaScript-подобный синтаксис и бывают четырёх типов.

1. read: описывает, разрешено ли чтение данных пользователям.
2. write: описывает возможность и время записи данных.
3. validate: определяет, как будет выглядеть правильно отформатированное значение, имеет ли оно дочерние атрибуты, и тип данных.
4. indexOn: определяет дочерний элемент для индексации для поддержки упорядочения и запросов.

База данных Firebase Realtime предоставляет полный набор инструментов для управления безопасностью вашего приложения. Эти инструменты облегчают аутентификацию ваших пользователей, обеспечение прав пользователей и проверку входных данных.

Приложения на базе Firebase выполняют больше клиентского кода, чем приложения со многими другими технологиями.

## **2.1.4.2 Аутентификация**

Распространённым первым шагом в защите вашего приложения является идентификация ваших пользователей. Этот процесс называется аутентификацией. Вы можете использовать Firebase Authentication, чтобы пользователи могли войти в ваше приложение. Аутентификация в Firebase включает в себя поддержку обычных методов аутентификации, таких как Google и Facebook, а также вход по электронной почте и паролю, анонимный вход и многое другое.

Идентификация пользователя является важной концепцией безопасности. У разных пользователей разные данные, а иногда у них разные возможности. Например, в приложении чата каждое сообщение связано с пользователем, который его создал. Пользователи также могут удалять свои сообщения, но не сообщения других пользователей.

## **2.1.4.3 Авторизация**

Идентификация вашего пользователя – это только часть безопасности. Как только вы узнаете, кто они, вам понадобится способ контролировать их доступ к данным в вашей базе данных. Правила базы данных в реальном времени позволяют вам контролировать доступ для каждого пользователя. Например, вот набор правил безопасности, который позволяет любому читать путь “/foo/”, но никто не может писать в него (рисунок 4):



Рисунок 4 – Набор правил безопасности

Правила базы данных реального времени включают в себя встроенные переменные и функции, которые позволяют ссылаться на другие пути, временные метки на стороне сервера, информацию об аутентификации и многое другое. Вот пример правила, которое предоставляет доступ на запись для аутентифицированных пользователей “/users/<uid>/”, где “<uid>” – это идентификатор пользователя, полученный с помощью Firebase Authentication (рисунок 5):



Рисунок 5 – Пример правила доступа на запись

## **2.1.4.4 Валидация данных**

База данных Firebase Realtime является нереляционной. Это позволяет легко что-то менять по мере разработки, но как только ваше приложение будет готово выходу в массы, важно, чтобы данные оставались согласованными. Язык правил включает в себя “.validate” правило, которое позволяет применять логику проверки с использованием тех же выражений, что “.read” и для “.write” правил. Единственное отличие состоит в том, что правила проверки не каскадируются, поэтому все соответствующие правила проверки должны иметь значение “true” для разрешения записи.

Эти правила обеспечивают, что записываемые данные “/foo/” должны быть строкой длиной менее 100 символов (рисунок 6):

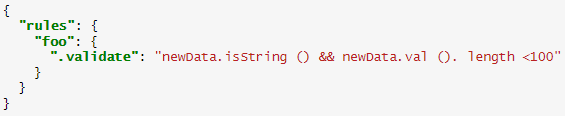


Рисунок 6 – Пример правила “данные должны быть строкой длиной менее 100 символов”

Правила проверки имеют доступ ко всем тем же встроенным функциям и переменным, что “.read” и для “.write” правила. Их можно использовать для создания правил проверки, которые будут знать данные в других местах вашей базы данных, личность вашего пользователя, время сервера и многое другое.

## **2.1.4.5 Определение индексов базы данных**

База данных Firebase Realtime позволяет упорядочивать и запрашивать данные. Для небольших размеров данных база данных поддерживает специальные запросы, поэтому индексы, как правило, не требуются во время разработки. Однако перед запуском приложения важно указать индексы для любых запросов, которые необходимо выполнить, чтобы они продолжали работать по мере роста вашего приложения.

Индексы указываются с использованием “.indexOn” правила. Вот пример объявления индекса, который будет индексировать поля высоты и длины для списка динозавров (рисунок 7):



Рисунок 7 – Пример объявления индекса

## **2.2 Логика построения БД чата в Firebase**

В базе данных есть 5 узлов (далее ключ = id).

* Узел node\_chat, вложенность 0. Отвечает за хранение списка чатов. Узлы внутри (0🡪id\_chat, вложенность 1) содержат объекты Chat, доступные по ключу – id чата (id\_chat).
* Узел node\_user, вложенность 0. Отвечает за хранение списка пользователей. Узлы внутри (0🡪id\_user, вложенность 1) содержат объекты User, доступные по ключу – id пользователя (id\_user).
* Узел node\_message, вложенность 0. Отвечает за хранение списка сообщений в каждом чате. Узлы внутри (0🡪id\_chat, вложенность 1) имеют ключ – id чата (id\_chat) и хранят списки сообщений. Узлы внутри (0🡪1🡪id\_message, вложенность 2) содержат объекты Message, доступные по ключу – id сообщений (id\_message).
* Узел node\_chat\_in\_user, вложенность 0. Отвечает за хранение списка id чатов для каждого пользователя. Узлы внутри (0🡪id\_user, вложенность 1) имеют ключ – id пользователя (id\_user) и хранят списки id чатов. Узлы внутри (0🡪1🡪id\_chat\_in\_user, вложенность 2) содержат объекты ChatInUser, доступные по ключу id\_chat\_in\_user и хранящие внутри себя id отдельного чата.
* Узел node\_user\_in\_chat, вложенность 0. Отвечает за хранение списка id пользователей для каждого чата. Узлы внутри (0🡪id\_chat, вложенность 1) имеют ключ – id чата (id\_chat) и хранят списки id пользователей. Узлы внутри (0🡪1🡪id\_user\_in\_chat, вложенность 2) содержат объекты UserInChat, доступные по ключу id\_user\_in\_chat и хранящие внутри себя id отдельного пользователя.

Модели, на основе которых строятся объекты, хранящиеся в базе данных, указаны ниже.

* Chat, имеет следующие поля: id\_chat, name.
* User, имеет следующие поля: id\_user, first\_name, last\_name, about\_user.
* Message, имеет следующие поля: id\_message, id\_chat, id\_user, text.
* ChatInUser, имеет следующие поля: id\_chat\_in\_user, id\_user, id\_chat.
* UserInChat, имеет следующие поля: id\_user\_in\_chat, id\_user, id\_chat.

Общий вид структуры базы данных (рисунок 8):

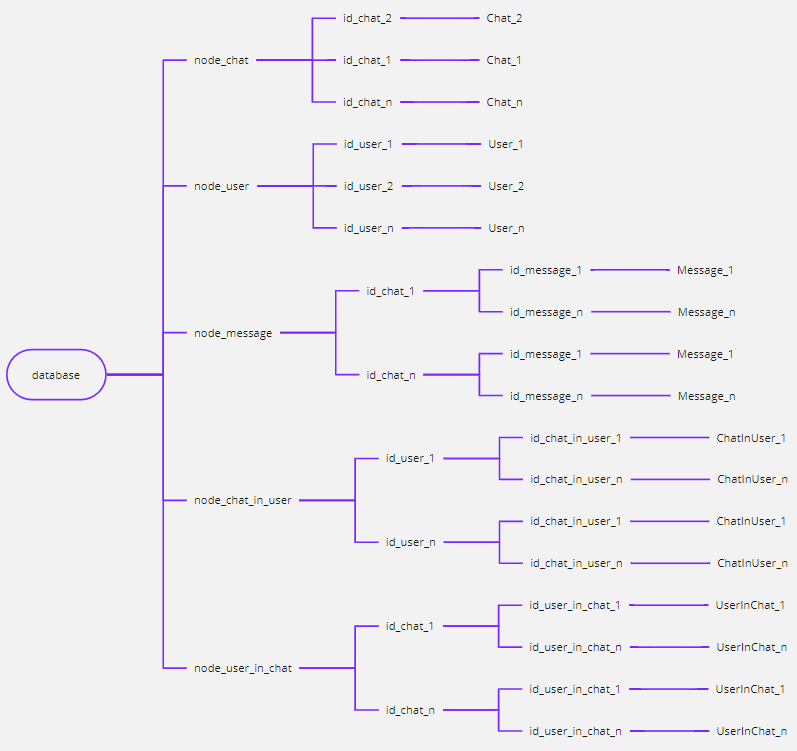


Рисунок 8 – Структура базы данных

## **2.3 Консоль Firebase**

Зарегистрированных пользователей можно посмотреть в консоли (рисунок 9).

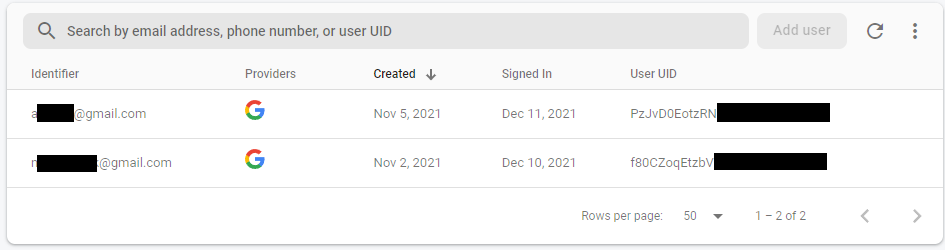


Рисунок 9 – Пользователи приложения

База данных, на основе которой построено приложение, показана на рисунке 10.



Рисунок 10 – База данных приложения

Регистрация в приложении осуществляется с помощью личного Google-аккаунта; в консоли Firebase этот метод указан в качестве провайдера (рисунок 11):

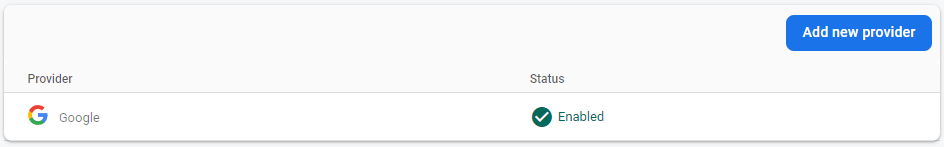


Рисунок 11 – Провайдер для регистрации в приложении

Также можно добавить и другие методы регистрации/авторизации, например по email.

Правила, установленные для базы данных показаны на рисунке 12.

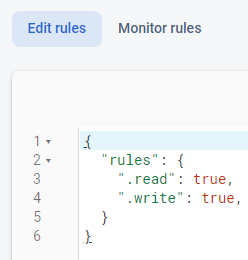


Рисунок 12 – Правила доступа к базе данных

Правила можно менять вручную. Здесь установлены права на чтение и запись без ограничений. Их можно закрыть полностью (false), открыть до определённого момента (now < “milliseconds”) или регулировать другими условиями.

# 3 Архитектура приложения Eco

Общая схема взаимодействия элементов приложения выглядит следующим образом (рисунок 13):

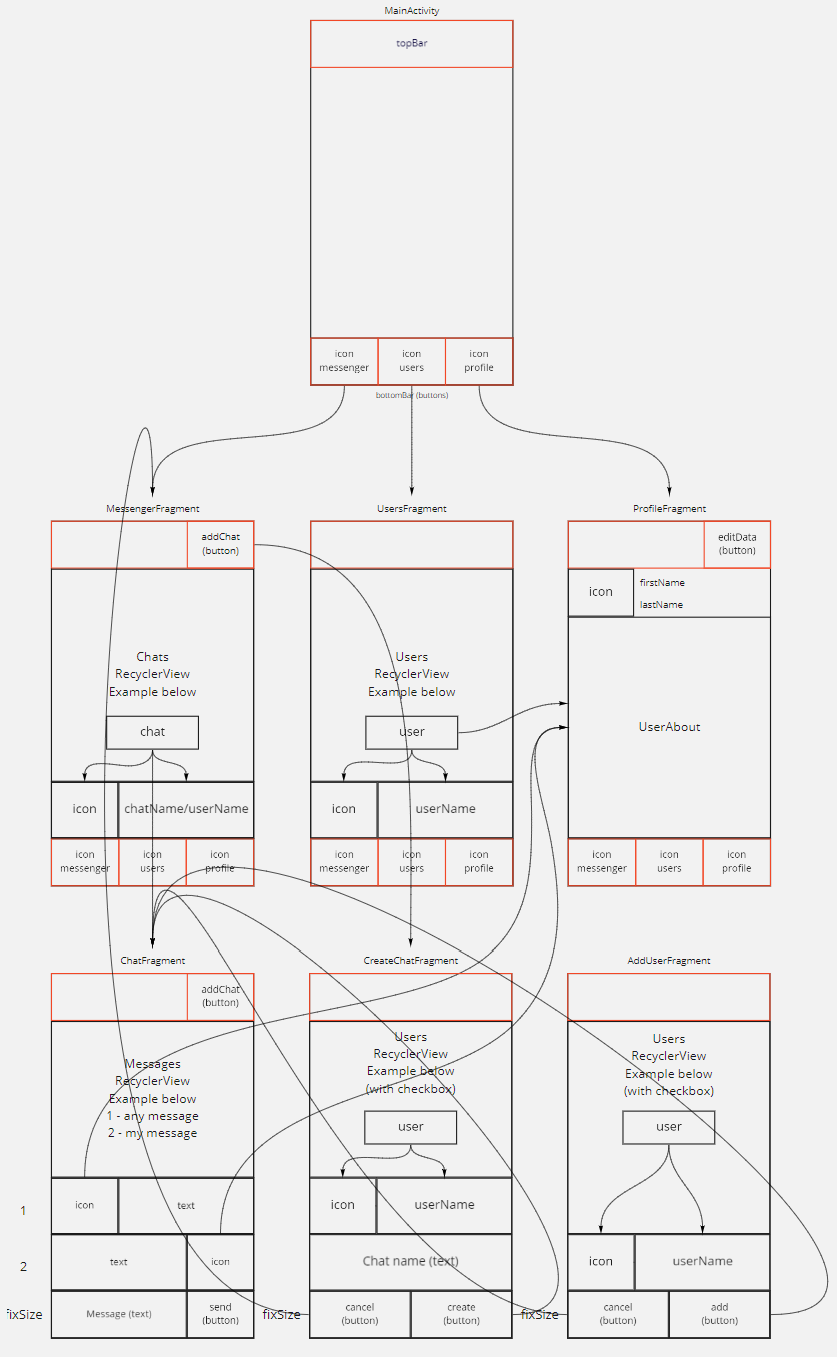


Рисунок 13 – Проект UI

Рассмотрим, что из себя представляет каждое из окон.

MainActivity – главное окно (активность) программы, которое содержит в себе фрагмент, с которым пользователь взаимодействует на текущий момент. Фрагменты меняются в зависимости от ситуации, но активность неизменна и работает на протяжение жизни всего приложения.

MessengerFragment – фрагмент, который отображает список чатов текущего пользователя, а также содержит кнопку “создать чат”.

UsersFragment – фрагмент, который отображает список всех пользователей, которые зарегистрированы в приложении.

ProfileFragment – фрагмент, который отображает информацию о конкретном пользователе, содержит кнопку редактирования данных и кнопку выхода из аккаунта (завершения сессии).

ChatFragment – фрагмент, который отображает список сообщений в конкретном чате, содержит кнопку “добавить пользователей”, текстовое поле с текстом сообщения и кнопку для отправки сообщения.

CreateChatFragment – фрагмент, который отображает список пользователей для добавления в чат, содержит текстовое поле с текстом названия чата, кнопку возврата в MessengerFragment и кнопку “создать чат”.

AddUserFragment – фрагмент, который отображает список доступных для добавления в чат пользователей, содержит кнопку возврата в ChatFragment и кнопку “добавить пользователей”.

## **3.1 MessengerFragment**

Фрагмент отображает список карточек чатов. Отдельно карточка чата представляет собой лишь название чата. При нажатии на карточку конкретного чата произойдёт переход в ChatFragment, где отображаются данные этого чата. При нажатии на кнопку “создать чат” произойдёт переход в CreateChatFragment.

## **3.2 UsersFragment**

Фрагмент отображает список карточек всех пользователей, которые зарегистрированы в приложении. Отдельно карточка пользователя представляет собой имя пользователя. Предполагается, что при нажатии на карточке произойдёт переход в ProfileFragment, чтобы можно было увидеть данные о пользователе, но сейчас эта функция не работает.

## **3.3 ProfileFragment**

Фрагмент, который должен отображать информацию о пользователе и поддерживать функцию редактирования данных пользователя. Также сверху можно увидеть кнопку выхода из аккаунта.

## **3.4 ChatFragment**

Фрагмент отображает список карточек сообщений. Карточка сообщения представляет собой текст сообщения и имя пользователя под ним. Предполагается, что при нажатии на карточку произойдёт переход в ProfileFragment, чтобы можно было увидеть данные о пользователе, который отправил сообщение, но сейчас эта функция не работает. При нажатии на кнопку “добавить пользователей” произойдёт переход в AddUserFragment. Во фрагменте есть поле для ввода сообщения. При нажатии на кнопку “отправить сообщение” сообщение будет отправлено и добавлено в БД.

## **3.5 CreateChatFragment**

Фрагмент отображает список карточек пользователей. Карточка пользователя представляет собой имя пользователя с checkbox’ом для выбора. Во фрагменте есть поле для ввода названия чата. При нажатии на кнопку “назад” произойдёт переход в MessengerFragment. При нажатии на кнопку “создать чат” произойдёт переход в ChatFragment, будет создан чат, добавлены участники и информация об этом будет добавлена в БД.

## **3.6 AddUserFragment**

Фрагмент отображает список карточек пользователей. Карточка пользователя представляет собой имя пользователя с checkbox’ом для выбора. При нажатии на кнопку “назад” произойдёт переход в ChatFragment. При нажатии на кнопку “добавить пользователей” произойдёт переход в ChatFragment, в чате появятся новые участники и информация об этом будет добавлена в БД.

# 4 Разработка приложения

## **4.1 Диаграмма классов**

Приложение построено на основе следующих классов.

* MainActivity – главное окно программы, которое содержит в себе
* фрагменты и реализует интерфейс IBottomNavViewCallback.
* IBottomNavViewCallback – интерфейс, который служит для реализации нижних кнопок навигации в приложении: Messenger, Users, Profile.
* LoginFragment – фрагмент, который отвечает за регистрацию пользователя.
* MessengerFragment – фрагмент, который отвечает за отображение списка чатов.
* UsersFragment – фрагмент, который отвечает за отображение списка пользователей.
* ProfileFragment – фрагмент, который отвечает за отображение информации о пользователе.
* ChatFragment – фрагмент, который отвечает за отображение списка сообщений в чате.
* CreateChatFragment – фрагмент, который отвечает за создание чата и добавление участников в чат.
* AddUserFragment – фрагмент, который отвечает за добавление участников в чат.
* MessengerAdapter – адаптер, который относится к RecyclerView в MessengerFragment. Отвечает за отображение данных каждого элемента списка и за события, которые реализованы для этих элементов (например, нажатие).
* UsersAdapter – адаптер, который относится к RecyclerView в UsersFragment.
* ChatAdapter – адаптер, который относится к RecyclerView в ChatFragment.
* CreateChatAdapter – адаптер, который относится к RecyclerView в CreateChatFragment.
* AddUserAdapter – адаптер, который относится к RecyclerView в AddUserFragment.
* Chat – модель данных, на основе которой может быть сформирован объект “чат” в базе данных.
* User – модель данных, на основе которой может быть сформирован объект “пользователь” в базе данных.
* Message – модель данных, на основе которой может быть сформирован объект “сообщение” в базе данных.
* ChatInUser – модель данных, на основе которой может быть сформирован объект “чат, который доступен пользователю” в базе данных.
* UserInChat – модель данных, на основе которой может быть сформирован объект “пользователь, который находится в чате” в базе данных.
* Database – класс, предназначенный для работы с базой данных. Класс хранит внутри списки и словари объектов Chat, User, Message, ChatInUser, UserInChat, объект пользователя, private-объект базы данных Firebase, плавающее (меняется при переходе в конкретный чат) поле chatId, адаптеры для RecyclerView фрагментов. Также класс хранит private-поля со ссылками на главные узлы (node\_<model>), private-методы (key<Model>) для формирования ключей (id) в конкретных узлах базы данных, функции-слушатели (listener<Model>) разных узлов базы данных в зависимости от того, для какого фрагмента они применяются, функции (setValue<Model>) для установки значений в базу данных. Для всего приложения создаётся всего 1 экземпляр Database, поэтому он реализует паттерн Singleton.

## **4.2 Интерфейс приложения**

|  |  |
| --- | --- |
| C:\Users\Владислав\Desktop\чаты.png | профиль |
| Рисунок 14 – Фрагмент Messenger | Рисунок 15 – Фрагмент Profile |

На рисунке 14 показан фрагмент, отображающий список имеющихся чатов. На рисунке 15 – профиль пользователя.

|  |  |
| --- | --- |
| создать чат | сообщения |
| Рисунок 16 – Фрагмент CreateChat | Рисунок 17 – Фрагмент Chat |

На рисунке 16 показан фрагмент, служащий для добавления чата, где можно увидеть пользователя, которого можно добавить в чат (в общем случае – список пользователей). На рисунке 17 – список сообщений в конкретном чате.

# ЗАКЛЮЧЕНИЕ

Работа является актуальной, поскольку на сегодняшний момент широко используются приложения для Android разной направленности, от приложений для государственных служб до развлекательных игр.

В данной работе были поставлены и решены следующие задачи:

* работа с консолью Google Firebase и интеграция облачной базы данных Firebase в проект приложения;
* спроектирована структура базы данных и реализован доступ к данным в Firebase на основе узлов;
* решено, как построить модель представления, модель данных, и как связать обе модели с помощью модели бизнес-логики в IDE Android Studio;
* разработан проект UI приложения, в т.ч. и как должен быть реализован переход между фрагментами, и само приложение на основе проекта.

В результате проделанной работы разработано мобильное приложение “Eco”, в котором пользователи могут общаться между собой в чатах, создавать новые и добавлять участников в текущие чаты. Регистрация осуществляется с помощью аккаунта Google. Все данные хранятся в облачной базе данных Firebase.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лекция 4 по архитектуре андроид приложения. Clean Architecture – Текст : электронный – 2017. – URL: https://www.fandroid.info/lektsiya-4-po-arhitekture-android-prilozheniya-clean-arcitecture/ (дата обращения: 12.12.2021г).
2. Что такое NoSQL? – Текст : электронный – 2018. – URL: https://aws.amazon.com/ru/nosql/ (дата обращения: 11.12.2021г).
3. Firebase Realtime Database – Текст : электронный – 2020. – URL: https://ru.bmstu.wiki/Firebase\_Realtime\_Database (дата обращения 11.12.2021г).

# ПРИЛОЖЕНИЕ А Главная активность приложения

Файл MainActivity.kt

package com.darf.example  
  
import androidx.appcompat.app.AppCompatActivity  
import android.os.Bundle  
import androidx.appcompat.app.AppCompatDelegate  
import androidx.navigation.NavController  
import androidx.navigation.Navigation  
import com.google.android.material.bottomnavigation.BottomNavigationView  
  
class MainActivity : AppCompatActivity(), IBottomNavViewCallback {  
  
 private lateinit var navController: NavController  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
 AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.*MODE\_NIGHT\_NO*)  
  
 navController = Navigation.findNavController(this, R.id.*main\_container*)  
 navController.navigate(R.id.*loginFragment*)  
 }  
  
 override fun runBottomNavigationView(bottomNavigationView: BottomNavigationView) {  
 bottomNavigationView.setOnItemSelectedListener **{** item **->** val fragment = when (item.*itemId*) {  
 R.id.*nav\_messenger* -> R.id.*messengerFragment* R.id.*nav\_users* -> R.id.*usersFragment* R.id.*nav\_profile* -> R.id.*profileFragment* else -> null  
 }  
  
 navController.navigate(fragment ?: R.id.*loginFragment*)  
  
 true  
 **}** }  
}

Файл IBottomNavViewCallback.kt

package com.darf.example  
  
import com.google.android.material.bottomnavigation.BottomNavigationView  
  
interface IBottomNavViewCallback {  
 fun runBottomNavigationView(bottomNavigationView: BottomNavigationView)  
}

# ПРИЛОЖЕНИЕ Б Фрагменты

Файл LoginFragment.kt

package com.darf.example.fragments  
  
import android.content.Intent  
import androidx.fragment.app.Fragment  
import android.os.Bundle  
import android.util.Log  
import android.view.\*  
import androidx.activity.result.ActivityResultLauncher  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.appcompat.app.AppCompatActivity  
import androidx.navigation.NavController  
import androidx.navigation.fragment.findNavController  
import com.darf.example.Database  
import com.darf.example.R  
import com.darf.example.databinding.FragmentLoginBinding  
import com.darf.example.models.User  
import com.google.android.gms.auth.api.signin.GoogleSignIn  
import com.google.android.gms.auth.api.signin.GoogleSignInClient  
import com.google.android.gms.auth.api.signin.GoogleSignInOptions  
import com.google.android.gms.common.api.ApiException  
import com.google.firebase.auth.FirebaseAuth  
import com.google.firebase.auth.GoogleAuthProvider  
import com.google.firebase.auth.ktx.auth  
import com.google.firebase.database.DataSnapshot  
import com.google.firebase.database.DatabaseError  
import com.google.firebase.database.ValueEventListener  
import com.google.firebase.database.ktx.database  
import com.google.firebase.ktx.Firebase  
  
class LoginFragment : Fragment() {  
  
 private lateinit var launcher: ActivityResultLauncher<Intent>  
 private lateinit var binding: FragmentLoginBinding  
 private lateinit var auth: FirebaseAuth  
 private lateinit var navController: NavController  
  
 private val database = Firebase.*database* private val dbRefNodeUser = database.getReference("node\_user")  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 (*activity* as AppCompatActivity).*title* = "Authentication"  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
  
 return inflater.inflate(R.layout.*fragment\_login*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 binding = FragmentLoginBinding.bind(view)  
  
 navController = *findNavController*()  
 auth = Firebase.*auth* launcher = registerForActivityResult(ActivityResultContracts.StartActivityForResult()) **{** val task = GoogleSignIn.getSignedInAccountFromIntent(**it**.*data*)  
 try {  
 val account = task.getResult(ApiException::class.*java*)  
 if (account != null)  
 firebaseAuthWithGoogle(account.*idToken*!!)  
 } catch (e: ApiException) {  
 Log.d("LoginFragment", "Api exception")  
 }  
 **}** binding.loginButton.setOnClickListener **{** signInWithGoogle()  
 **}** checkAuthState()  
 }  
  
 private fun getClient(): GoogleSignInClient {  
 val gso = GoogleSignInOptions  
 .Builder(GoogleSignInOptions.*DEFAULT\_SIGN\_IN*)  
 .requestIdToken(getString(R.string.default\_web\_client\_id))  
 .requestEmail()  
 .build()  
  
 return GoogleSignIn.getClient(requireActivity(), gso)  
 }  
  
 private fun signInWithGoogle() {  
 val signInClient = getClient()  
 launcher.launch(signInClient.*signInIntent*)  
 }  
  
 private fun firebaseAuthWithGoogle(idToken: String) {  
 val credential = GoogleAuthProvider.getCredential(idToken, null)  
 auth.signInWithCredential(credential).addOnCompleteListener **{** if (**it**.*isSuccessful*) {  
 Log.d("LoginFragment", "Google signIn done")  
 checkAuthState()  
 } else  
 Log.d("LoginFragment", "Google signIn error")  
 **}** }  
  
 private fun checkAuthState() {  
 val currentUser = auth.*currentUser* if (currentUser != null) {  
 var userExistInUserNode = false  
 dbRefNodeUser.addListenerForSingleValueEvent(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 snapshot.*children*.*forEach* **{** snap **->** if (currentUser.*uid* == snap.*key*)  
 userExistInUserNode = true  
 **}** }  
  
 override fun onCancelled(error: DatabaseError) {  
  
 }  
 })  
  
 val db = Database.instance()  
 if (!userExistInUserNode)  
 db.setValueUser("")  
 navController.navigate(R.id.*messengerFragment*)  
 }  
 }  
}

Файл MessengerFragment.kt

package com.darf.example.fragments  
  
import android.os.Bundle  
import android.view.\*  
import androidx.appcompat.app.AppCompatActivity  
import androidx.fragment.app.Fragment  
import androidx.navigation.NavController  
import androidx.navigation.fragment.findNavController  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.darf.example.Database  
import com.darf.example.IBottomNavViewCallback  
import com.darf.example.R  
import com.darf.example.adapters.MessengerAdapter  
import com.darf.example.databinding.FragmentMessengerBinding  
import com.darf.example.models.ChatInUser  
import com.google.firebase.auth.FirebaseAuth  
import com.google.firebase.auth.ktx.auth  
import com.google.firebase.database.DataSnapshot  
import com.google.firebase.database.DatabaseError  
import com.google.firebase.database.ValueEventListener  
import com.google.firebase.database.ktx.database  
import com.google.firebase.ktx.Firebase  
import java.lang.Thread.sleep  
  
class MessengerFragment : Fragment() {  
 private lateinit var navController: NavController  
 private lateinit var binding: FragmentMessengerBinding  
 private lateinit var curActivity: AppCompatActivity  
  
 private val db = Database.get()  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setHasOptionsMenu(true)  
  
 navController = *findNavController*()  
 db.adapterMessenger = MessengerAdapter(db.listChatInUser, navController)  
  
 onDataChange()  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?,  
 ): View? {  
 curActivity = *activity* as AppCompatActivity  
 curActivity.*title* = "Messenger"  
 return inflater.inflate(R.layout.*fragment\_messenger*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 binding = FragmentMessengerBinding.bind(view)  
  
 binding.*apply* **{** messengerRecyclerView.*layoutManager* = LinearLayoutManager(*activity*)  
 messengerRecyclerView.*adapter* = db.adapterMessenger  
 **}** }  
  
 override fun onStart() {  
 super.onStart()  
  
 val bottomNavigationView = binding.include.bottomNavigation  
 bottomNavigationView.*selectedItemId* = R.id.*nav\_messenger* (*activity* as IBottomNavViewCallback).runBottomNavigationView(bottomNavigationView)  
 }  
  
 override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {  
 inflater.inflate(R.menu.*top\_messenger\_menu*, menu)  
 }  
  
 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
 if (item.*itemId* == R.id.*messenger\_add*) {  
// db.setValueChat("Chat name #${(1..100).shuffled().last()}")  
// db.setValueChatInUser()  
// db.setValueUserInChat()  
 navController.navigate(R.id.*createChatFragment*)  
 }  
  
 return super.onOptionsItemSelected(item)  
 }  
  
 private fun onDataChange() {  
 db.listenerChat()  
 db.listenerUser()  
// db.listenerUserInChat()  
 db.listenerChatInUserChild()  
 }  
}

Файл UsersFragment.kt

package com.darf.example.fragments  
  
import android.os.Bundle  
import androidx.fragment.app.Fragment  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import androidx.appcompat.app.AppCompatActivity  
import androidx.navigation.NavController  
import androidx.navigation.fragment.findNavController  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.darf.example.Database  
import com.darf.example.IBottomNavViewCallback  
import com.darf.example.R  
import com.darf.example.adapters.UsersAdapter  
import com.darf.example.databinding.FragmentUsersBinding  
import com.darf.example.models.User  
import com.google.firebase.database.DataSnapshot  
import com.google.firebase.database.DatabaseError  
import com.google.firebase.database.ValueEventListener  
import com.google.firebase.database.ktx.database  
import com.google.firebase.ktx.Firebase  
  
class UsersFragment : Fragment() {  
  
 lateinit var navController: NavController  
 lateinit var binding: FragmentUsersBinding  
 lateinit var curActivity: AppCompatActivity  
  
 private val db = Database.get()  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 navController = *findNavController*()  
  
 onDataChange()  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 curActivity = *activity* as AppCompatActivity  
 curActivity.*title* = "Users"  
 return inflater.inflate(R.layout.*fragment\_users*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 binding = FragmentUsersBinding.bind(view)  
  
 binding.*apply* **{** usersRecyclerView.*layoutManager* = LinearLayoutManager(*activity*)  
 usersRecyclerView.*adapter* = db.adapterUsers  
 **}** }  
  
 override fun onStart() {  
 super.onStart()  
  
 val bottomNavigationView = binding.include.bottomNavigation  
 bottomNavigationView.*selectedItemId* = R.id.*nav\_users* (*activity* as IBottomNavViewCallback).runBottomNavigationView(bottomNavigationView)  
 }  
  
 private fun onDataChange() {  
 db.listenerUser()  
 }  
}

Файл ProfileFragment.kt

package com.darf.example.fragments  
  
import android.os.Bundle  
import android.view.\*  
import androidx.appcompat.app.AppCompatActivity  
import androidx.fragment.app.Fragment  
import androidx.fragment.app.FragmentManager  
import androidx.navigation.NavController  
import androidx.navigation.fragment.findNavController  
import com.darf.example.Database  
import com.darf.example.IBottomNavViewCallback  
import com.darf.example.R  
import com.darf.example.databinding.FragmentProfileBinding  
import com.google.firebase.auth.FirebaseAuth  
import com.google.firebase.auth.ktx.auth  
import com.google.firebase.ktx.Firebase  
  
class ProfileFragment : Fragment() {  
  
 lateinit var navController: NavController  
 lateinit var binding: FragmentProfileBinding  
 lateinit var curActivity: AppCompatActivity  
  
 private val db = Database.get()  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setHasOptionsMenu(true)  
  
 navController = *findNavController*()  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 curActivity = *activity* as AppCompatActivity  
 curActivity.*title* = db.user.*displayName* return inflater.inflate(R.layout.*fragment\_profile*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 binding = FragmentProfileBinding.bind(view)  
 }  
  
 override fun onStart() {  
 super.onStart()  
  
 val bottomNavigationView = binding.include.bottomNavigation  
 bottomNavigationView.*selectedItemId* = R.id.*nav\_profile* (*activity* as IBottomNavViewCallback).runBottomNavigationView(bottomNavigationView)  
 }  
  
 override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {  
 inflater.inflate(R.menu.*top\_profile\_menu*, menu)  
 }  
  
 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
 if (item.*itemId* == R.id.*profile\_sign\_out*) {  
 db.auth.signOut()  
// navController.backStack.clear()  
 navController.navigate(R.id.*action\_profileFragment\_to\_loginFragment*)  
 }  
  
 return super.onOptionsItemSelected(item)  
 }  
}

Файл ChatFragment.kt

package com.darf.example.fragments  
  
import android.os.Bundle  
import android.util.Log  
import android.view.\*  
import androidx.appcompat.app.AppCompatActivity  
import androidx.fragment.app.Fragment  
import androidx.navigation.NavController  
import androidx.navigation.fragment.findNavController  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.darf.example.Database  
import com.darf.example.R  
import com.darf.example.adapters.ChatAdapter  
import com.darf.example.databinding.FragmentChatBinding  
import com.darf.example.models.Chat  
import com.darf.example.models.ChatInUser  
import com.darf.example.models.Message  
import com.google.firebase.auth.FirebaseAuth  
import com.google.firebase.auth.ktx.auth  
import com.google.firebase.database.DataSnapshot  
import com.google.firebase.database.DatabaseError  
import com.google.firebase.database.ValueEventListener  
import com.google.firebase.database.ktx.database  
import com.google.firebase.ktx.Firebase  
import java.lang.Thread.sleep  
  
const val *KEY\_GET\_CHAT\_ID* = "get\_chat\_id"  
const val *KEY\_GET\_CHAT\_NAME* = "get\_chat\_name"  
  
class ChatFragment : Fragment() {  
  
 private lateinit var navController: NavController  
 private lateinit var binding: FragmentChatBinding  
 private lateinit var curActivity: AppCompatActivity  
  
 private val db = Database.get()  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setHasOptionsMenu(true)  
  
 navController = *findNavController*()  
  
 onDataChange()  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?,  
 ): View? {  
 curActivity = *activity* as AppCompatActivity  
 curActivity.*title* = db.listChat[db.chatId]?.chatName  
 return inflater.inflate(R.layout.*fragment\_chat*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 binding = FragmentChatBinding.bind(view)  
 Log.d("ChatFragment", "Запускается еще раз")  
  
 binding.*apply* **{** db.adaptersChat[db.chatId] =  
 db.adaptersChat[db.chatId] ?: ChatAdapter(*listOf*()) // костыль  
 chatRecyclerView.*layoutManager* = LinearLayoutManager(*activity*)  
 chatRecyclerView.*adapter* = db.adaptersChat[db.chatId]  
  
 chatButtonSend.setOnClickListener **{** val message = chatMessage.*text*.toString()  
 chatMessage.*text*.clear()  
 db.setValueMessage(message)  
 **}  
 }** }  
  
 override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {  
 inflater.inflate(R.menu.*top\_chat\_menu*, menu)  
 }  
  
 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
  
 if (item.*itemId* == R.id.*chat\_add*)  
 navController.navigate(R.id.*addUserFragment*)  
  
 return super.onOptionsItemSelected(item)  
 }  
  
 private fun onDataChange() {  
 db.listenerMessageChild()  
 db.listenerUserInChatChild()  
 }  
}

Файл CreateCharFragment.kt

package com.darf.example.fragments  
  
import android.os.Bundle  
import android.util.Log  
import androidx.fragment.app.Fragment  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.view.WindowId  
import androidx.appcompat.app.AppCompatActivity  
import androidx.navigation.NavController  
import androidx.navigation.fragment.findNavController  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.darf.example.Database  
import com.darf.example.R  
import com.darf.example.databinding.FragmentCreateChatBinding  
  
class CreateChatFragment : Fragment() {  
  
 private lateinit var navController: NavController  
 private lateinit var binding: FragmentCreateChatBinding  
 private lateinit var curActivity: AppCompatActivity  
  
 private val db = Database.get()  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 navController = *findNavController*()  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?,  
 ): View? {  
 curActivity = *activity* as AppCompatActivity  
 curActivity.*title* = "Create Chat"  
 return inflater.inflate(R.layout.*fragment\_create\_chat*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 binding = FragmentCreateChatBinding.bind(view)  
  
 binding.*apply* **{** createChatNoUsersAvailable.*visibility* =  
 if (db.adapterCreateChat.*itemCount* != 0) View.*GONE* else View.*VISIBLE* createChatRecyclerView.*layoutManager* = LinearLayoutManager(*activity*)  
 createChatRecyclerView.*adapter* = db.adapterCreateChat  
  
 createChatButtonCreate.setOnClickListener **{** val chatName = createChatChatName.*text*.toString()  
 db.setValueChat(chatName = chatName)  
 // для текущего user  
 db.setValueUserInChat()  
 db.setValueChatInUser()  
 // для отмеченных user  
 db.adapterCreateChat.getCheckedUserList().*forEach* **{** user **->** db.setValueUserInChat(userId = user.userId!!)  
 db.setValueChatInUser(userId = user.userId)  
 **}** navController.popBackStack()  
 navController.navigate(R.id.*chatFragment*)  
 **}** createChatButtonCancel.setOnClickListener **{** navController.popBackStack()  
 **}  
 }** }  
}

Файл AddUserFragment.kt

package com.darf.example.fragments  
  
import android.os.Bundle  
import androidx.fragment.app.Fragment  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import androidx.appcompat.app.AppCompatActivity  
import androidx.navigation.NavController  
import androidx.navigation.fragment.findNavController  
import androidx.recyclerview.widget.LinearLayoutManager  
import com.darf.example.Database  
import com.darf.example.R  
import com.darf.example.adapters.AddUserAdapter  
import com.darf.example.adapters.ChatAdapter  
import com.darf.example.databinding.FragmentAddUserBinding  
import com.darf.example.databinding.FragmentChatBinding  
import com.darf.example.models.User  
import com.darf.example.models.UserInChat  
import com.google.firebase.auth.FirebaseAuth  
import com.google.firebase.auth.ktx.auth  
import com.google.firebase.database.DataSnapshot  
import com.google.firebase.database.DatabaseError  
import com.google.firebase.database.ValueEventListener  
import com.google.firebase.database.ktx.database  
import com.google.firebase.ktx.Firebase  
  
class AddUserFragment : Fragment() {  
  
 lateinit var navController: NavController  
 lateinit var binding: FragmentAddUserBinding  
 lateinit var curActivity: AppCompatActivity  
  
 private val db = Database.get()  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 navController = *findNavController*()  
  
 onDataChange()  
 }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?,  
 ): View? {  
 curActivity = *activity* as AppCompatActivity  
 curActivity.*title* = "Add User"  
 return inflater.inflate(R.layout.*fragment\_add\_user*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 binding = FragmentAddUserBinding.bind(view)  
  
 binding.*apply* **{** addUserNoUsersAvailable.*visibility* =  
 if (db.adapterAddUser.*itemCount* != 0) View.*GONE* else View.*VISIBLE* addUserRecyclerView.*layoutManager* = LinearLayoutManager(*activity*)  
 addUserRecyclerView.*adapter* = db.adapterAddUser  
  
 addUserButtonAdd.setOnClickListener **{** db.adapterAddUser.getCheckedUserList().*forEach* **{** user **->** db.setValueUserInChat(user.userId!!)  
 db.setValueChatInUser(user.userId)  
 **}** navController.popBackStack()  
 **}** addUserButtonCancel.setOnClickListener **{** navController.popBackStack()  
 **}  
 }** }  
  
 private fun onDataChange() {  
// db.listenerUser()  
// db.listenerUserInChatChild()  
 }  
}

# ПРИЛОЖЕНИЕ В Адаптеры

Файл MessengerAdapter.kt

package com.darf.example.adapters  
  
import android.annotation.SuppressLint  
import android.os.Bundle  
import android.util.Log  
import android.view.LayoutInflater  
import android.view.ViewGroup  
import androidx.navigation.NavController  
import androidx.recyclerview.widget.RecyclerView  
import com.darf.example.Database  
import com.darf.example.R  
import com.darf.example.databinding.MessengerRecyclerviewItemBinding  
import com.darf.example.fragments.*KEY\_GET\_CHAT\_ID*import com.darf.example.fragments.*KEY\_GET\_CHAT\_NAME*import com.darf.example.models.Chat  
import com.darf.example.models.ChatInUser  
import com.darf.example.models.User  
import com.google.firebase.database.\*  
import com.google.firebase.database.ktx.database  
import com.google.firebase.ktx.Firebase  
  
class MessengerAdapter(  
 private var chatList: List<ChatInUser>,  
 private val navController: NavController,  
) :  
 RecyclerView.Adapter<MessengerAdapter.ViewHolder>() {  
  
 class ViewHolder(  
 private val binding: MessengerRecyclerviewItemBinding,  
 private val db: Database = Database.get(),  
 ) :  
 RecyclerView.ViewHolder(binding.*root*) {  
  
 @SuppressLint("SetTextI18n")  
 fun bind(chatInUser: ChatInUser, nav: (String?, String) -> Unit) = *with*(binding) **{** messengerChatName.*text* = "[CHAT] ${db.listChat[chatInUser.chatId]?.chatName}"  
  
 messengerCard.setOnClickListener **{** Log.d("Adapter", "Clicked")  
 db.chatId = chatInUser.chatId!!  
 val chatId = chatInUser.chatId  
 val chatName = messengerChatName.*text*.toString()  
 nav(chatId, chatName)  
 **}  
 }** companion object {  
 fun create(parent: ViewGroup): ViewHolder {  
 val inflater = LayoutInflater.from(parent.*context*)  
 val bindingTo = MessengerRecyclerviewItemBinding.inflate(inflater, parent, false)  
 return ViewHolder(bindingTo)  
 }  
 }  
 }  
  
 fun submitList(list: List<ChatInUser>) {  
 chatList = list.*toList*()  
 notifyDataSetChanged()  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 val itemView = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*messenger\_recyclerview\_item*, parent, false)  
 return ViewHolder(MessengerRecyclerviewItemBinding.bind(itemView))  
 }  
  
 override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
 holder.bind(chatList[position]) **{** chatId, chatName **->** val bundle = Bundle()  
 bundle.putString(*KEY\_GET\_CHAT\_ID*, chatId)  
 bundle.putString(*KEY\_GET\_CHAT\_NAME*, chatName)  
 navController.navigate(R.id.*chatFragment*, bundle)  
 **}** }  
  
 override fun getItemCount() = chatList.size  
}

Файл UsersAdapter.kt

package com.darf.example.adapters  
  
import android.annotation.SuppressLint  
import android.view.LayoutInflater  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.darf.example.R  
import com.darf.example.databinding.UsersRecyclerviewItemBinding  
import com.darf.example.models.User  
  
class UsersAdapter(private var userList: List<User>) :  
 RecyclerView.Adapter<UsersAdapter.ViewHolder>() {  
  
 class ViewHolder(private val binding: UsersRecyclerviewItemBinding) :  
 RecyclerView.ViewHolder(binding.*root*) {  
  
 @SuppressLint("SetTextI18n")  
 fun bind(user: User) = *with*(binding) **{** usersUserName.*text* = "[USER] ${user.userName}"  
 **}** companion object {  
 fun create(parent: ViewGroup): ViewHolder {  
 val inflater = LayoutInflater.from(parent.*context*)  
 val bindingTo = UsersRecyclerviewItemBinding.inflate(inflater, parent, false)  
 return ViewHolder(bindingTo)  
 }  
 }  
 }  
  
 fun submitList(list: ArrayList<User>) {  
 userList = list.*toList*()  
 notifyDataSetChanged()  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 val itemView = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*users\_recyclerview\_item*, parent, false)  
 return ViewHolder(UsersRecyclerviewItemBinding.bind(itemView))  
 }  
  
 override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
 holder.bind(userList[position])  
 }  
  
 override fun getItemCount() = userList.size  
}

Файл ChatAdapter.kt

package com.darf.example.adapters  
  
import android.annotation.SuppressLint  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.darf.example.Database  
import com.darf.example.R  
import com.darf.example.databinding.ChatRecyclerviewItemLeftBinding  
import com.darf.example.databinding.ChatRecyclerviewItemRightBinding  
import com.darf.example.models.Message  
import com.google.firebase.auth.FirebaseAuth  
import com.google.firebase.auth.ktx.auth  
import com.google.firebase.ktx.Firebase  
  
class ChatAdapter(  
 private var messageList: List<Message>,  
 private val auth: FirebaseAuth = Firebase.*auth*) :  
 RecyclerView.Adapter<ChatAdapter.ViewHolder>() {  
  
 class ViewHolder(  
 itemView: View,  
 private val db: Database = Database.get()  
 ) :  
 RecyclerView.ViewHolder(itemView) {  
  
 @SuppressLint("SetTextI18n")  
 fun bind(message: Message) {  
 val bindingMy =  
 if (*itemViewType* == USER.*MY*.ordinal) ChatRecyclerviewItemRightBinding.bind(itemView) else null  
 val bindingAny =  
 if (*itemViewType* == USER.*ANY*.ordinal) ChatRecyclerviewItemLeftBinding.bind(itemView) else null  
  
 val user = db.mapUser[message.userId]!!  
 if (*itemViewType* == USER.*MY*.ordinal)  
 bindingMy?.*apply* **{** chatMessageUserName.*text* = "[YOU] ${user.userName}"  
 chatMessageText.*text* = message.text  
 **}** else if (*itemViewType* == USER.*ANY*.ordinal)  
 bindingAny?.*apply* **{** chatMessageUserName.*text* = "[USER] ${user.userName}"  
 chatMessageText.*text* = message.text  
 **}** }  
/\*  
 companion object {  
 fun create(parent: ViewGroup): ViewHolder {  
 val inflater = LayoutInflater.from(parent.context)  
 val leftBinding = ChatRecyclerviewItemLeftBinding.inflate(inflater, parent, false)  
 val rightBinding = ChatRecyclerviewItemRightBinding.inflate(inflater, parent, false)  
 return ViewHolder(leftBinding, rightBinding)  
 }  
 }  
 \*/  
 }  
  
 fun submitList(list: List<Message>) {  
 messageList = list.*toList*()  
 notifyDataSetChanged()  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 val itemLeftView = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*chat\_recyclerview\_item\_left*, parent, false)  
 val itemRightView = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*chat\_recyclerview\_item\_right*, parent, false)  
  
 val itemView = when (viewType) {  
 USER.*ANY*.ordinal -> itemLeftView  
 USER.*MY*.ordinal -> itemRightView  
 else -> null  
 }!!  
  
 // вернём itemViewRight, если отрисовываем "моё" сообщение  
 // вернём itemViewLeft, если отрисовываем "чужое" сообщение  
 return ViewHolder(  
// ChatRecyclerviewItemLeftBinding.bind(itemLeftView),  
 itemView  
 )  
 }  
  
 override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
 holder.bind(messageList[position])  
 }  
  
 override fun getItemCount() = messageList.size  
  
 override fun getItemViewType(position: Int): Int = when (messageList[position].userId) {  
 auth.*currentUser*!!.*uid* -> USER.*MY*.ordinal  
 else -> USER.*ANY*.ordinal  
 }  
  
 companion object {  
 private enum class USER {  
 *MY*,  
 *ANY* }  
 }  
}

Файл CreateChatAdapter.kt

package com.darf.example.adapters  
  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.darf.example.R  
import com.darf.example.databinding.CreateChatRecyclerviewItemBinding  
import com.darf.example.models.User  
  
class CreateChatAdapter(  
 private var userList: List<User>,  
) :  
 RecyclerView.Adapter<CreateChatAdapter.ViewHolder>() {  
  
 private val checkedUserList: MutableList<User> = *mutableListOf*()  
  
 class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {  
  
 fun bind(user: User, addUserToList: (Boolean) -> Unit) {  
 val binding = CreateChatRecyclerviewItemBinding.bind(itemView)  
 binding.createChatCheckBox.*text* = user.userName  
  
 binding.createChatCheckBox.*also* **{  
 it**.setOnCheckedChangeListener **{** \_, isChecked **->** addUserToList(isChecked)  
 **}  
 }** }  
 }  
  
 fun submitList(list: List<User>) {  
 userList = list.*toList*()  
 notifyDataSetChanged()  
 }  
  
 fun getCheckedUserList() = checkedUserList.*toList*()  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 val itemView = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*create\_chat\_recyclerview\_item*, parent, false)  
 return ViewHolder(itemView)  
 }  
  
 override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
 val user = userList[position]  
 holder.bind(user) **{** isChecked **->** if (isChecked && user !in checkedUserList)  
 checkedUserList.add(user)  
 else if (!isChecked && user in checkedUserList)  
 checkedUserList.remove(user)  
 **}** }  
  
 override fun getItemCount() = userList.size  
}

Файл AddUserAdapter.kt

package com.darf.example.adapters  
  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.darf.example.R  
import com.darf.example.databinding.AddUserRecyclerviewItemBinding  
import com.darf.example.models.User  
  
class AddUserAdapter(  
 private var userList: List<User>  
) :  
 RecyclerView.Adapter<AddUserAdapter.ViewHolder>() {  
  
 private val checkedUserList: MutableList<User> = *mutableListOf*()  
  
 class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {  
  
 fun bind(user: User, addUserToList: (Boolean) -> Unit) {  
 val binding = AddUserRecyclerviewItemBinding.bind(itemView)  
 binding.addUserCheckBox.*text* = user.userName  
  
 binding.addUserCheckBox.*also* **{  
 it**.setOnCheckedChangeListener **{** \_, isChecked **->** addUserToList(isChecked)  
 **}  
 }** }  
 }  
  
 fun submitList(list: List<User>) {  
 userList = list.*toList*()  
 notifyDataSetChanged()  
 }  
  
 fun getCheckedUserList() = checkedUserList.*toList*()  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 val itemView = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*add\_user\_recyclerview\_item*, parent, false)  
 return ViewHolder(itemView)  
 }  
  
 override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
 val user = userList[position]  
 holder.bind(user) **{** isChecked **->** if (isChecked && user !in checkedUserList)  
 checkedUserList.add(user)  
 else if (!isChecked && user in checkedUserList)  
 checkedUserList.remove(user)  
 **}** }  
  
 override fun getItemCount() = userList.size  
}

# ПРИЛОЖЕНИЕ Г Модели данных

Файл Chat.kt

package com.darf.example.models  
  
data class Chat(  
 val chatId: String? = null,  
 val chatName: String? = null  
)

Файл User.kt

package com.darf.example.models  
  
data class User(  
 val userId: String? = null,  
 val userName: String? = null,  
 val aboutUser: String? = null  
)

Файл Message.kt

package com.darf.example.models  
  
data class Message(  
 val messageId: String? = null,  
 val chatId: String? = null,  
 val userId: String? = null,  
 val text: String? = null  
)

Файл ChatInUser.kt

package com.darf.example.models  
  
data class ChatInUser(  
 val chatInUserId: String? = null,  
 val userId: String? = null,  
 val chatId: String? = null  
)

Файл UserInChat.kt

package com.darf.example.models  
  
data class UserInChat(  
 val userInChatId: String? = null,  
 val userId: String? = null,  
 val chatId: String? = null  
)

# ПРИЛОЖЕНИЕ Д Класс Database

Файл Database.kt

package com.darf.example  
  
import androidx.collection.ArrayMap  
import androidx.navigation.NavController  
import com.darf.example.adapters.\*  
import com.darf.example.models.\*  
import com.google.firebase.auth.ktx.auth  
import com.google.firebase.database.DataSnapshot  
import com.google.firebase.database.DatabaseError  
import com.google.firebase.database.ValueEventListener  
import com.google.firebase.database.ktx.database  
import com.google.firebase.ktx.Firebase  
  
class Database private constructor() {  
 private val database = Firebase.*database* val auth = Firebase.*auth* val user = auth.*currentUser*!!  
  
 var chatId = ""  
  
 val listChat = ArrayMap<String, Chat>()  
 val listUser = ArrayList<User>()  
 val mapUser = ArrayMap<String, User>()  
 val listMessage = ArrayMap<String, MutableList<Message>>()  
 val listChatInUser = ArrayList<ChatInUser>()  
 val listUserInChat = ArrayMap<String, MutableList<UserInChat>>()  
  
 var adapterMessenger: MessengerAdapter? = null  
 val adapterAddUser = AddUserAdapter(listUser)  
 val adapterCreateChat = CreateChatAdapter(listUser)  
 val adapterUsers = UsersAdapter(listUser)  
 val adaptersChat = ArrayMap<String, ChatAdapter>()  
  
 init {  
 adaptersChat[chatId]  
 }  
  
 private val dbRefChat = database.getReference("node\_chat")  
 private val dbRefUser = database.getReference("node\_user")  
 private val dbRefMessage = database.getReference("node\_message")  
 private val dbRefChatInUser = database.getReference("node\_chat\_in\_user")  
 private val dbRefUserInChat = database.getReference("node\_user\_in\_chat")  
  
 private fun keyChat() = dbRefChat.push().*key*!!  
 private fun keyMessage(chatId: String) = dbRefMessage.child(chatId).push().*key*!!  
 private fun keyChatInUser(userId: String) = dbRefChatInUser.child(userId).push().*key*!!  
 private fun keyUserInChat(chatId: String) = dbRefUserInChat.child(chatId).push().*key*!!  
  
 // LISTENERS  
 fun listenerChat() {  
// val listChatId = listChatInUser.map { it.chatId!! }.listIterator()  
 dbRefChat.addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 listChat.clear()  
 snapshot.*children*.*forEach* **{** snap **->**// if (snap.key == listChatId.next()) {  
 val chat = snap.getValue(Chat::class.*java*)!!  
 listChat[snap.*key*] = chat  
// }  
 **}** }  
  
 override fun onCancelled(error: DatabaseError) = Unit  
 })  
 }  
  
 fun listenerUser() {  
 dbRefUser.addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 listUser.clear()  
 mapUser.clear()  
 snapshot.*children*.*forEach* **{** snap **->** val user = snap.getValue(User::class.*java*)!!  
 listUser += user  
 mapUser[user.userId] = user  
 **}** adapterUsers.submitList(listUser)  
 adapterCreateChat.submitList(listUser - mapUser[user.*uid*]!!)  
 }  
  
 override fun onCancelled(error: DatabaseError) = Unit  
 })  
 }  
  
 fun listenerMessage() {  
// val listChatId = listChatInUser.map { it.chatId!! }.listIterator()  
 dbRefMessage.addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 listMessage.clear()  
 snapshot.*children*.*forEach* **{** snapChatId **->**// var id: String  
// if (snapChatId.key == (listChatId.next().also { id = it })) {  
 val listMsg = ArrayList<Message>()  
 snapChatId.*children*.*forEach* **{** snap **->** val message = snap.getValue(Message::class.*java*)!!  
 listMsg += message  
 **}** val id = snapChatId.*key* //?  
 listMessage[id] = listMsg  
 if (adaptersChat[id] == null) adaptersChat[id] = ChatAdapter(listMsg)  
 else adaptersChat[id]!!.submitList(listMsg)  
// }  
 **}** }  
  
 override fun onCancelled(error: DatabaseError) = Unit  
 })  
 }  
  
 fun listenerMessageChild() {  
 dbRefMessage.child(chatId).addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 listMessage[chatId]?.clear()  
 val listMsg = ArrayList<Message>()  
 snapshot.*children*.*forEach* **{** snap **->** val message = snap.getValue(Message::class.*java*)!!  
 listMsg += message  
 **}** listMessage[chatId] = listMsg  
 if (adaptersChat[chatId] == null) adaptersChat[chatId] = ChatAdapter(listMsg)  
 else adaptersChat[chatId]!!.submitList(listMsg)  
 }  
  
 override fun onCancelled(error: DatabaseError) = Unit  
 })  
 }  
  
 fun listenerChatInUserChild() {  
 dbRefChatInUser.child(user.*uid*).addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 listChatInUser.clear()  
 snapshot.*children*.*forEach* **{** snap **->** val chatInUser = snap.getValue(ChatInUser::class.*java*)!!  
 listChatInUser += chatInUser  
 **}** adapterMessenger!!.submitList(listChatInUser)  
 }  
  
 override fun onCancelled(error: DatabaseError) = Unit  
 })  
 }  
  
 fun listenerUserInChat() {  
// val listChatId = listChatInUser.map { it.chatId!! }.listIterator()  
 dbRefUserInChat.addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 listUserInChat.clear()  
 snapshot.*children*.*forEach* **{** snapChatId **->** // не хватает условия сравнения snapChatId == listChatId  
 val listUIC = ArrayList<UserInChat>()  
 snapChatId.*children*.*forEach* **{** snap **->** val userInChat = snap.getValue(UserInChat::class.*java*)!!  
 listUIC += userInChat  
 **}**// listUserInChat[listChatId.next()] = listUIC  
 listUserInChat[snapChatId.*key*] = listUIC //?  
 **}** }  
  
 override fun onCancelled(error: DatabaseError) = Unit  
 })  
 }  
  
 fun listenerUserInChatChild() {  
 dbRefUserInChat.child(chatId).addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 listUserInChat[chatId]?.clear()  
 val listUIC = ArrayList<UserInChat>()  
 snapshot.*children*.*forEach* **{** snap **->** val userInChat = snap.getValue(UserInChat::class.*java*)!!  
 listUIC += userInChat  
 **}** listUserInChat[chatId] = listUIC  
 val listU = listUIC.*map* **{** mapUser[**it**.userId] **}** adapterAddUser.submitList(listUser.*filterNot* **{** listU.contains(**it**) **}**)  
 }  
  
 override fun onCancelled(error: DatabaseError) = Unit  
 })  
 }  
  
 // SET VALUES  
 fun setValueChat(chatName: String = "Chat name #${(1..100).*shuffled*().*last*()}") {  
 val key = keyChat()  
 chatId = key  
 dbRefChat.child(key).setValue(  
 Chat(  
 chatId = key,  
 chatName = chatName  
 )  
 )  
 }  
  
 fun setValueUser(aboutUser: String = "About user #${(1..100).*shuffled*().*last*()}") {  
 dbRefUser.child(user.*uid*).setValue(  
 User(  
 userId = user.*uid*,  
 userName = user.*displayName*,  
 aboutUser = aboutUser  
 )  
 )  
 }  
  
 fun setValueMessage(text: String = "Message text #${(1..100).*shuffled*().*last*()}") {  
 val key = keyMessage(chatId)  
 dbRefMessage.child(chatId).child(key).setValue(  
 Message(  
 messageId = key,  
 chatId = chatId,  
 userId = user.*uid*,  
 text = text  
 )  
 )  
 }  
  
 fun setValueChatInUser(userId: String = this.user.*uid*, chatId: String = this.chatId) {  
 val key = keyChatInUser(userId)  
 dbRefChatInUser.child(userId).child(key).setValue(  
 ChatInUser(  
 chatInUserId = key,  
 userId = userId,  
 chatId = chatId  
 )  
 )  
 }  
  
 fun setValueUserInChat(userId: String = this.user.*uid*, chatId: String = this.chatId) {  
 val key = keyUserInChat(chatId)  
 dbRefUserInChat.child(chatId).child(key).setValue(  
 UserInChat(  
 userInChatId = key,  
 userId = userId,  
 chatId = chatId  
 )  
 )  
 }  
  
 companion object {  
 private var obj: Database? = null  
  
 fun instance(): Database {  
 if (obj == null)  
 obj = Database()  
 return obj!!  
 }  
  
 fun get(): Database = obj!!  
 }  
}