МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ

КАЛИНИНГРАДСКОЙ ОБЛАСТИ

ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ

«КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СТРОИТЕЛЬСТВА»

(ГБУ КО ПОО «КИТиС»)

|  |  |  |
| --- | --- | --- |
| Курсовой проект  допущен к защите  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата)  Зам.директора по УМР  \_\_\_\_\_\_\_\_\_\_\_\_\_Павленко Г.Я.  (подпись) |  | Курсовой проект  защищен с оценкой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  Руководитель работы  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Пояснительная записка к курсовому проекту

по дисциплине: МДК 01.01 Разработка программных модулей

Тема: Разработка мобильного приложения для проведения процедуры голосования

Специальность 09.02.07 «Информационные системы и программирование»

|  |  |
| --- | --- |
| Выполнила: | студентка 3 курса,  группы ИСп21-1  Смирнова Валерия  Алексеевна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Руководитель: | Большакова-Стрекалова Анна Викторовна, преподаватель ГБУ КО ПОО КИТиС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Калининград

2024

Аннотация

Тема курсового проекта: создание мобильного приложения для проведения процедуры голосования.

Работа состоит из: введения, трех глав, заключения, списка источников, используемых в процессе выполнения работы, а также листинга программ, упоминаемых при описании программного продукта.

Во введении обоснована актуальность предметной области и общие идеи цели и задач.

Первая часть "Описание предметной области" углубляется в аналогии с разрабатываемым приложением и определяет его технические требования.

Вторая часть, "Описание структуры приложения", обосновывает выбор средств разработки, разрабатывает интерфейс и логику работы приложения, а также описывает его компоненты.

Третья часть включает в себя тестирование и установку приложения. В заключении делаются основные выводы по разработке мобильного приложения.

Ключевые слова проекта: мобильное приложение, голосование, результаты голосования, аутентификация, организатор голосования. ПО разработано с использованием Android Studio версии Iguana 2023.2.1 Patch 2.

Общий объём работы 77 страниц.

Ссылка на GitHub репозиторий: https://github.com/Leksmr05/Voice.git

СОДЕРЖАНИЕ

[Введение 4](#_Toc165514189)

[1 Описание предметной области 5](#_Toc165514190)

[1.1 Аналоги разрабатываемого приложения 5](#_Toc165514191)

[1.2 Техническое задание 9](#_Toc165514192)

[1.3 Описание структуры программы 12](#_Toc165514193)

[2 Описание разработки приложения 13](#_Toc165514194)

[2.1 Обоснование средств разработки 13](#_Toc165514195)

[2.2 Разработка интерфейса 18](#_Toc165514196)

[2.3 Разработка логики работы приложения. Схема взаимодействия компонентов проекта 24](#_Toc165514197)

[2.4 Описание переменных, компонентов, классов и подпрограмм 25](#_Toc165514198)

[3 Тестирование и установка приложения 37](#_Toc165514199)

[Заключение 40](#_Toc165514200)

[Список использованной литературы 41](#_Toc165514201)

[Приложение А. Листинг программы 42](#_Toc165514202)

[Приложение Б. Руководство пользователя 75](#_Toc165514203)

# Введение

В современном обществе процессы голосования играют ключевую роль в принятии важных решений на всех уровнях - от выборов национальных лидеров до голосований в учебных заведениях. С развитием технологий и распространением мобильных устройств электронные системы голосования становятся все более популярными и востребованными.

Целью данного курсового проекта является разработка современного приложения для голосования. Это приложение будет предоставлять студентам возможность участвовать в различных голосованиях.

Проект будет фокусироваться не только на функциональности приложения, но и на обеспечении его безопасности и защиты данных, а также на обеспечении удобства использования для всех участников процесса голосования. Он будет учитывать особенности студенческой среды и потребности пользователей, обеспечивая простоту использования и надежность системы.

Актуальность данного проекта подтверждается потребностью в современных решениях для организации голосований и выборов в учебных заведениях. Электронные системы голосования становятся все более популярными благодаря своему удобству и эффективности. Приложение для голосования в колледже направлено на упрощение процессов выборов студенческого совета, определения приоритетов и принятия важных решений в учебной среде.

1. Описание предметной области

Приложение для голосования в колледже разработано для управления процессом голосования студентов по различным вопросам, связанным с учебным процессом, организационными вопросами и другими важными решениями, принимаемыми в образовательном учреждении.

Результатом приложения является обеспечение прозрачности, эффективности и удобства в проведении голосований, а также в создании механизмов для демократического участия студентов в управлении колледжа.

* 1. Аналоги разрабатываемого приложения

На данный момент существует не очень много мобильных приложений для проведения процедуры голосования, направленных для учебных заведений, но имеются обычные приложения, заключающиеся в выборах и опросах.

Можно выделить следующие основные аналоги:

* «Опросы для всех»

Мобильное приложение «Опросы для всех» предназначено для создания опросов, соответствующие предпочтениям пользователей, и участие в опросах, которые им интересны. Независимо от того, хочет ли пользователь опросить своих друзей на частную тему или узнать, что думают люди о последних новостях. Иконка и структура приложения изображена на 1, 2, 3 и 4 рисунке.



Рисунок 1 - Иконка приложения «Опросы для всех»

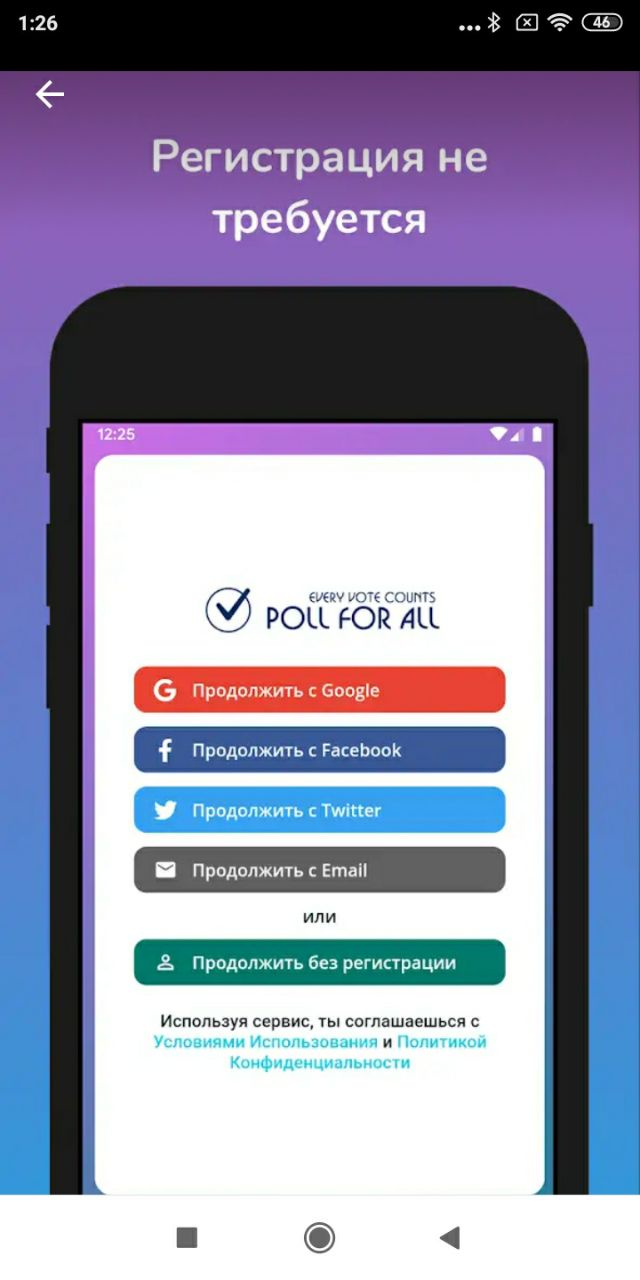


Рисунок 2 - Раздел «Авторизация» приложения «Опросы для всех»

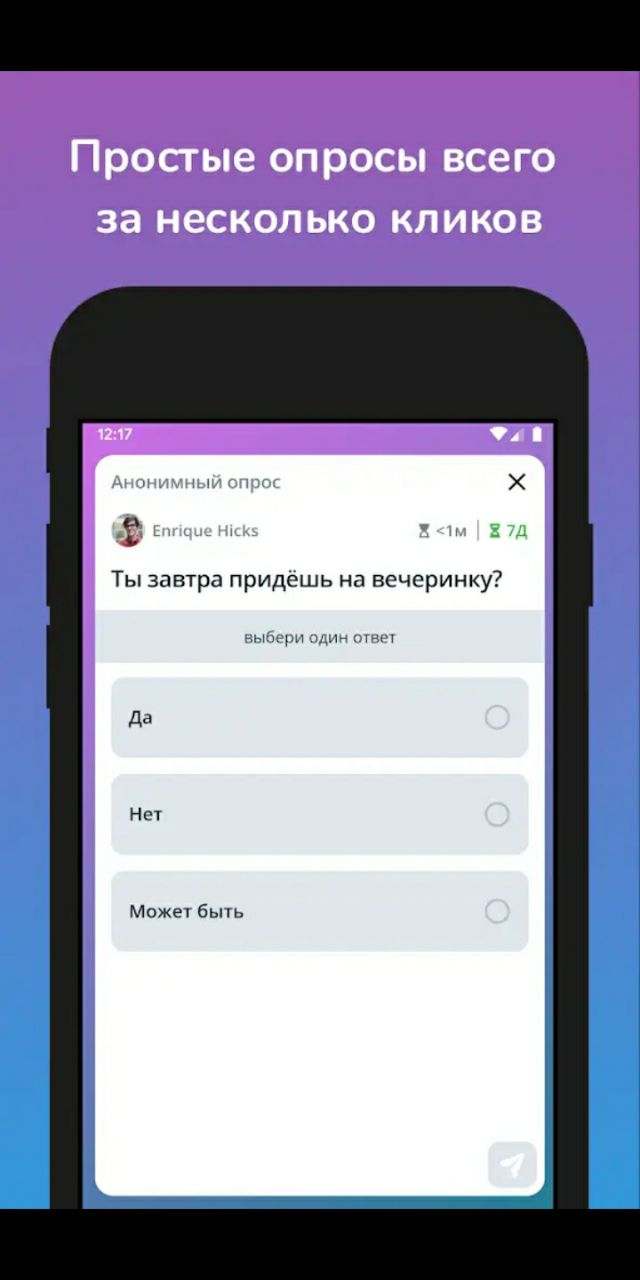


Рисунок 3 - Раздел «Опросы» приложения «Опросы для всех»

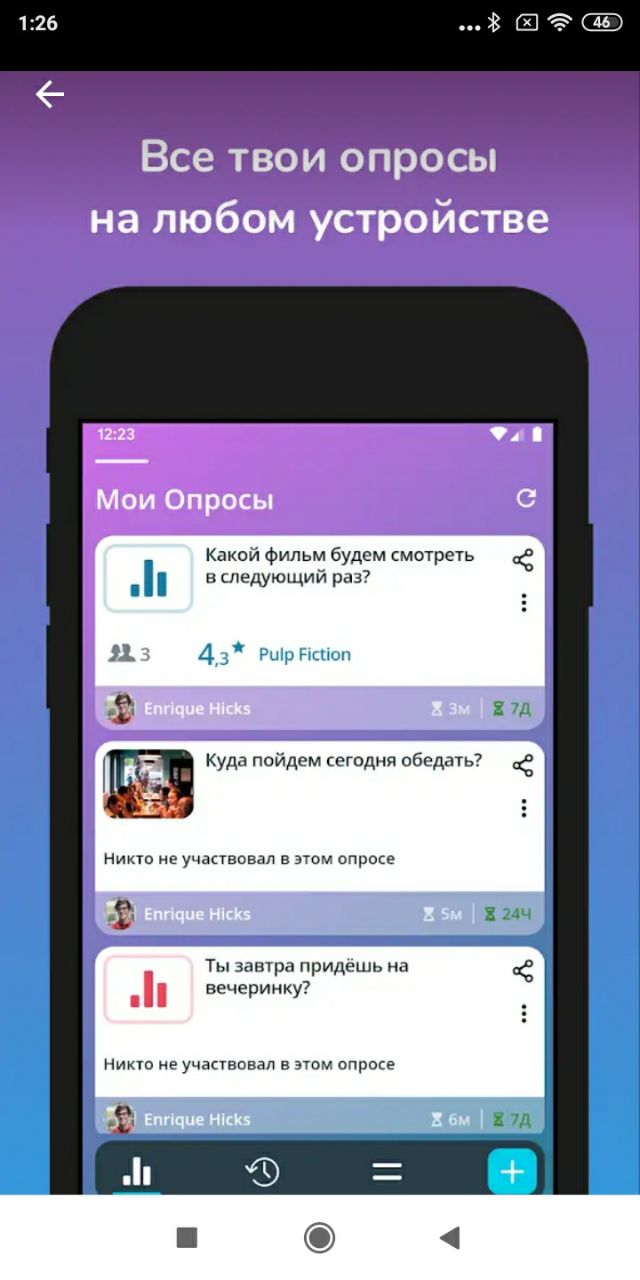


Рисунок 4 - Раздел «Мои опросы» приложения «Опросы для всех»

Данное мобильное приложение дает возможность пользователям участвовать в опросах и создавать свои опросы без регистрации.

На основе аналога данного мобильного приложения можно подчеркнуть удобство доступа опроса только определенным пользователям. Однако существуют и минусы, такие как возможные сбои при участии в анонимном опросе и создание опросов без регистрации.

* «Pollie»

Мобильное приложение «Pollie» предназначено для создания голосования, в котором можно установить дату закрытия опроса, и так же выбрать то, что увидят ваши пользователи. Иконка и структура приложения представлена на 5, 6, 7 и 8 рисунке.



Рисунок 5 - Иконка приложения «Pollie»

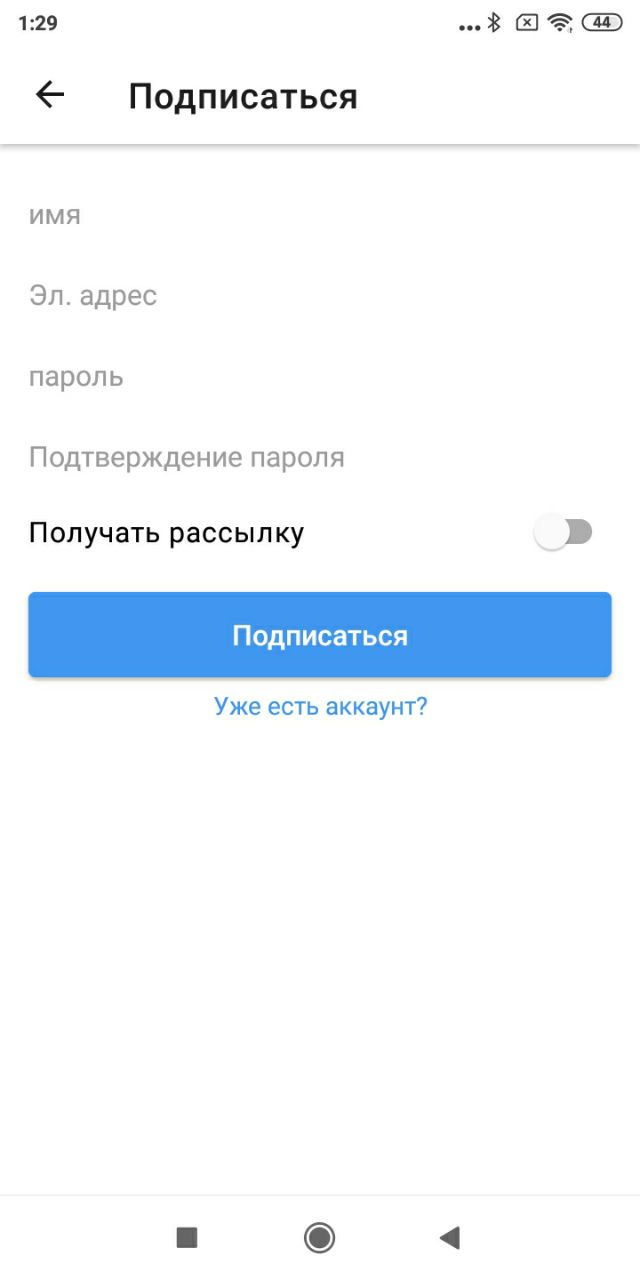


Рисунок 6 – Раздел «Регистрация» приложения «Pollie»



Рисунок 7 - Раздел «Мои опросы» приложения «Pollie»

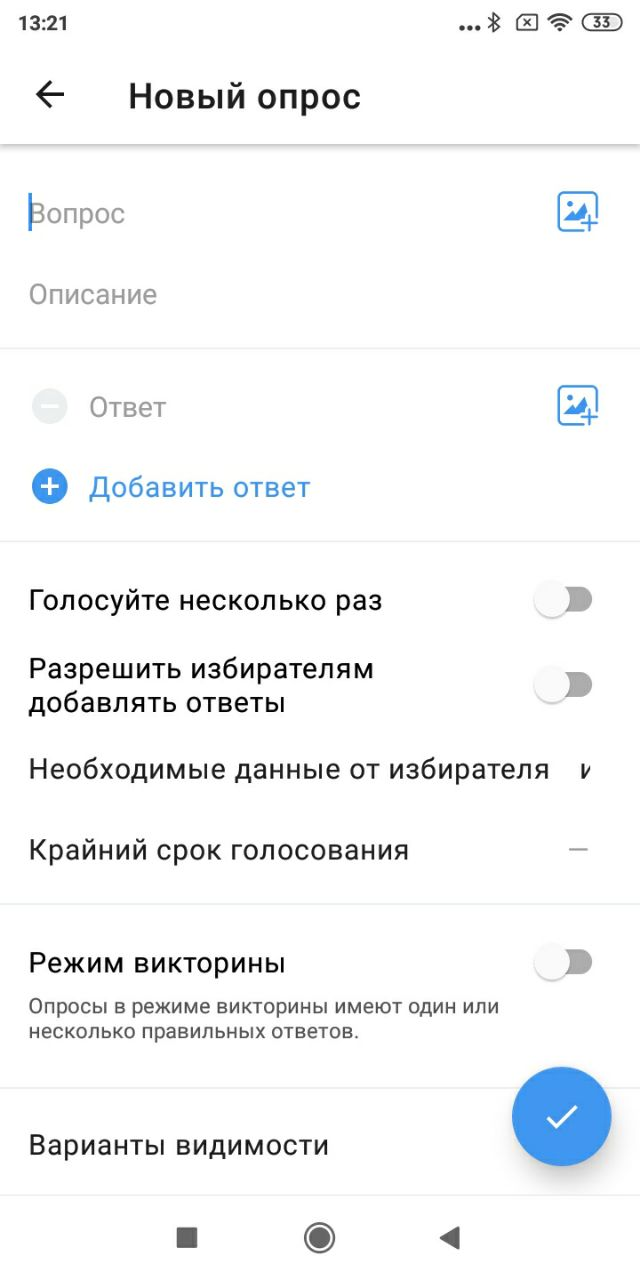


Рисунок 8 - Раздел «Новый опрос» приложения «Pollie»

На основе мобильного приложения «Pollie» можно выделить распространение опроса по ссылке, перейдя по которой пользователь может проголосовать в опросе не скачивая приложение. Однако существуют и минусы, такие как возможные проблемы с точностью данных и конфиденциальностью информации пользователей.

Исходя из результатов анализа мобильных приложений выше, можно сделать вывод, что наиболее подходящим аналогом для создания программного изделия будет приложение «Опросы Для Всех».

Будущее мобильное приложение должно содержать простой интерфейс и доступность использования для каждого пользователя, а также удобное расположение всех компонентов на экране.

* 1. Техническое задание

1. Общие сведения

* Наименование: Мобильное приложение «Voice».
* Назначение и область применения: Данное мобильное приложение «Voice» будет предоставлять студентам возможность участвовать в различных голосованиях. Это позволит обеспечить удобства использования для всех участников процесса голосования.

1. Цели создания системы

* Предоставления пользователям удобного и дистанционного голосования в вопросах, которые интересуют пользователей.

1. Требования к системе

Требования к структуре и функционированию системы:

- Система должна быть организована так, чтобы каждая ее часть работала максимально эффективно, с учетом специфики функционала.

- Система должна легко поддаваться расширению и обновлению для адаптации к изменяющимся требованиям и возможностям.

Требования безопасности:

* Система должна быть обеспечена комплексными мерами безопасности для защиты от угроз.

Требования к надежности:

* Система должна обеспечивать высокую степень надежности с целью минимизации риска возникновения сбоев и аварий.

Требования к эргономике и технической эстетике:

* Интерфейс системы должен быть интуитивно понятным и удобным для пользователей.

Требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы:

* Система должна иметь подробные инструкции по эксплуатации и обслуживанию, а также обеспечивать регулярное обновление и резервное копирование данных для их сохранности.

Требования к защите информации от несанкционированного доступа:

* В системе должны быть реализованы механизмы аутентификации, авторизации и шифрования данных для обеспечения конфиденциальности и безопасности информации.

Требования по сохранности информации при авариях:

* Предусмотреть меры по резервному копированию и восстановлению данных в случае возникновения аварийных ситуаций.

Дополнительные требования:

* Необходимо реализовать дополнительные функции и возможности, учитывающие специфику бизнес-процессов и потребности пользователей.

Требования к качеству реализации функций, задач:

* Необходима высокая точность, эффективность и надежность подсистемы для минимизации ошибок и обеспечения быстрой обработки данных.

Требования к программному обеспечению

* Программное обеспечение будет совместимо с операционной системой «Android».

Требования к документации

* Техническая документация должна содержать техническое задание, спецификацию требований, документацию по программному продукту.
* Должна присутствовать пользовательская документация (руководство пользователя).

Требования к техническому обеспечению:

* Система должна работать на мобильных устройствах с Android не ниже 5 версии, чтобы быть доступной для широкого круга пользователей.

Требования к сопровождению

* Должна производиться поддержка и обновление приложения после его внедрения до конца жизненного цикла продукта.

1. Стадии разработки:

* Постановка целей и задач программного продукта
* Анализ аналогичных продуктов
* Изучение необходимых технологий для разработки программного продукта
* Разработка программного продукта
* Тестирование программного продукта
  1. Описание структуры программы

Структура мобильного приложения может варьироваться в зависимости от выбранной платформы и используемых технологий. Однако, общая структура приложения может включать следующие основные компоненты:

* Экран заставки, который будет временным экраном, информирующим пользователя о текущем процессе загрузки приложения.
* Экран аутентификации и идентификации, который будет предназначен для процесса входа пользователя в приложение. Пользователь сможет ввести свои учетные данные. Также будет возможность, регистрации нового пользователя.
* Экран создания профиля, который предоставит возможность заполнить профиль пользователю.
* Экран голосования, где пользователи смогут голосовать в понравившихся им опросах, а также видеть сколько человек проголосовало за и против.
* Экран мои голоса, который позволит пользователям видеть, за что они проголосовали.
* Экран создания голосования, в котором пользователь сможет лично создать свой опрос, чтоб другие пользователи смогли также проголосовать.

1. Описание разработки приложения

Разработка приложения включает несколько этапов:

* Начало проекта: начинается с запуска среды разработки и создания нового проекта с использованием пустого шаблона. Здесь указывается название приложения и выбирается язык программирования.
* Добавление ресурсов: В этом этапе загружаются изображения, которые будут использоваться в приложении, в папку res/drawable.
* Создание экранов: разрабатываются различные экраны приложения с использованием файлов макетов XML. В этих макетах определяются элементы пользовательского интерфейса, такие как ImageView, TextView и Button, для отображения изображений и текста.
* Реализация логики: на этом этапе добавляется логика в приложение для обеспечения правильного порядка отображения экранов и обработки пользовательского ввода. Управление жизненным циклом экранов осуществляется с использованием классов Activity, а для передачи данных между экранами - классов Intent.
* Тестирование: проверяется работоспособность приложения на различных устройствах с целью выявления и исправления ошибок. Проверяется корректное отображение изображений и текста, а также правильная обработка пользовательского ввода. Обоснование средств разработки
  1. Обоснование средств разработки

Платформы разработки программных приложений специально созданы для разработки, тестирования, отладки и выпуска приложений различных типов, предназначенных для различных платформ.

Flutter:

Иконка «Flutter» представлена на рисунке 9.



Рисунок 9 – Иконка платформы разработки «Flutter»

Фреймворк от Google, который позволяет разрабатывать кроссплатформенные мобильные приложения с использованием языка программирования Dart. Flutter обеспечивает быструю разработку и высокую производительность благодаря своей архитектуре, основанной на виджетах.

Структура «Flutter» представлена на рисунке 10.

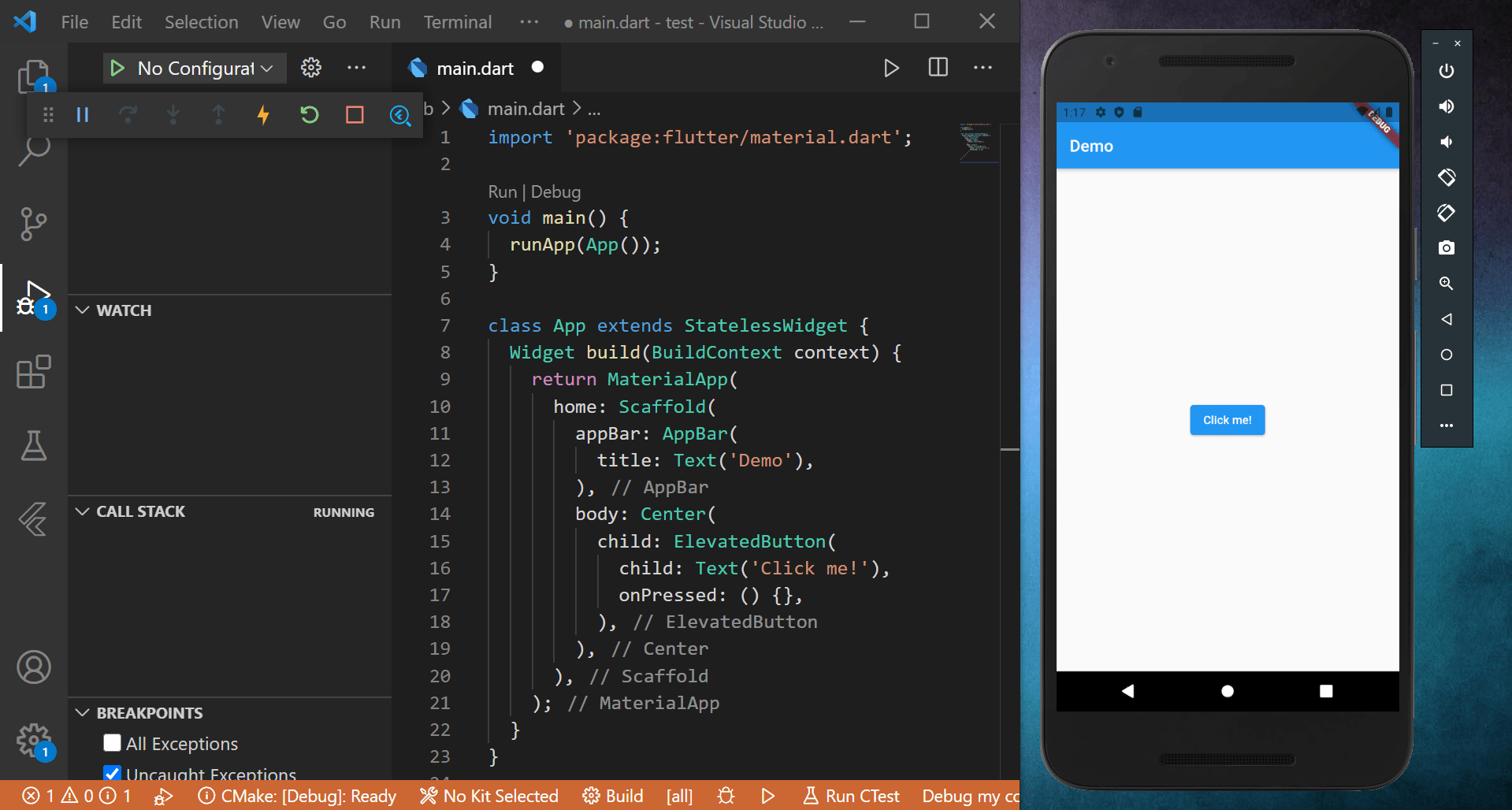


Рисунок 10 – Общий вид среды разработки «Flutter»

Android Studio:

Иконка «Android Studio» представлена на рисунке 11.



Рисунок 11 – Иконка платформы разработки «Android Studio»

Данное приложение разработки мобильных приложений поддерживает различные языки программирования, такие как C/C++, Kotlin и Java, обладает встроенным эмулятором и обширной библиотекой с разнообразными шаблонами, и компонентами, что значительно облегчает и ускоряет процесс создания программного обеспечения.

Структура «AndroidStudoi» представлена на рисунке 12.

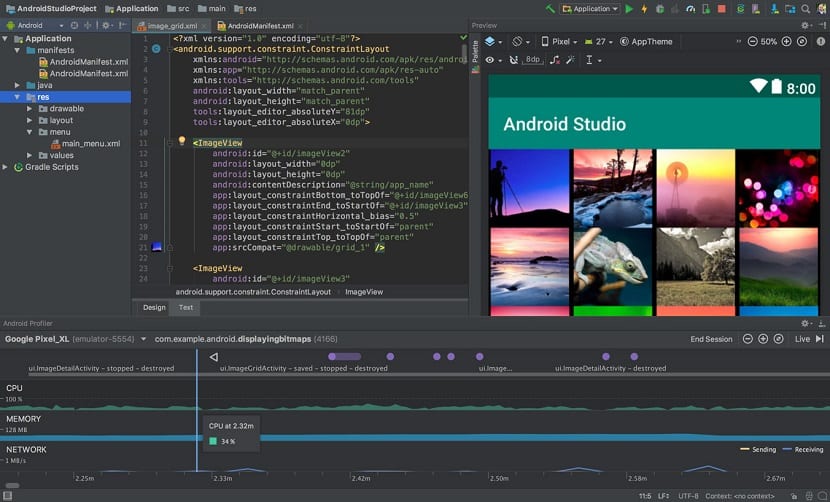


Рисунок 12 – Общий вид среды разработки «Android Studio»

Unity:

Иконка «Unity» представлена на рисунке 13.



Рисунок 13 – Иконка платформы разработки «Unity»

Хоть Unity в первую очередь ассоциируется с разработкой игр, он также может быть использован для создания мобильных приложений, в том числе и для Android. Unity обеспечивает кроссплатформенную разработку и поддерживает языки программирования C# и JavaScript.

Структура «Unity» представлена на рисунке 14.

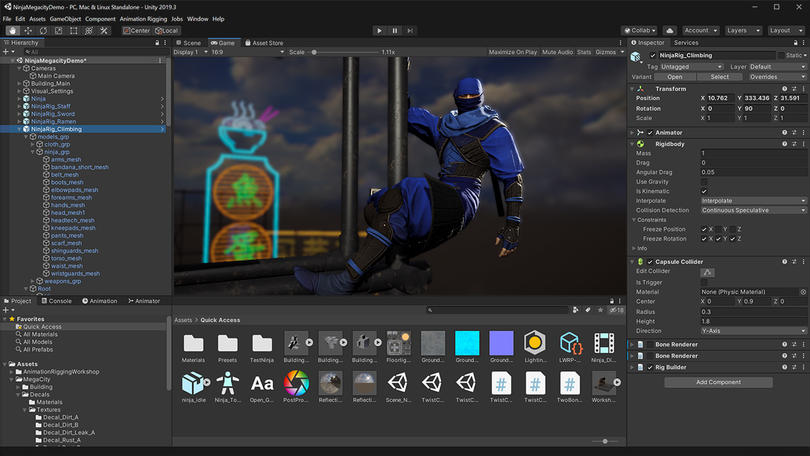


Рисунок 14 – Общий вид среды разработки «Unity»

В ходе разработки мобильного приложения «Voice» было решено использовать «Android Studio» в качестве среды разработки. Это обосновано тем, что Android Studio является официальным интегрированным средством разработки (IDE) для создания приложений под платформу Android, предоставляя обширный набор инструментов и ресурсов, необходимых для разработки высококачественных мобильных приложений. Кроме того, Android Studio обладает удобным интерфейсом, хорошей поддержкой различных языков программирования (в том числе Java и Kotlin) и обширным сообществом разработчиков, что делает его оптимальным выбором для создания мобильных приложений под платформу Android.

База данных предоставляет возможность быстро перейти к стандартизированной и распределенной инфраструктуре, которая обеспечивает масштабирование, эластичность и гибкость для мобильных приложений.

MySQL:

Иконка базы данных представлена на рисунке 15.



Рисунок 15 – Иконка базы данных «MySQL»

Эта система применяется как для маленьких, так и для крупных интернет-проектов, и ее бесплатность является значимой особенностью. MySQL отличается надежностью, высокой скоростью и гибкостью, но наиболее подходит для языков программирования, подобных C++. Структура базы данных представлена на рисунке 16.

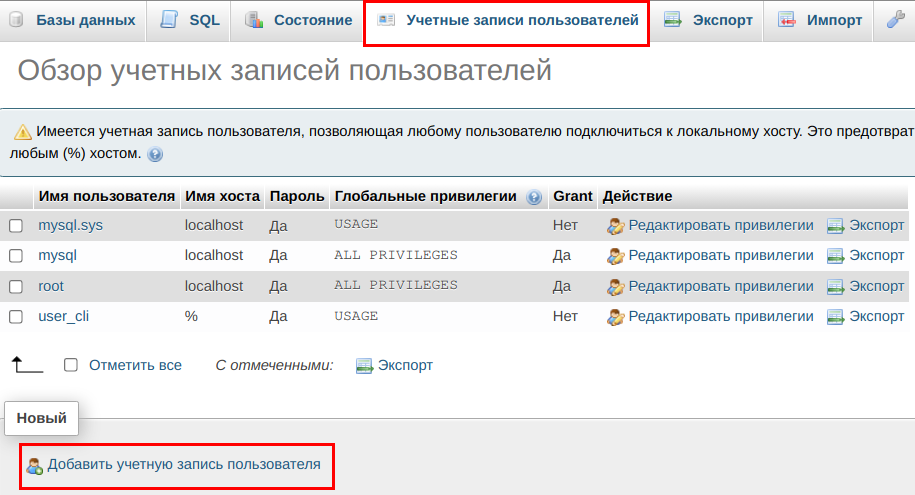


Рисунок 16 – Общий вид базы данных «MySQL»

Firebase:

Иконка базы данных представлена на рисунке 17.



Рисунок 17 – Иконка базы данных «Firebase»

Firebase - облачная база данных, которая обеспечивает хранение и доступ к данным, а также предоставляет удобные инструменты для их управления. Она использует JSON для хранения текстовых данных и предоставляет простые методы для работы с ними. Кроме того, Firebase поддерживает функционал для регистрации и аутентификации пользователей, управления сессиями и легкий доступ к медиафайлам через Cloud Storage.

Однако Firebase не является полностью бесплатным, и некоторые ключевые возможности доступны только на платной основе. Ее недостатком является зависимость от Интернета, что может быть проблематично из-за его ненадежности, а также ограниченная поддержка языков программирования, что может привести к проблемам с интеграцией в программное обеспечение. Структура базы данных представлена на рисунке 18.

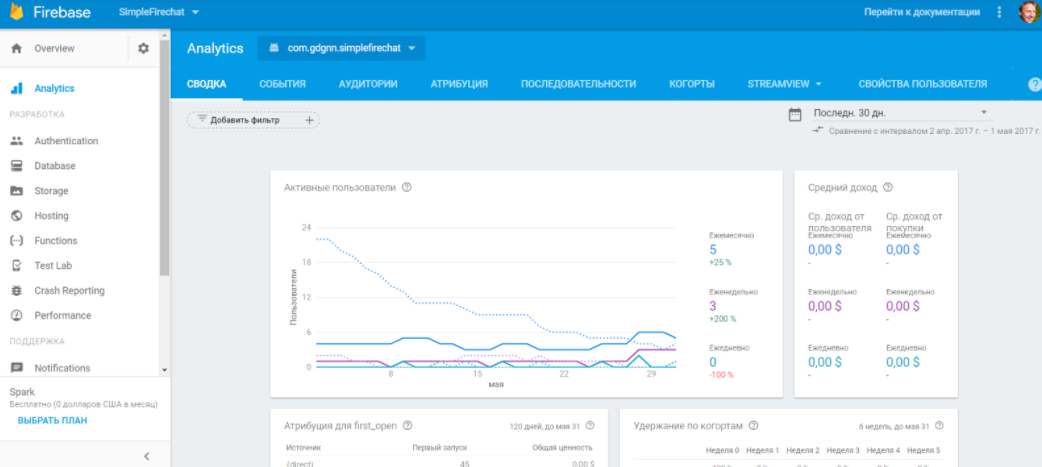


Рисунок 18 – Общий вид сайта базы данных «Firebase»

Подводя итоги, было принято решение в пользу базы данных "Firebase".

Это обосновано несколькими факторами. Во-первых, Firebase предоставляет удобный инструментарий для создания запросов и управления данными. Во-вторых, ее облачная природа обеспечивает высокую скорость работы и возможность хранения данных в облаке, что снижает нагрузку на локальные системы. Также, Firebase тщательно протестирована, что гарантирует надежность ее компонентов и снижает риск неожиданного поведения. Ее компактный размер и открытый доступ также были важными факторами. Не менее важно то, что использование Firebase не ограничено юридическими препятствиями. И, наконец, кроссплатформенность и возможность работы в автономном режиме делают Firebase привлекательным выбором для мобильных приложений.

* 1. Разработка интерфейса

При создании мобильного приложения для проведения процедуры голосования, разработка дизайна и интерфейса было полностью сделано в среде разработки Android Studio.

Приложение «Voice»:

Иконка и содержание приложения «Voice» представлена на рисунке 19-28.



Рисунок 19 – Иконка приложения «Voice»

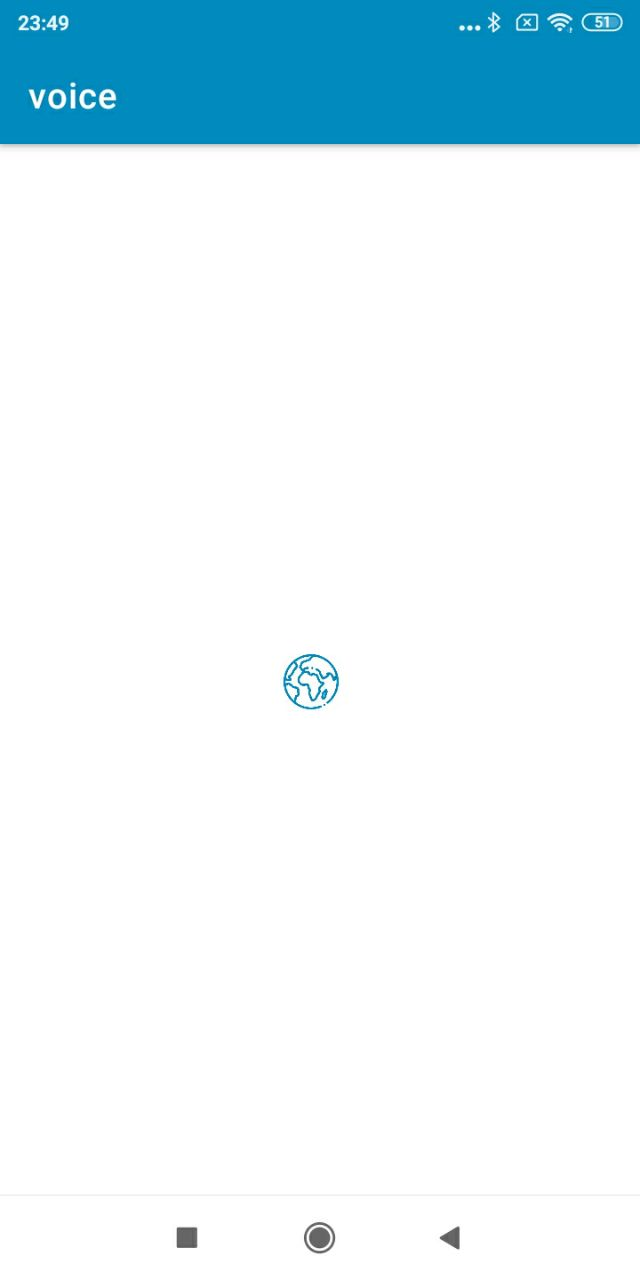


Рисунок 20 – Экран заставки приложения «Voice»

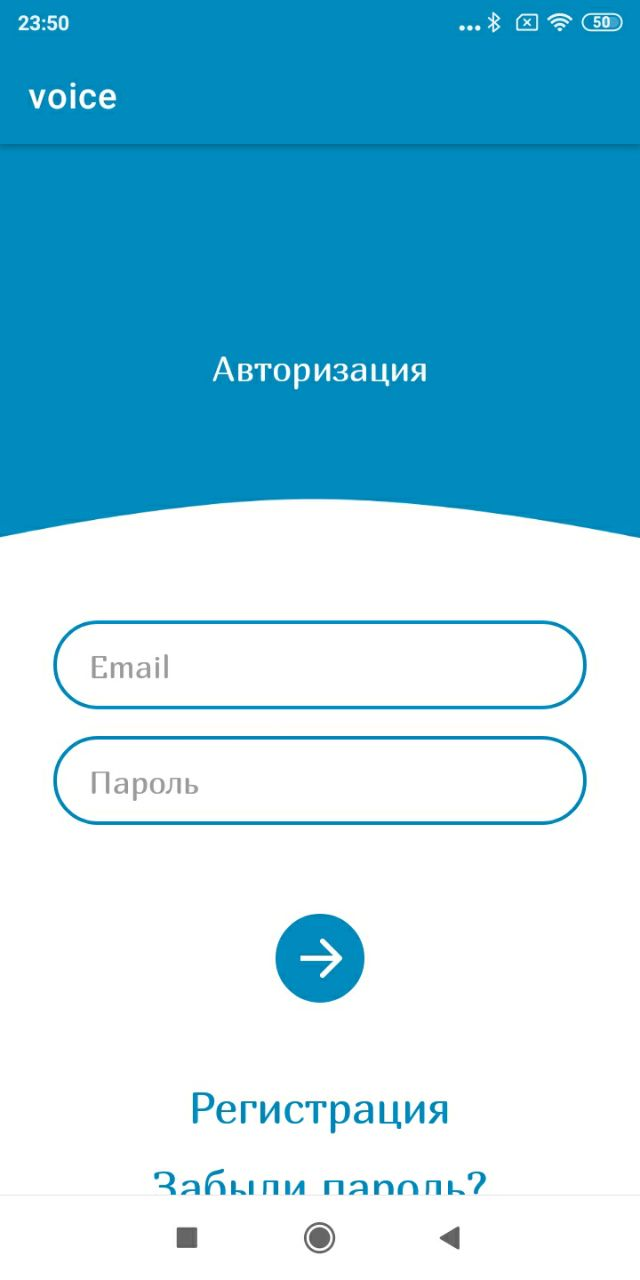


Рисунок 21 – Экран входа в существующий аккаунт

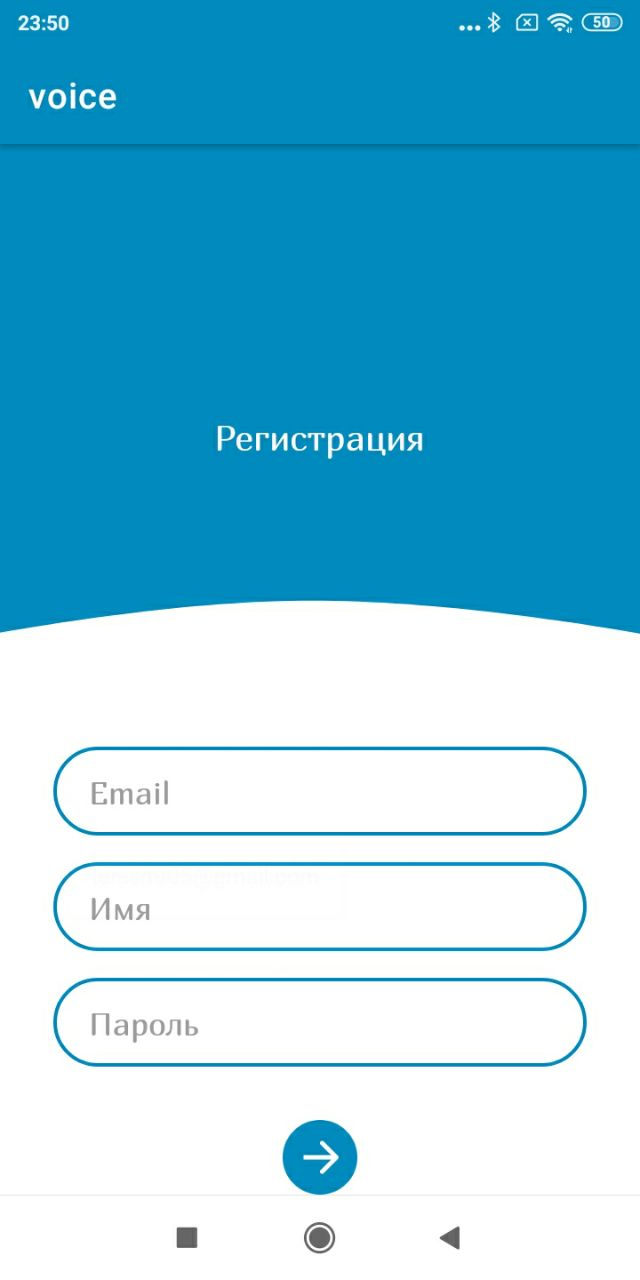


Рисунок 22 – Экран регистрации аккаунта

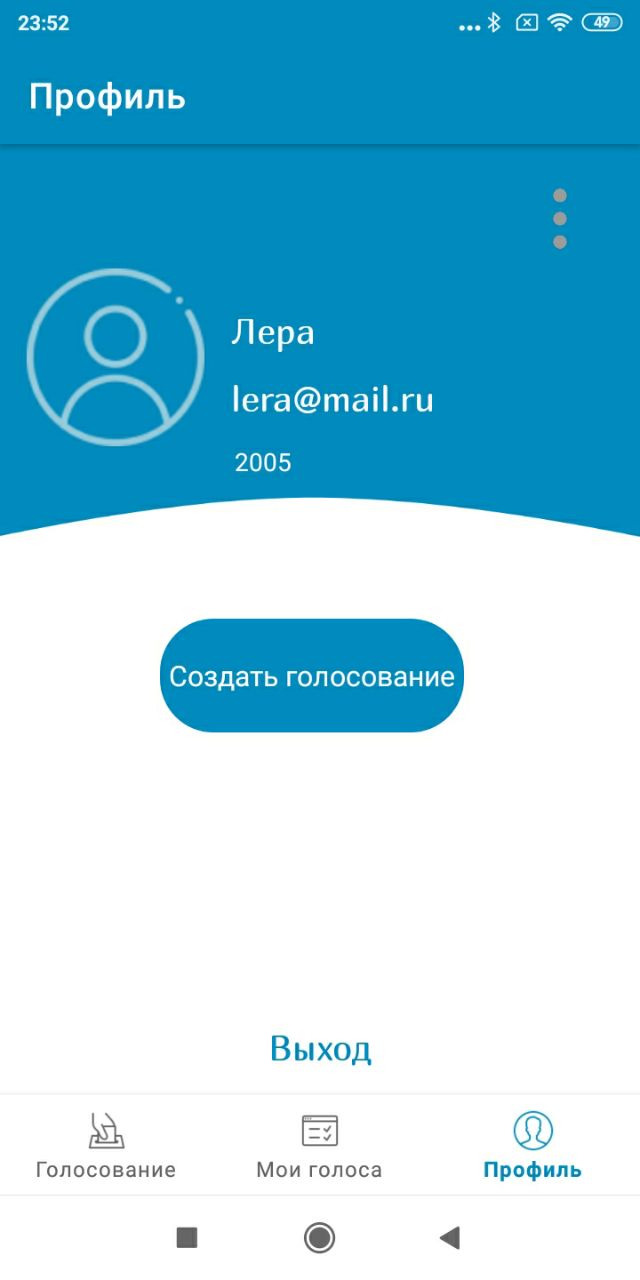


Рисунок 23 – Экран личной страницы пользователя



Рисунок 24 – Экран заполнения данных пользователя



Рисунок 25 – Экран создания собственного голосования



Рисунок 26 – Экран голосований

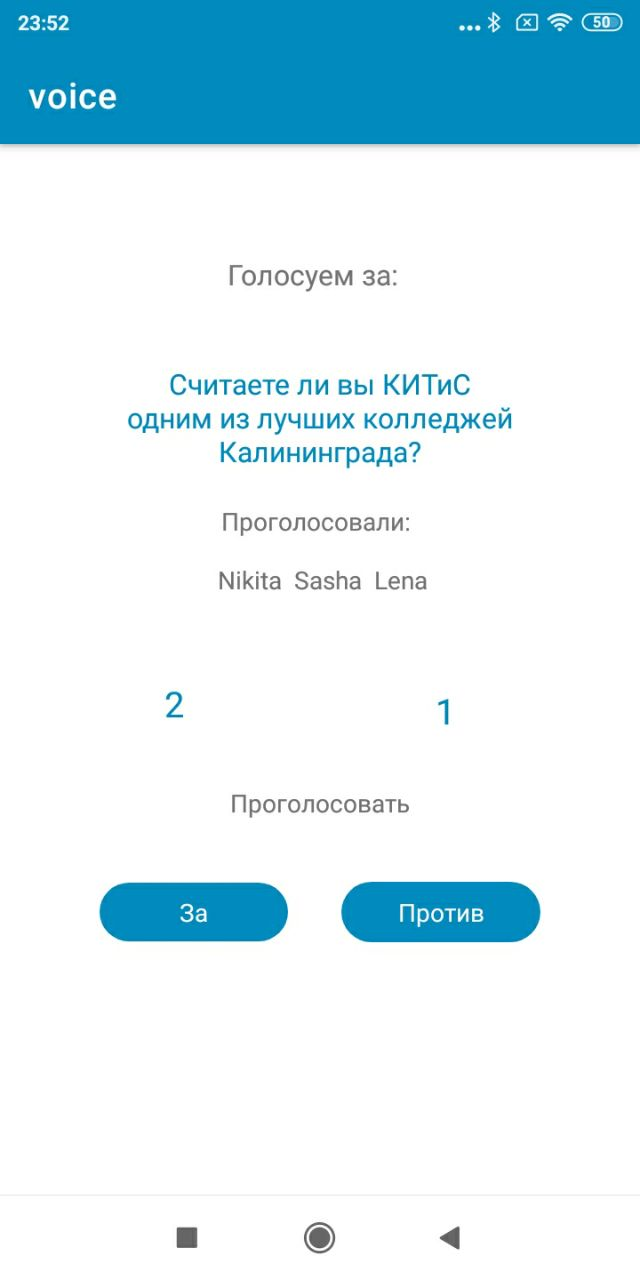


Рисунок 27 – Экран выбранного голосования пользователя

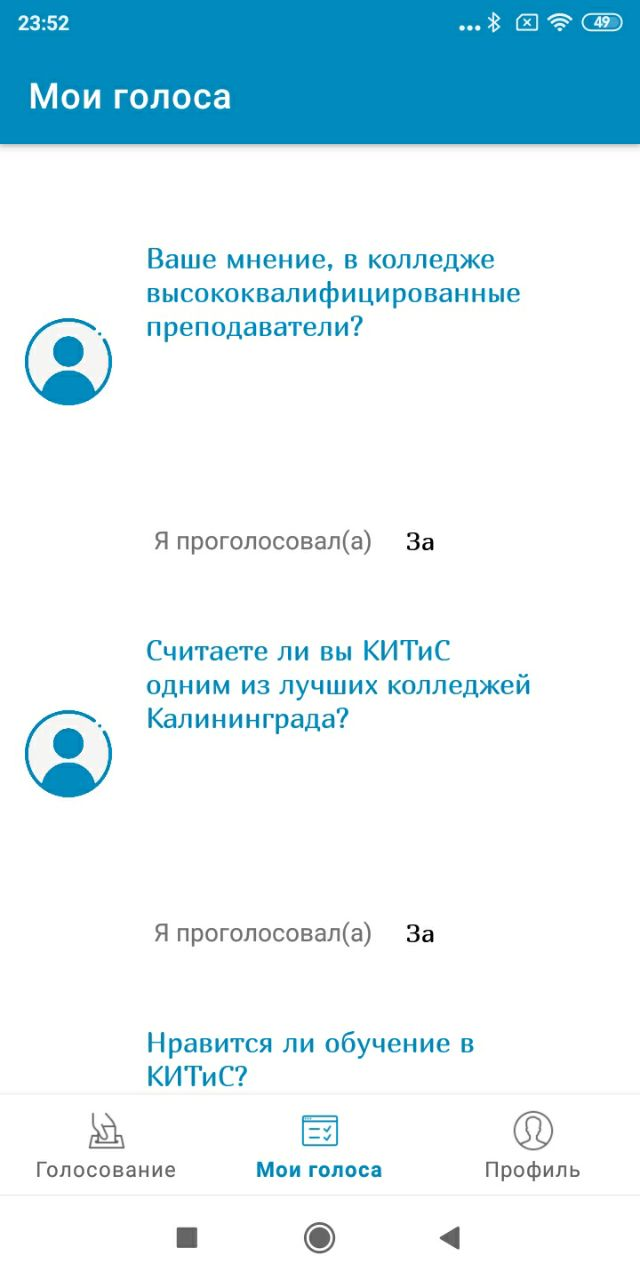


Рисунок 28 – Экран «Мои голоса» пользователя

Описание функционала:

Первый экран – заставка при входе в приложении.

Второй экран – предоставляется авторизация пользователя, который уже зарегистрирован в системе.

Третий экран – предназначен для регистрации пользователя, который первый раз в системе.

Четвертый экран – профиль пользователя, на котором отображена краткая информация о нем, а также выход из приложения.

Пятый экран – меню, где пользователь может заполнить или изменить информацию о себе.

Шестой экран – создание своего голосования и его название.

Седьмой экран – раздел голосования, где пользователи могут выбрать один из опросов и проголосовать в нем.

Восьмой экран – раздел голосования, где пользователь может проголосовать в определенном опросе за или против.

Девятый экран – раздел, в котором отображаются голосования, в котором пользователь принял участие.

* 1. Разработка логики работы приложения. Схема взаимодействия компонентов проекта

Схема представлена на рисунке 29.

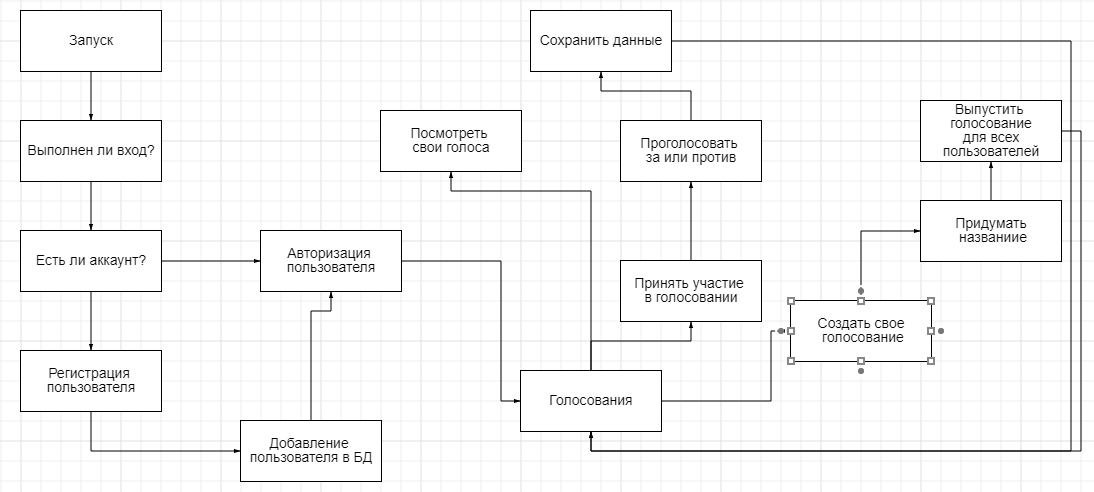


Рисунок 29 – Схема взаимодействия компонентов проекта

При запуске приложения пользователь попадает на главную страницу приложения. Если пользователь отсутствует в системе, то необходимо зарегистрироваться. После регистрации, пользователи добавляются в базу данных приложения в таблицу «User». Далее, зарегистрированный пользователь может использовать свой логин и пароль для входа.

После авторизации в приложении пользователь попадает в свой профиль.

При переходе пользователя в раздел голосования, он может выбрать понравившиеся вопрос и проголосовать за него. Все голоса сохраняются и хранятся в базе данных.

Когда пользователь проголосовал за какой-либо опрос, во вкладки мои голоса он может увидеть то, за что отдал свой голос.

При желании, пользователь может создать свое голосование, которое также сохранится в базу данных, и другие пользователи могут принять участие в опросе.

* 1. Описание переменных, компонентов, классов и подпрограмм

Структура проекта включает в себя такие классы, как:

1. Класс «CreateVoiceActivity» представляет собой активность для создания голосовых записей в приложении.

Переменные и методы:

* «mAuth»: экземпляр FirebaseAuth, который используется для аутентификации пользователей;
* «db»: экземпляр FirebaseFirestore, который представляет базу данных Firestore для работы с данными;
* «nameEdit»: EditText для ввода имени голосовой записи;
* «voice1Edit» и «voice2Edit»: EditText для ввода дополнительных данных голосовой записи;
* «creatButton»: TextView, представляющий кнопку для создания голосовой записи;
* «onCreate(Bundle savedInstanceState)»: метод жизненного цикла активности, вызываемый при ее создании;
* «setVoiceFirebase(String name)»: метод для сохранения голосовой записи в базу данных Firestore;
* «createTransactionID()»: метод для создания уникального идентификатора транзакции, который используется в качестве имени документа в Firestore.

1. Класс «SplashActivity» представляет собой отображение заставки при запуске приложения.

Переменные и методы:

* «aUser»: экземпляр класса FirebaseUser для представления текущего пользователя;
* «onCreate(Bundle savedInstanceState)»: метод жизненного цикла, вызываемый при создании активности;
* «isNetworkAvailable(Context context)»: метод для проверки доступности сети;

1. Класс «RegistrUserActivity» также представляет собой активность в приложении, ответственную за регистрацию пользователей.

Переменные и методы:

* «emailEditText», «nameEditText», «passwordEditText»: EditText для ввода адреса электронной почты, имени пользователя и пароля соответственно;
* «registrTextView»: TextView для перехода к экрану аутентификации;
* «nextImageView»: ImageView, представляющий кнопку для завершения регистрации;
* «mAuth»: экземпляр FirebaseAuth для аутентификации пользователей;
* «db»: экземпляр FirebaseFirestore для работы с базой данных Firestore;
* «user»: текущий пользователь Firebase;
* «onCreate(Bundle savedInstanceState)»: метод жизненного цикла активности, вызываемый при ее создании.;
* «createAccount(String email, String password, String name)»: Метод для создания аккаунта пользователя с помощью адреса электронной почты и пароля. После успешного создания аккаунта информация о пользователе сохраняется в базе данных Firestore;
* «setUserInfo(String uid, String email, String name)»: метод для сохранения информации о пользователе в коллекцию «users» базы данных Firestore. Здесь сохраняются адрес электронной почты и имя пользователя.

1. Класс «GivingVoiceActivity» представляет собой функционал для голосования, где пользователь может отдать свой голос за или против чего-то.

Переменные и методы:

* «giving», «butZa», «butProtiv», «usersVoice», «usersZa», «usersProtiv»: переменные представляют собой элементы пользовательского интерфейса, с которыми работает активность;
* «mAuth»: экземпляр класса FirebaseAuth для аутентификации Firebase;
* «db»: экземпляр класса FirebaseFirestore для взаимодействия с базой данных Firestore;
* «user»: предположительно, объект FirebaseUser, представляющий текущего пользователя;
* «userInfo»: возможно, это объект, содержащий информацию о пользователе, такую как имя;
* «onCreate(Bundle savedInstanceState)»: стандартный метод жизненного цикла Android, вызываемый при создании активности;
* «setVoiceInfo(String uid, String name, String field)»: метод для обновления данных в коллекции Firestore «voice». Принимает UID, имя и поле для обновления. Использует транзакции для обновления массивов данных в документе Firestore;
* «setMyVoiceFirebase(String name, String voice)»: метод для добавления записи о голосе пользователя в коллекцию Firestore «myvoice». Создает новый документ с уникальным идентификатором транзакции и информацией о голосе пользователя;
* «createTransactionID()»: метод для создания уникального идентификатора транзакции в формате UUID.

1. Класс «AuthActivity» представляет собой функционал для аутентификации пользователей в приложении. Пользователь может войти существующим аккаунтом, используя электронную почту и пароль, или перейти к процессу регистрации нового аккаунта.

Переменные и методы:

* «emailEditText», «passwordEditText», «nextImageView», «registrTextVie», «fogotPassTextView»: эти переменные представляют собой элементы пользовательского интерфейса, такие как текстовые поля, изображения и текстовые представления;
* «mAuth»: экземпляр класса FirebaseAuth для аутентификации Firebase;
* «aUser»: экземпляр класса FirebaseUser, представляющий текущего пользователя Firebase;
* «db»: экземпляр класса FirebaseFirestore для взаимодействия с базой данных Firestore;
* «onCreate(Bundle savedInstanceState)»: стандартный метод жизненного цикла Android, вызываемый при создании активности;
* «signIn(String email, String password)»: метод для входа пользователя с использованием электронной почты и пароля;
* «sendEmailVerification()»: метод для отправки подтверждения по электронной почте. Отправляет письмо с подтверждением электронной почты текущему пользователю Firebase.

1. Класс «ProfileViewModel» представляет собой часть архитектуры MVVM и используется для хранения и обработки данных, связанных с профилем пользователя.

Переменные и методы:

* «mText»: экземпляр класса MutableLiveData, хранящий текстовое значение
* «ProfileViewModel()»: конструктор класса ProfileViewModel, инициализирующий переменную mText и устанавливающий начальное значение;
* «getText()»: метод, возвращающий экземпляр класса LiveData, содержащий текстовое значение.

1. Класс «ProfileFragment» отвечает за отображение профиля пользователя и управление некоторыми действиями, такими как выход из аккаунта.

Переменные и методы:

* «binding»: экземпляр класса «FragmentProfileBinding», используемый для связывания макета фрагмента;
* «mText»: экземпляр класса «TextView», для отображения текста на пользовательском интерфейсе;
* «aUser»: экземпляр класса «FirebaseUser», представляющий текущего пользователя Firebase;
* «mAuth»: экземпляр класса «FirebaseAuth» для аутентификации Firebase;
* «db»: экземпляр класса «FirebaseFirestore» для взаимодействия с базой данных Firestore;
* «onCreateView()»: метод, вызываемый при создании представления фрагмента. Здесь происходит инициализация пользовательского интерфейса, установка слушателей для кнопок и другие операции;
* «onDestroyView()»: метод, вызываемый при уничтожении представления фрагмента.

1. Класс «MyVoiceViewModel» является частью архитектуры MVVM и используется для хранения и обработки данных.

Переменные и методы:

* «mText»: экземпляр класса «MutableLiveData», который хранит текстовое значение
* «MyVoiceViewModel()»: конструктор класса «MyVoiceViewModel», инициализирующий переменную «mText» и устанавливающий начальное значение;
* «getText()»: метод, возвращающий экземпляр класса «LiveData», содержащий текстовое значение.

1. Класс «MyVoiceFragment» отвечает за отображение списка элементов, связанных с голосом пользователя.

Переменные и методы:

* «binding»: экземпляр класса «FragmentMyvoiceBinding», используемый для связывания макета фрагмента;
* «onCreateView()»: метод, вызываемый при создании представления фрагмента;
* «onDestroyView()»: метод, вызываемый при уничтожении представления фрагмента. Здесь происходит очистка переменной «binding».

1. Класс «HomeViewModel» подобно «MyVoiceViewModel», также является частью архитектуры MVVM и используется для хранения и обработки данных.

Переменные и методы:

* «mText»: экземпляр класса «MutableLiveData», который хранит текстовое значение;
* «HomeViewModel()»: конструктор класса «HomeViewModel», инициализирующий переменную «mText» и устанавливающий начальное значение;
* «getText()»: метод, возвращающий экземпляр класса «LiveData», содержащий текстовое значение.

1. Класс «HomeFragment» отвечает за отображение списка элементов, связанных с голосом пользователя.

Переменные и методы:

* «binding»: экземпляр класса «FragmentHomeBinding», используемый для связывания макета фрагмента;
* «onCreateView()»: метод, вызываемый при создании представления фрагмента;
* «onDestroyView()»: метод, вызываемый при уничтожении представления фрагмента. Здесь происходит очистка переменной «binding».

1. Класс «VoiceInfo» представляет собой модель данных для хранения информации о голосе. Он содержит различные переменные для хранения имени голоса, информации о пользователях, списков элементов и голосов «за» и «против».

Переменные и методы:

* «name»: строка, хранящая имя голоса;
* «users»: строка, хранящая информацию о пользователях;
* «item»: список строк, представляющий элементы;
* «za»: список строк, представляющий голоса «за»;
* «protiv»: список строк, представляющий голоса «против»;
* «id»: строка, представляющая идентификатор голоса;
* «setName(String name)»: метод для установки имени голоса;
* «getName()»: метод для получения имени голоса;
* «setItem(List<String> item)»: метод для установки списка элементов;
* «getItem()»: метод для получения списка элементов;
* «setUsers(String users)»: метод для установки информации о пользователях;
* «getUsers()»: метод для получения информации о пользователях;
* «setZa(List<String> za)»: метод для установки списка голосов «за»;
* «getZa()»: Метод для получения списка голосов «за»;
* «setProtiv(List<String> protiv)»: метод для установки списка голосов «против»;
* «getProtiv()»: метод для получения списка голосов «против»;
* «setId(String id)»: метод для установки идентификатора голоса;
* «getId()»: метод для получения идентификатора голоса.

1. Класс «user\_info» служит для хранения информации о пользователе, включая его имя, адрес электронной почты, идентификатор.

Переменные и методы:

* «email»: строка, хранящая адрес электронной почты пользователя;
* «name»: строка, содержащая имя пользователя;
* «id»: строка, представляющая идентификатор пользователя;
* «year»: строка, хранящая информацию о годе пользователя;
* «getEmail()»: метод для получения адреса электронной почты пользователя;
* «setEmail(String email)»: метод для установки адреса электронной почты пользователя;
* «getName()»: метод для получения имени пользователя;
* «setName(String name)»: метод для установки имени пользователя;
* «getId()»: метод для получения идентификатора пользователя;
* «setId(String id)»: метод для установки идентификатора пользователя;
* «getYear()»: метод для получения информации о годе пользователя;
* «setYear(String year)»: метод для установки информации о годе пользователя.

1. Класс «profile\_info» используется для хранения информации о профиле пользователя и голосов, связанных с этим профилем.

Переменные и методы:

* «dataObject»: приватная статическая переменная класса, используемая для хранения экземпляра «profile\_info»;
* «mInfos»: список объектов класса «VoiceInfo», хранящий информацию о голосах;
* «myvoiceInfos»: список объектов класса «MyvoiceInfo», предположительно, хранящий информацию о голосах пользователя;
* «my\_user\_info»: объект класса «user\_info», хранящий информацию о пользователе;
* «getVoiceInfos()»: метод для получения списка объектов «VoiceInfo»;
* «setVoiceInfos(ArrayList<VoiceInfo> mapMarketInfos)»: метод для установки списка объектов «VoiceInfo»;
* «getMyvoice()»: метод для получения списка объектов «MyvoiceInfo»;
* «setMyvoice(ArrayList<MyvoiceInfo> Infos)»: метод для установки списка объектов «MyvoiceInfo»;
* «getMy\_user\_info()»: метод для получения объекта «user\_info», содержащего информацию о пользователе;
* «setMy\_user\_info(user\_info my\_user\_info)»: метод для установки объекта «user\_info», содержащего информацию о пользователе;
* «getInstance()»: статический метод, возвращающий единственный экземпляр класса «profile\_info». Если экземпляр еще не создан, создается новый экземпляр и возвращается.

1. Класс «Myvoiceinfo» используется для хранения информации о голосе конкретного пользователя.

Переменные и методы:

* «name»: строка, содержащая имя пользователя, связанное с голосом;
* «voice»: строка, представляющая голос пользователя;
* «id»: строка, представляющая идентификатор голоса;
* «setId(String id)»: метод для установки идентификатора голоса;
* «getId()»: Метод для получения идентификатора голоса;
* «setName(String name)»: метод для установки имени пользователя, связанного с голосом;
* «getName()»: метод для получения имени пользователя, связанного с голосом;
* «setVoice(String voice)»: метод для установки голоса пользователя;
* «getVoice()»: метод для получения голоса пользователя.

1. Класс «FireBaseLoad» представляет собой управление загрузкой данных из Firebase.

Переменные и методы:

* «context»: контекст приложения Android;
* «user»: объект класса «user\_info», используемый для хранения информации о пользователе;
* «voiceInfos»: список объектов класса «VoiceInfo», содержащий информацию о голосах;
* «myInfos»: список объектов класса «MyvoiceInfo», содержащий информацию о голосе конкретного пользователя;
* «mAuth`: объект класса «FirebaseAuth», используемый для аутентификации пользователя;
* «userDB»: объект класса «FirebaseUser», представляющий текущего пользователя Firebase;
* «db»: объект класса «FirebaseFirestore», представляющий базу данных Firestore;
* «profileInfo»: объект класса «profile\_info», используемый для доступа к информации о профиле пользователя;
* «FireBaseLoad(Context context)»: конструктор класса, принимающий контекст приложения Android и инициализирующий его переменную «context»;
* «getUserInfo(String UserToken)»: метод для получения информации о пользователе из Firestore. Возвращает объект класса «user\_info», содержащий информацию о пользователе;
* «getVoiceInfo()»: метод для получения информации о голосах из Firestore и сохранения ее в списке `voiceInfos`;
* «getMyVoiceInfo()»: метод для получения информации о голосе конкретного пользователя из Firestore и сохранения ее в списке «myInfos»;
* «stringToArrList(String s)»: метод для преобразования строки в список строк;
* «notNullStr(QueryDocumentSnapshot document, String tokenName)»: метод для получения значения из поля документа Firestore. Если поле не существует, возвращает пустую строку.

1. Класс «AdapterVoice» предназначен для связывания данных о голосах с представлением пользовательского интерфейса в «RecyclerView».

Переменные и методы:

* «onClickListener»: интерфейсный слушатель для обработки кликов на элементах списка голосов;
* «inflater»: объект «LayoutInflater», используемый для создания представлений элементов списка из макета XML;
* «voiceInfoList»: список объектов «VoiceInfo», содержащий информацию о голосах;
* «AdapterVoice(OnStateClickListener onClickListener, Context context, List<VoiceInfo> voiceInfoList)»: конструктор класса, принимающий слушатель кликов, контекст приложения и список объектов «VoiceInfo»;
* «onCreateViewHolder(ViewGroup parent, int viewType)»: метод вызывается, когда «RecyclerView» требует нового представления для элемента списка;
* «getItemCount()»: метод возвращает общее количество элементов в списке голосов.

1. Класс «ViewHolder» предоставляет связь между данными о голосах и их представлением в «RecyclerView».

Переменные и методы:

* «ViewHolder(View view)»: конструктор класса, принимающий представление элемента списка и инициализирующий его компоненты.

1. Тестирование и установка приложения

Основная цель тестирования - убедиться в корректной работе приложения, что ожидаемые результаты соответствуют фактическим, требования выполняются и нет критических ошибок. Тестирование приложения оценивает его функциональность, безопасность и удобство использования, а также проверяет соответствие требованиям, как вручную, так и с использованием специальных инструментов.

Анализ процесса установки:

Для проверки процесса установки необходимо сначала создать файл APK приложения, показанный на рисунке 30.



Рисунок 30 – Создание apk-файла приложения

После создания APK-файла нужно его запустить на устройстве для проведения тестирования приложения. Запуск APK-файла инициирует процесс установки приложения на устройство. Установка приложение отображена на рисунке 31.

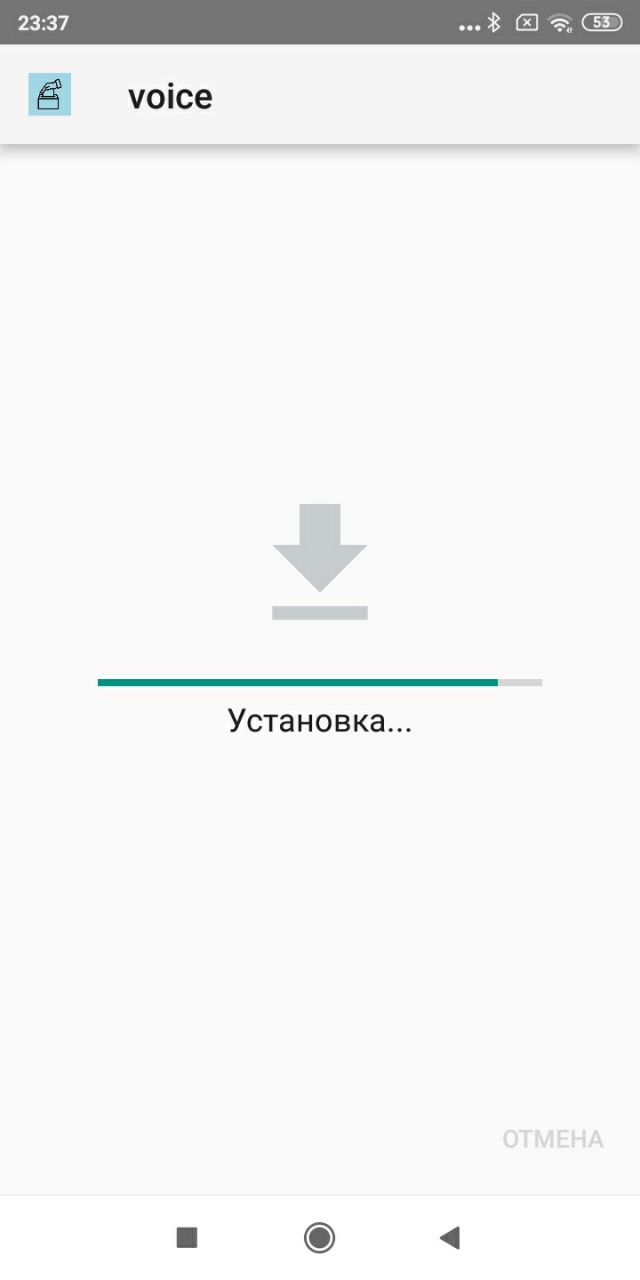


Рисунок 31 – Установка apk-файла приложения

После установки apk-файла приложение запускается и работает без ошибок.

Установка приложения из среды разработки «Android Studio» версии Iguana 2023.2.1 Patch 2 на мобильное устройство Android или планшет происходит следующим образом:

* Открываем файл проекта и запускаем проект через кнопку «Run» на верхней панели инструментов, как показано на рисунке 32.

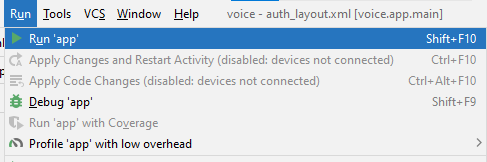


Рисунок 32 – Запуск проекта через кнопку «Run»

* Затем происходит компилирование проекта на нижней панели инструментов, как показано на рисунке 33.



Рисунок 33 – Компилирование проекта

Если проект успешно скомпилировался, то он появляется на рабочем столе мобильного устройства, представленный на рисунке 34.

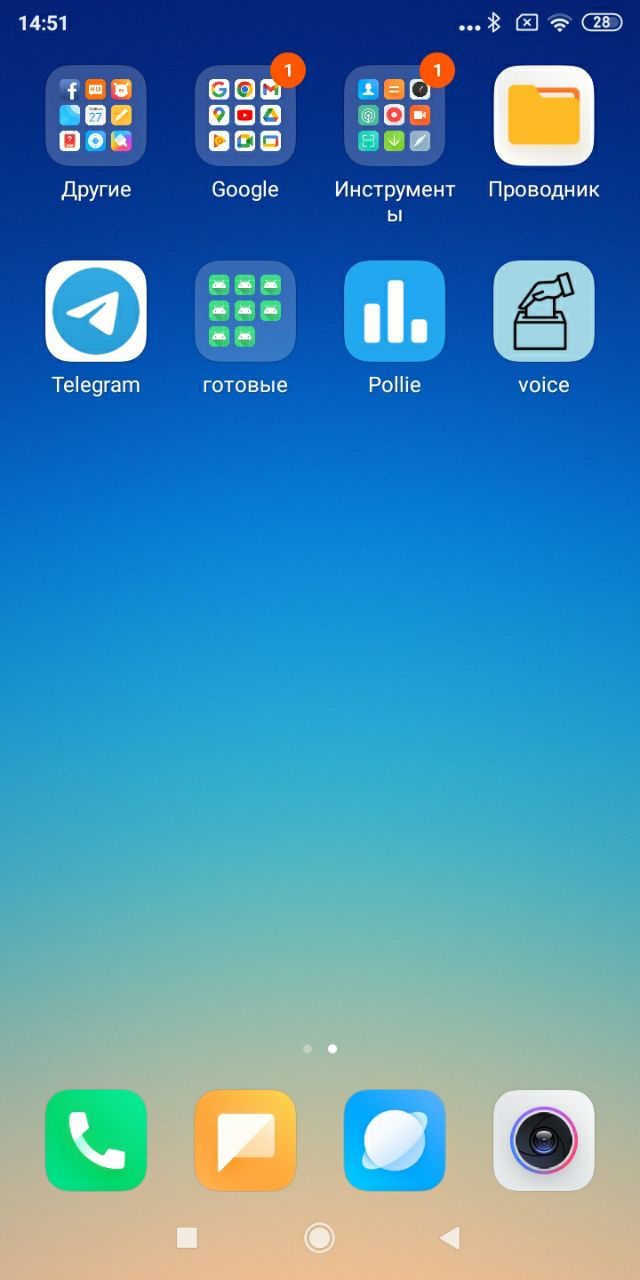


Рисунок 34 – Установленное приложение на рабочем столе мобильного устройства

# Заключение

В ходе разработки мобильного приложения для проведения процедуры голосования была реализована программа, предназначенная для голосования в разных сферах жизнедеятельности. Приложение предоставляет голосовать и создавать собственное голосование на разные темы. Данное приложение можно использовать не только для голосования в выборах, но также и в учебных заведениях

Важным аспектом разработанного приложения является его простота использования и интуитивно понятный интерфейс. Пользователи могут легко найти необходимую информацию, выполнить нужные действия.

# Список использованной литературы

1. Бьюли А. Изучаем SQL / Бьюли А. // Символ-Плюс – 2016 – Т. 1-№100 – С. 312
2. Шилдт Г. Java Руководство для начинающих / Шилдт Г. // Диалектика – 2018 – Т. 1-№100-803
3. Эккель Б. Философия Java / Эккель Б. // Издательский дом ПИТЕР – 2019 – Т. 1-№120 – С.1168
4. <https://developer.android.com/studio> - Официальный сайт «Android Studio».
5. <https://firebase.google.com/> - Официальный сайт «Firebase».
6. <https://flutter.dev/> - Официальный сайт «Flutter»
7. <https://www.figma.com/files/recents-and-sharing/recently-viewed?fuid=1273997628875279247> – Официальный сайт «Figma».
8. <https://www.mychart.org/> - Официальный сайт «MyChart».
9. <https://www.mysql.com/> - Официальный сайт «MySQL».
10. <https://javarush.com/> - портал JavaRush: язык программирования Java
11. <https://www.sqlite.org/> - Официальный сайт «SQLite».
12. <https://itproger.com/course/android-firebase/2> - Подключение базы данных Firebase и библиотек к проекту.
13. <https://itproger.com/course/android-chat> - Создание чат программы на Android.

# Приложение А. Листинг программы

Код класса MainActivity

package com.boom.voice;  
import android.os.Bundle;  
import com.google.android.material.bottomnavigation.BottomNavigationView;  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.navigation.NavController;  
import androidx.navigation.Navigation;  
import androidx.navigation.ui.AppBarConfiguration;  
import androidx.navigation.ui.NavigationUI;  
import com.boom.voice.databinding.ActivityMainBinding;  
// Главная активность приложения  
public class MainActivity extends AppCompatActivity {  
 // Привязка для макета активности  
 private ActivityMainBinding binding;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 // Настройка привязки макета активности  
 binding = ActivityMainBinding.inflate(getLayoutInflater());  
 setContentView(binding.getRoot());  
 // Настройка нижней навигации  
 BottomNavigationView navView = findViewById(R.id.nav\_view);  
 // Определение конфигурации панели действий  
 AppBarConfiguration appBarConfiguration = new AppBarConfiguration.Builder(  
 R.id.navigation\_home, R.id.navigation\_myvoice, R.id.navigation\_profile)  
 .build();  
 NavController navController = Navigation.findNavController(this, R.id.nav\_host\_fragment\_activity\_main);  
 NavigationUI.setupActionBarWithNavController(this, navController, appBarConfiguration);  
 NavigationUI.setupWithNavController(binding.navView, navController);  
 }  
}

Код класса SplashActivity

package com.boom.voice.ui.ui;  
import static android.content.ContentValues.TAG;  
import android.content.Context;  
import android.content.Intent;  
import android.net.ConnectivityManager;  
import android.os.Bundle;  
import android.os.Handler;  
import android.util.Log;  
import androidx.appcompat.app.AppCompatActivity;  
import com.boom.voice.MainActivity;  
import com.boom.voice.R;  
import com.google.firebase.auth.FirebaseAuth;  
import com.google.firebase.auth.FirebaseUser;  
import com.boom.voice.ui.data.FireBaseLoad;  
import com.boom.voice.ui.data.profile\_info;  
import com.boom.voice.ui.data.user\_info;  
// Активность для отображения заставки при запуске приложения  
public class SplashActivity extends AppCompatActivity {  
 // Пользователь Firebase  
 private FirebaseUser aUser;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.splash\_layout); // Установка макета заставки  
 // Получение текущего пользователя Firebase  
 aUser = FirebaseAuth.getInstance().getCurrentUser();  
 // Проверка, авторизован ли пользователь  
 if (aUser != null) {  
 // Если пользователь авторизован  
 String uid = aUser.getUid(); // Получение идентификатора пользователя  
 FireBaseLoad data = new FireBaseLoad(this); // Инициализация загрузки данных из Firebase  
 data.getUserInfo(uid); // Получение информации о пользователе  
 data.getVoiceInfo(); // Получение информации о голосованиях  
 data.getMyVoiceInfo(); // Получение информации о пользовательских голосованиях  
 // Проверка доступности сети  
 if (isNetworkAvailable(this)) {  
 // Если сеть доступна, переход к главной активности через некоторое время  
 new Handler().postDelayed(new Runnable() {  
 @Override  
 public void run() {  
 // Получение информации о пользователе  
 user\_info userInfo = profile\_info.getInstance().getMy\_user\_info();  
 Intent i = new Intent(SplashActivity.this, MainActivity.class); // Создание намерения для перехода к главной активности  
 startActivity(i); // Запуск главной активности  
 }  
 }, 4000); // Задержка перед переходом (в миллисекундах)  
 }  
 } else {  
 // Если пользователь не авторизован, переход к активности авторизации  
 Intent i = new Intent(SplashActivity.this, AuthActivity.class);  
 startActivity(i);  
 Log.d(TAG, "onAuthStateChanged:signed\_out"); // Логирование выхода пользователя  
 }  
 }  
 // Метод для проверки доступности сети  
 public boolean isNetworkAvailable(Context context) {  
 ConnectivityManager connectivityManager = ((ConnectivityManager) context.getSystemService(Context.CONNECTIVITY\_SERVICE));  
 return connectivityManager.getActiveNetworkInfo() != null && connectivityManager.getActiveNetworkInfo().isConnected();  
 }  
}

Код класса RegistrUserActivity

package com.boom.voice.ui.ui;  
import static android.content.ContentValues.TAG;  
import android.content.Intent;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.View;  
import android.widget.EditText;  
import android.widget.ImageView;  
import android.widget.TextView;  
import android.widget.Toast;  
import androidx.annotation.NonNull;  
import androidx.appcompat.app.AppCompatActivity;  
import com.google.android.gms.tasks.OnCompleteListener;  
import com.google.android.gms.tasks.Task;  
import com.google.firebase.auth.AuthResult;  
import com.google.firebase.auth.FirebaseAuth;  
import com.google.firebase.auth.FirebaseUser;  
import com.google.firebase.firestore.\*;  
import java.util.HashMap;  
import java.util.Map;  
import com.boom.voice.R;  
// Активность для регистрации нового пользователя  
public class RegistrUserActivity extends AppCompatActivity {  
 // Поля для ввода данных  
 EditText emailEditText, nameEditText, passwordEditText;  
 // Кнопки и текстовое поле  
 TextView registrTextView;  
 ImageView nextImageView;  
 // Firebase  
 private FirebaseAuth mAuth;  
 FirebaseFirestore db;  
 FirebaseUser user;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.registr\_user\_layout); // Установка макета активности  
 // Инициализация элементов интерфейса  
 emailEditText = findViewById(R.id.emailEditText);  
 nameEditText = findViewById(R.id.nameEditText);  
 passwordEditText = findViewById(R.id.passwordEditText);  
 registrTextView = findViewById(R.id.registrTextView);  
 nextImageView = findViewById(R.id.nextImageView);  
 // Инициализация Firebase  
 mAuth = FirebaseAuth.getInstance();  
 db = FirebaseFirestore.getInstance();  
 user = mAuth.getCurrentUser();  
 // Обработчик клика по тексту для перехода к активности аутентификации  
 registrTextView.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent intent = new Intent(RegistrUserActivity.this, AuthActivity.class);  
 startActivity(intent);  
 finish();  
 }  
 });  
 // Обработчик клика по кнопке для регистрации пользователя  
 nextImageView.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 // Получение введенных пользователем данных  
 String email = emailEditText.getText().toString();  
 String name = nameEditText.getText().toString();  
 String password = passwordEditText.getText().toString();  
 // Проверка на заполненность всех полей  
 if (!email.equals("") && !name.equals("") && !password.equals("")) {  
 // Создание аккаунта пользователя  
 createAccount(email, password, name);  
 Intent intent = new Intent(RegistrUserActivity.this, SplashActivity.class);  
 startActivity(intent);  
 finish();  
 }  
 }  
 });  
 }  
 // Метод для создания аккаунта пользователя  
 private void createAccount(String email, String password, String name) {  
 mAuth.createUserWithEmailAndPassword(email, password)  
 .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if (task.isSuccessful()) {  
 // Успешная регистрация пользователя  
 Log.d(TAG, "createUserWithEmail:success");  
 // Получение идентификатора пользователя  
 String uid = user.getUid();  
 Log.d("UID", uid);  
 // Установка информации о пользователе  
 setUserInfo(uid, email, name);  
 Intent intent = new Intent(RegistrUserActivity.this, SplashActivity.class);  
 startActivity(intent);  
 finish();  
 } else {  
 // Ошибка при регистрации пользователя  
 Log.w(TAG, "createUserWithEmail:failure", task.getException());  
 Toast.makeText(RegistrUserActivity.this, "Error" + task.getException().getMessage(), Toast.LENGTH\_LONG).show();  
 Toast.makeText(RegistrUserActivity.this, "Authentication failed.", Toast.LENGTH\_SHORT).show();  
 }  
 }  
 });  
 }  
 // Метод для установки информации о пользователе в базу данных Firestore  
 public void setUserInfo(String uid, String email, String name) {  
 CollectionReference users = db.collection("users");  
 Map<String, Object> data = new HashMap<>();  
 data.put("email", email);  
 data.put("name", name);  
 users.document(uid).set(data);  
 }  
}

Код класса ProfileRenameActivity

package com.boom.voice.ui.ui;  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.EditText;  
import android.widget.TextView;  
import com.boom.voice.R;  
import com.google.firebase.auth.FirebaseAuth;  
import com.google.firebase.auth.FirebaseUser;  
import com.google.firebase.firestore.CollectionReference;  
import com.google.firebase.firestore.FirebaseFirestore;  
import java.util.HashMap;  
import java.util.Map;  
// Активность для изменения профиля пользователя  
public class ProfileRenameActivity extends AppCompatActivity {  
 // Элементы интерфейса  
 TextView butSave;  
 EditText nameEdit, yearEdit, emailEdit;  
 // Firebase  
 private FirebaseAuth mAuth;  
 FirebaseFirestore db;  
 FirebaseUser user;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_profile\_rename); // Установка макета активности  
 // Инициализация элементов интерфейса  
 butSave = findViewById(R.id.saveName);  
 nameEdit = findViewById(R.id.nameEdit);  
 yearEdit = findViewById(R.id.yearEdit);  
 emailEdit = findViewById(R.id.emailEdit);  
 // Инициализация Firebase  
 mAuth = FirebaseAuth.getInstance();  
 db = FirebaseFirestore.getInstance();  
 user = mAuth.getCurrentUser();  
 // Обработчик клика по кнопке для сохранения изменений профиля  
 butSave.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 // Получение введенных пользователем данных  
 String name = nameEdit.getText().toString();  
 String year = yearEdit.getText().toString();  
 String email = emailEdit.getText().toString();  
 // Проверка на заполненность всех полей  
 if (!name.equals("") && !year.equals("") && !email.equals("")) {  
 // Получение идентификатора пользователя  
 String uid = user.getUid();  
 // Установка информации о пользователе  
 setUserInfo(uid, year, name, email);  
 Intent intent = new Intent(ProfileRenameActivity.this, SplashActivity.class);  
 startActivity(intent);  
 finish();  
 }  
 }  
 });  
 }  
 // Метод для установки информации о пользователе в базу данных Firestore  
 public void setUserInfo(String uid, String year, String name, String email) {  
 CollectionReference users = db.collection("users");  
 Map<String, Object> data = new HashMap<>();  
 data.put("year", year);  
 data.put("name", name);  
 data.put("email", email);  
 users.document(uid).set(data);  
 }  
}

Код класса GivingVoiceActivity

package com.boom.voice.ui.ui;  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.TextView;  
import com.boom.voice.R;  
import com.boom.voice.ui.data.VoiceInfo;  
import com.boom.voice.ui.data.profile\_info;  
import com.boom.voice.ui.data.user\_info;  
import com.google.firebase.auth.FirebaseAuth;  
import com.google.firebase.auth.FirebaseUser;  
import com.google.firebase.firestore.CollectionReference;  
import com.google.firebase.firestore.FieldValue;  
import com.google.firebase.firestore.FirebaseFirestore;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import java.util.UUID;  
// Активность для голосования  
public class GivingVoiceActivity extends AppCompatActivity {  
 // Элементы интерфейса  
 TextView giving, butZa, butProtiv, usersVoice, usersZa, usersProtiv;  
 // Firebase  
 private FirebaseAuth mAuth;  
 FirebaseFirestore db;  
 FirebaseUser user;  
 user\_info userInfo;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_giving\_voice); // Установка макета активности  
 // Инициализация элементов интерфейса  
 giving = findViewById(R.id.giving);  
 butZa = findViewById(R.id.butZa);  
 butProtiv = findViewById(R.id.butProtiv);  
 usersVoice = findViewById(R.id.usersVoice);  
 usersZa = findViewById(R.id.usersZa);  
 usersProtiv = findViewById(R.id.usersProtiv);  
 // Инициализация Firebase  
 mAuth = FirebaseAuth.getInstance();  
 db = FirebaseFirestore.getInstance();  
 user = mAuth.getCurrentUser();  
 // Получение информации о текущем пользователе  
 userInfo = profile\_info.getInstance().getMy\_user\_info();  
 // Получение информации о выбранном элементе  
 Bundle arguments = getIntent().getExtras();  
 VoiceInfo voiceInfo;  
 if (arguments != null) {  
 voiceInfo = (VoiceInfo) arguments.getSerializable(VoiceInfo.class.getSimpleName());  
 // Установка названия элемента в заголовок  
 giving.setText(voiceInfo.getName());  
 // Получение информации о голосах  
 List<String> item = voiceInfo.getItem();  
 String text = "Проголосовали: \n \n";  
 for (int i = 0; i < item.size(); i++) {  
 text = text + " " + item.get(i);  
 }  
 usersVoice.setText(text);  
 usersProtiv.setText(String.valueOf(voiceInfo.getProtiv().size()));  
 if (voiceInfo.getZa().size() == 0 || voiceInfo.getZa().get(0).equals("")) {  
 usersZa.setText("0");  
 } else {  
 usersZa.setText(String.valueOf(voiceInfo.getZa().size()));  
 }  
 if (voiceInfo.getProtiv().size() == 0 || voiceInfo.getProtiv().get(0).equals("")) {  
 usersProtiv.setText("0");  
 } else {  
 usersProtiv.setText(String.valueOf(voiceInfo.getProtiv().size()));  
 }  
 // Обработчик клика по кнопке "За"  
 butZa.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 // Добавление голоса "За" и обновление базы данных  
 setVoiceInfo(voiceInfo.getId(), userInfo.getName(), "za");  
 try {  
 setMyVoiceFirebase(voiceInfo.getName(), "За");  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 // Переход к главному экрану  
 Intent intent = new Intent(GivingVoiceActivity.this, SplashActivity.class);  
 startActivity(intent);  
 finish();  
 }  
 });  
 // Обработчик клика по кнопке "Против"  
 butProtiv.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 // Добавление голоса "Против" и обновление базы данных  
 setVoiceInfo(voiceInfo.getId(), userInfo.getName(), "protiv");  
 try {  
 setMyVoiceFirebase(voiceInfo.getName(), "Против");  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 // Переход к главному экрану  
 Intent intent = new Intent(GivingVoiceActivity.this, SplashActivity.class);  
 startActivity(intent);  
 finish();  
 }  
 });  
 }  
 }  
 // Метод для добавления информации о голосе в базу данных Firestore  
 public void setVoiceInfo(String uid, String name, String field) {  
 CollectionReference cities = db.collection("voice");  
 final Map<String, Object> addUserToArrayMap = new HashMap<>();  
 addUserToArrayMap.put(field, FieldValue.arrayUnion(name));  
 final Map<String, Object> addUserToItem = new HashMap<>();  
 addUserToItem.put("item", FieldValue.arrayUnion(name));  
 cities.document(uid).update(addUserToArrayMap);  
 cities.document(uid).update(addUserToItem);  
 }  
 // Метод для добавления информации о голосе пользователя в базу данных Firestore  
 public void setMyVoiceFirebase(String name, String voice) throws Exception {  
 String uid = user.getUid();  
 CollectionReference cities = db.collection("myvoice");  
 Map<String, Object> data1 = new HashMap<>();  
 data1.put("id", uid);  
 data1.put("name", name);  
 data1.put("myvoice", voice);  
 cities.document(createTransactionID()).set(data1);  
 }  
 // Метод для создания уникального идентификатора  
 public String createTransactionID() throws Exception {  
 return UUID.randomUUID().toString().replaceAll("-", "").toUpperCase();  
 }  
}

Код класса CreatVoiceActivity

package com.boom.voice.ui.ui;  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.EditText;  
import android.widget.TextView;  
import com.boom.voice.R;  
import com.google.firebase.auth.FirebaseAuth;  
import com.google.firebase.firestore.CollectionReference;  
import com.google.firebase.firestore.FirebaseFirestore;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.UUID;  
// Активность для создания нового элемента для голосования  
public class CreatVoiceActivity extends AppCompatActivity {  
 // Firebase  
 private FirebaseAuth mAuth;  
 FirebaseFirestore db;  
 // Элементы интерфейса  
 EditText nameEdit;  
 TextView creatButton;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_creat\_voice); // Установка макета активности  
 // Инициализация элементов интерфейса  
 nameEdit = findViewById(R.id.idCreatVoice);  
 creatButton = findViewById(R.id.creatButton);  
 // Инициализация Firebase  
 mAuth = FirebaseAuth.getInstance();  
 db = FirebaseFirestore.getInstance();  
 // Обработчик клика по кнопке "Создать"  
 creatButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 String name = nameEdit.getText().toString(); // Получение названия элемента  
 if (!name.equals("")) { // Проверка наличия названия  
 try {  
 setVoiceFirebase(name); // Добавление нового элемента в базу данных  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 // Переход к главному экрану  
 Intent i = new Intent(CreatVoiceActivity.this, SplashActivity.class);  
 startActivity(i);  
 }  
 }  
 });  
 }  
 // Метод для добавления нового элемента в базу данных Firestore  
 public void setVoiceFirebase(String name) throws Exception {  
 CollectionReference cities = db.collection("voice"); // Ссылка на коллекцию "voice"  
 Map<String, Object> data1 = new HashMap<>();  
 data1.put("name", name); // Название элемента  
 cities.document(createTransactionID()).set(data1); // Добавление документа с уникальным идентификатором  
 }  
 // Метод для создания уникального идентификатора  
 public String createTransactionID() throws Exception {  
 return UUID.randomUUID().toString().replaceAll("-", "").toUpperCase(); // Генерация случайного UUID  
 }  
}

Код класса AuthActivity

package com.boom.voice.ui.ui;  
import static android.content.ContentValues.TAG;  
import android.content.Context;  
import android.content.Intent;  
import android.graphics.drawable.ColorDrawable;  
import android.net.ConnectivityManager;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.Gravity;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.WindowManager;  
import android.widget.EditText;  
import android.widget.ImageView;  
import android.widget.TextView;  
import android.widget.Toast;  
import androidx.annotation.NonNull;  
import androidx.appcompat.app.AlertDialog;  
import androidx.appcompat.app.AppCompatActivity;  
import com.boom.voice.MainActivity;  
import com.google.android.gms.tasks.OnCompleteListener;  
import com.google.android.gms.tasks.Task;  
import com.google.firebase.auth.AuthResult;  
import com.google.firebase.auth.FirebaseAuth;  
import com.google.firebase.auth.FirebaseUser;  
import com.google.firebase.firestore.\*;  
import com.boom.voice.R;  
import com.boom.voice.ui.data.profile\_info;  
import com.boom.voice.ui.data.user\_info;  
// Активность аутентификации  
public class AuthActivity extends AppCompatActivity {  
 // Элементы интерфейса  
 EditText emailEditText, passwordEditText;  
 ImageView nextImageView;  
 TextView registrTextView, fogotPassTextView;  
 // Firebase  
 private FirebaseAuth mAuth;  
 private FirebaseUser aUser;  
 FirebaseFirestore db;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.auth\_layout); // Установка макета активности  
 // Инициализация элементов интерфейса  
 emailEditText = findViewById(R.id.emailEditText);  
 passwordEditText = findViewById(R.id.passwordEditText);  
 nextImageView = findViewById(R.id.nextImageView);  
 registrTextView = findViewById(R.id.registrTextView);  
 fogotPassTextView = findViewById(R.id.fogotPassTextView);  
 final AlertDialog dialogBuilder = new AlertDialog.Builder(this).create();  
 dialogBuilder.getWindow().setBackgroundDrawable(new ColorDrawable(android.graphics.Color.TRANSPARENT));  
 LayoutInflater inflater = this.getLayoutInflater();  
 View dialogView = inflater.inflate(R.layout.popup\_choose\_user\_layout, null);  
 TextView user = dialogView.findViewById(R.id.user);  
 TextView cansel = dialogView.findViewById(R.id.cansel);  
 // Создание диалога выбора пользователя  
 // Инициализация Firebase  
 mAuth = FirebaseAuth.getInstance();  
 aUser = FirebaseAuth.getInstance().getCurrentUser();  
 db = FirebaseFirestore.getInstance();  
 // Проверка авторизации при запуске активности  
 if (aUser != null) {  
 // Пользователь вошел в систему  
 String uid = aUser.getUid();  
 // Проверка доступности сети  
 if (isNetworkAvailable(this)) {  
 // Получение информации о пользователе  
 getUserInfo(uid);  
 try {  
 Thread.sleep(2000); // Задержка для имитации загрузки данных  
 Intent i = new Intent(AuthActivity.this, MainActivity.class);  
 i.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK | Intent.FLAG\_ACTIVITY\_CLEAR\_TASK);  
 startActivity(i); // Переход к главному экрану  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 } else {  
 // Пользователь вышел из системы  
 Log.d(TAG, "onAuthStateChanged:signed\_out");  
 }  
 // Обработчик нажатия на кнопку "Отмена" в диалоговом окне  
 cansel.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 dialogBuilder.dismiss();  
 }  
 });  
 // Обработчик нажатия на текст "Пользователь" в диалоговом окне  
 user.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent intent = new Intent(AuthActivity.this, RegistrUserActivity.class);  
 startActivity(intent); // Переход к активности регистрации  
 finish();  
 }  
 });  
// Установка параметров для диалогового окна  
 WindowManager.LayoutParams wlmp = dialogBuilder.getWindow().getAttributes();  
 wlmp.gravity = Gravity.BOTTOM;  
 dialogBuilder.setView(dialogView);  
 // Обработчик нажатия на кнопку "Далее"  
 nextImageView.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 String email = emailEditText.getText().toString();  
 String password = passwordEditText.getText().toString();  
 if (!email.equals("") || !password.equals("")) {  
 signIn(email , password); // Вызов метода входа в систему  
 }  
 }  
 });  
 // Обработчик нажатия на текст "Регистрация"  
 registrTextView.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent i = new Intent(AuthActivity.this, RegistrUserActivity.class);  
 startActivity(i); // Переход к активности регистрации  
 }  
 });  
 }  
 // Метод для входа в систему по электронной почте и паролю  
 private void signIn(String email, String password) {  
 mAuth.signInWithEmailAndPassword(email, password)  
 .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if (task.isSuccessful()) {  
 // Успешный вход в систему  
 Log.d(TAG, "signInWithEmail:success");  
 FirebaseUser user = mAuth.getCurrentUser();  
 String uid = user.getUid();  
 String rr = mAuth.getUid();  
 getUserInfo(uid); // Получение информации о пользователе  
 Intent intent = new Intent(AuthActivity.this, SplashActivity.class);  
 startActivity(intent); // Переход к главному экрану  
 finish();  
 } else {  
 // Ошибка входа в систему  
 Log.w(TAG, "signInWithEmail:failure", task.getException());  
 Toast.makeText(AuthActivity.this, "Authentication failed.",  
 Toast.LENGTH\_SHORT).show();  
 }  
 }  
 });  
 }  
 // Метод для отправки подтверждения по электронной почте  
 private void sendEmailVerification() {  
 final FirebaseUser user = mAuth.getCurrentUser();  
 user.sendEmailVerification()  
 .addOnCompleteListener(this, new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task) {  
 // Письмо отправлено  
 }  
 });  
 }  
 // Метод для получения информации о пользователе из базы данных Firestore  
 public user\_info getUserInfo (String UserToken) {  
 user\_info userInfo = new user\_info();  
 profile\_info profileInfo = new profile\_info();  
 DocumentReference docRef = db.collection("users").document(UserToken);  
 docRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<DocumentSnapshot> task) {  
 if (task.isSuccessful()) {  
 DocumentSnapshot document = task.getResult();  
 if (document.exists()) {  
 Log.d(TAG, "DocumentSnapshot data: " + document.getData());  
 String email= notNullStrForUsers(document , "email" );  
 String name= notNullStrForUsers(document , "name" );  
 userInfo.setEmail(email);  
 userInfo.setName(name);  
 profileInfo.getInstance().setMy\_user\_info(userInfo);  
 } else {  
 Log.d(TAG, "No such document");  
 }  
 } else {  
 Log.d(TAG, "get failed with ", task.getException());  
 }  
 }  
 });  
 return userInfo;  
 }  
 // Метод для получения строки из документа Firestore  
 public String notNullStr (QueryDocumentSnapshot document ,String tokenName) {  
 String dd = "";  
 try {  
 dd = document.getData().get(tokenName).toString();  
 } catch (Exception e) {  
  
 }  
 return dd;  
 }  
 // Метод для получения строки из документа Firestore для пользователей  
 public String notNullStrForUsers (DocumentSnapshot document ,String tokenName) {  
 String dd = "";  
 try {  
 dd = document.getData().get(tokenName).toString();  
 } catch (Exception e) {  
  
 }  
 return dd;  
 }  
 // Проверка доступности сети  
 public boolean isNetworkAvailable(Context context) {  
 ConnectivityManager connectivityManager = ((ConnectivityManager) context.getSystemService(Context.CONNECTIVITY\_SERVICE));  
 return connectivityManager.getActiveNetworkInfo() != null && connectivityManager.getActiveNetworkInfo().isConnected();  
 }  
}

Код класса ProfileViewModel

package com.boom.voice.ui.profile;  
import androidx.lifecycle.LiveData;  
import androidx.lifecycle.MutableLiveData;  
import androidx.lifecycle.ViewModel;  
// ViewModel для фрагмента профиля, отображающий информацию о пользователе и предоставляющий возможность управления профилем.  
public class ProfileViewModel extends ViewModel {  
 // данные для текста на фрагменте  
 private final MutableLiveData<String> mText;  
 // Конструктор класса ProfileViewModel  
 public ProfileViewModel() {  
 // Инициализация живых данных  
 mText = new MutableLiveData<>();  
 // Установка значения по умолчанию  
 mText.setValue("This is notifications fragment");  
 }  
 // Получение данных для текста  
 public LiveData<String> getText() {  
 return mText;  
 }  
}

Код класса ProfileFragment

package com.boom.voice.ui.profile;  
import android.app.AlertDialog;  
import android.app.Dialog;  
import android.content.DialogInterface;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ImageView;  
import android.widget.TextView;  
import androidx.annotation.NonNull;  
import androidx.fragment.app.DialogFragment;  
import androidx.fragment.app.Fragment;  
import androidx.fragment.app.FragmentManager;  
import androidx.fragment.app.FragmentTransaction;  
import androidx.lifecycle.ViewModelProvider;  
import com.boom.voice.R;  
import com.boom.voice.databinding.FragmentProfileBinding;  
import com.boom.voice.ui.data.profile\_info;  
import com.boom.voice.ui.data.user\_info;  
import com.boom.voice.ui.ui.AuthActivity;  
import com.boom.voice.ui.ui.CreatVoiceActivity;  
import com.boom.voice.ui.ui.ProfileRenameActivity;  
import com.google.firebase.auth.FirebaseAuth;  
import com.google.firebase.auth.FirebaseUser;  
import com.google.firebase.firestore.FirebaseFirestore;  
// Фрагмент профиля пользователя, отображающий информацию о пользователе и предоставляющий возможность выхода из учетной записи.  
public class ProfileFragment extends Fragment {  
 // Привязка для макета фрагмента  
 private FragmentProfileBinding binding;  
 // Поля для работы с Firebase  
 TextView mText;  
 private FirebaseUser aUser;  
 private FirebaseAuth mAuth;  
 FirebaseFirestore db;  
 // Создание представления фрагмента  
 public View onCreateView(@NonNull LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
 // Получение экземпляра ViewModel для фрагмента  
 ProfileViewModel notificationsViewModel =  
 new ViewModelProvider(this).get(ProfileViewModel.class);  
 // Настройка привязки макета фрагмента  
 binding = FragmentProfileBinding.inflate(inflater, container, false);  
 View root = binding.getRoot();  
 // Получение элементов макета  
 TextView nameText = binding.nameText;  
 TextView emailText = binding.emailText;  
 TextView year = binding.yearProfile;  
 TextView exit = binding.exitText;  
 TextView creat = binding.creatVoice;  
 ImageView button = (ImageView)root.findViewById(R.id.profileRename);  
 // Инициализация Firebase  
 mAuth = FirebaseAuth.getInstance();  
 aUser = FirebaseAuth.getInstance().getCurrentUser();  
 db = FirebaseFirestore.getInstance();  
 // Получение информации о пользователе из профиля и установка значений в соответствующие текстовые поля  
 user\_info userInfo = profile\_info.getInstance().getMy\_user\_info();  
 nameText.setText(userInfo.getName());  
 emailText.setText(userInfo.getEmail());  
 year.setText(userInfo.getYear());  
 // Обработчик нажатия на кнопку выхода из учетной записи  
 exit.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 // Отображение диалогового окна подтверждения выхода  
 DialogFragmentExit myDialogFragment = new DialogFragmentExit();  
 FragmentManager manager = getFragmentManager();  
 FragmentTransaction transaction = manager.beginTransaction();  
 myDialogFragment.show(transaction, "dialog");  
 }  
 });  
 // Обработчик нажатия на кнопку создания нового голосования  
 creat.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent i = new Intent(getActivity(), CreatVoiceActivity.class);  
 startActivity(i);  
 }  
 });  
 // Обработчик нажатия на кнопку редактирования профиля  
 button.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent i = new Intent(getActivity(), ProfileRenameActivity.class);  
 startActivity(i);  
 }  
 });  
 return root;  
 }  
 // Уничтожение привязки макета фрагмента при уничтожении фрагмента  
 @Override  
 public void onDestroyView() {  
 super.onDestroyView();  
 binding = null;  
 }  
 // Вложенный класс для диалогового окна подтверждения выхода из учетной записи  
 public static class DialogFragmentExit extends DialogFragment {  
  
 // Создание диалогового окна  
 @NonNull  
 @Override  
 public Dialog onCreateDialog(Bundle savedInstanceState) {  
 // Заголовок и кнопки диалогового окна  
 String title = "Вы уверены, что хотите завершить сеанс? ";  
 String button1String = "Да, я уверен(а)";  
 String button2String = "Отмена";  
 // Создание диалогового окна  
 AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());  
 builder.setTitle(title); // заголовок  
 // Установка обработчика для кнопки "Да"  
 builder.setPositiveButton(button1String, new DialogInterface.OnClickListener() {  
 public void onClick(DialogInterface dialog, int id) {  
 // Выход из учетной записи пользователя  
 FirebaseAuth.getInstance().signOut();  
 Intent intent = new Intent();  
 intent.setClass(getActivity(), AuthActivity.class);  
 getActivity().startActivity(intent);  
 }  
 });  
 // Установка обработчика для кнопки "Отмена"  
 builder.setNegativeButton(button2String, new DialogInterface.OnClickListener() {  
 public void onClick(DialogInterface dialog, int id) {  
 // Закрытие диалогового окна  
 }  
 });  
 builder.setCancelable(true);  
  
 return builder.create();  
 }

Код класса MyVoiceViewModel

package com.boom.voice.ui.myvoice;  
import androidx.lifecycle.LiveData;  
import androidx.lifecycle.MutableLiveData;  
import androidx.lifecycle.ViewModel;  
// ViewModel для фрагмента, отображающий список пользовательских голосований.  
public class MyVoiceViewModel extends ViewModel {  
 // данные для текста на фрагменте  
 private final MutableLiveData<String> mText;  
 // Конструктор класса MyVoiceViewModel  
 public MyVoiceViewModel() {  
 // Инициализация данных  
 mText = new MutableLiveData<>();  
 // Установка значения по умолчанию  
 mText.setValue("This is dashboard fragment");  
 }  
 // Получение данных для текста  
 public LiveData<String> getText() {  
 return mText;  
 }  
}

Код класса MyVoiceFragment

package com.boom.voice.ui.myvoice;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import androidx.annotation.NonNull;  
import androidx.fragment.app.Fragment;  
import androidx.lifecycle.ViewModelProvider;  
import androidx.recyclerview.widget.RecyclerView;  
import com.boom.voice.Adapter.AdapterMyVoice;  
import com.boom.voice.databinding.FragmentMyvoiceBinding;  
import com.boom.voice.ui.data.MyvoiceInfo;  
import com.boom.voice.ui.data.profile\_info;  
import java.util.ArrayList;  
// Фрагмент, отображающий список пользовательских голосований.  
public class MyVoiceFragment extends Fragment {  
 // Привязка для макета фрагмента  
 private FragmentMyvoiceBinding binding;  
 // Создание представления фрагмента  
 public View onCreateView(@NonNull LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
 // Получение экземпляра ViewModel для фрагмента  
 MyVoiceViewModel dashboardViewModel = new ViewModelProvider(this).get(MyVoiceViewModel.class);  
 // Настройка привязки макета фрагмента  
 binding = FragmentMyvoiceBinding.inflate(inflater, container, false);  
 View root = binding.getRoot();  
 // Получение списка информации о пользовательских голосованиях из профиля  
 ArrayList<MyvoiceInfo> partnersInfos = profile\_info.getInstance().getMyvoice();  
 // Получение RecyclerView для отображения списка голосований  
 RecyclerView recyclerView = binding.myVoiceList;  
 // Логирование размера списка голосований  
 Log.d("PARTNERS\_SIZE", String.valueOf(partnersInfos.get(0).getName()));  
 // Создание адаптера для списка пользовательских голосований  
 AdapterMyVoice adapter = new AdapterMyVoice(getActivity(), partnersInfos);  
 // Установка адаптера для RecyclerView, если список не пустой  
 if (partnersInfos.size()!=0) {  
 recyclerView.setAdapter(adapter);  
 }  
 return root;  
 }  
 // Уничтожение привязки макета фрагмента при уничтожении фрагмента  
 @Override  
 public void onDestroyView() {  
 super.onDestroyView();  
 binding = null;  
 }  
}

Код класса HomeViewModel

package com.boom.voice.ui.home;  
import androidx.lifecycle.LiveData;  
import androidx.lifecycle.MutableLiveData;  
import androidx.lifecycle.ViewModel;  
// ViewModel, отображающий список голосований  
public class HomeViewModel extends ViewModel {  
 // данные для текста на домашнем фрагменте  
 private final MutableLiveData<String> mText;  
 // Конструктор класса HomeViewModel  
 public HomeViewModel() {  
 // Инициализация живых данных  
 mText = new MutableLiveData<>();  
 // Установка значения по умолчанию  
 mText.setValue("This is home fragment");  
 }  
 // Получение данных для текста  
 public LiveData<String> getText() {  
 return mText;  
 }  
}

Код класса HomeFragment

package com.boom.voice.ui.home;  
import android.content.Intent;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import androidx.annotation.NonNull;  
import androidx.fragment.app.Fragment;  
import androidx.lifecycle.ViewModelProvider;  
import androidx.recyclerview.widget.RecyclerView;  
import com.boom.voice.Adapter.AdapterVoice;  
import com.boom.voice.databinding.FragmentHomeBinding;  
import com.boom.voice.ui.data.VoiceInfo;  
import com.boom.voice.ui.data.profile\_info;  
import com.boom.voice.ui.ui.GivingVoiceActivity;  
import java.util.ArrayList;  
import java.util.List;  
// Фрагмент домашней страницы, отображающий список голосований.  
public class HomeFragment extends Fragment {  
 // Привязка для макета фрагмента  
 private FragmentHomeBinding binding;  
 // Создание представления фрагмента  
 public View onCreateView(@NonNull LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
 // Получение экземпляра ViewModel для фрагмента  
 HomeViewModel homeViewModel = new ViewModelProvider(this).get(HomeViewModel.class);  
 // Настройка привязки макета фрагмента  
 binding = FragmentHomeBinding.inflate(inflater, container, false);  
 View root = binding.getRoot();  
 // Получение списка информации о голосовании из профиля  
 ArrayList<VoiceInfo> listInfos = profile\_info.getInstance().getVoiceInfos();  
 // Получение RecyclerView для отображения списка голосований  
 RecyclerView recyclerView = binding.VoiceList;  
 // Получение списка элементов первого голосования  
 List<String> item = new ArrayList();  
 item = listInfos.get(0).getItem();  
 // Логирование размера списка элементов первого голосования  
 Log.d("PARTNERS\_SIZE", String.valueOf(item.get(0).length()));  
 // Обработчик клика на элемент списка  
 AdapterVoice.OnStateClickListener stateClickListener = new AdapterVoice.OnStateClickListener() {  
 @Override  
 public void onStateClick(VoiceInfo partners, int position) {  
 // Переход к активности для участия в голосовании  
 Intent i = new Intent(getActivity(), GivingVoiceActivity.class);  
 i.putExtra(VoiceInfo.class.getSimpleName(), partners);  
 startActivity(i);  
 }  
 };  
 // Создание адаптера для списка голосований  
 AdapterVoice adapter = new AdapterVoice(stateClickListener, getActivity(), listInfos);  
 // Установка адаптера для RecyclerView, если список голосований не пустой  
 if (listInfos.size()!=0) {  
 recyclerView.setAdapter(adapter);  
 }  
 return root;  
 }  
 // Уничтожение привязки макета фрагмента при уничтожении фрагмента  
 @Override  
 public void onDestroyView() {  
 super.onDestroyView();  
 binding = null;  
 }  
}

Код класса VoiceInfo

package com.boom.voice.ui.data;  
import java.io.Serializable;  
import java.util.List;  
// Этот класс представляет информацию о голосовании, включая название, пользователей, список элементов,  
// список "за", список "против" и идентификатор.  
public class VoiceInfo implements Serializable {  
 // Название голосования.  
 String name;  
 // Пользователи.  
 String users;  
 // Список элементов.  
 List<String> item;  
 // Список "за".  
 List<String> za;  
 // Список "против".  
 List<String> protiv;  
 // Идентификатор голосования.  
 String id;  
 // Получает список "против".  
 public List<String> getProtiv() {  
 return protiv;  
 }  
 // Устанавливает список "против".  
 public void setProtiv(List<String> protiv) {  
 this.protiv = protiv;  
 }  
 // Получает список "за".  
 public List<String> getZa() {  
 return za;  
 }  
 // Устанавливает список "за".  
 public void setZa(List<String> za) {  
 this.za = za;  
 }  
 // Устанавливает название голосования.  
 public void setName(String name) {  
 this.name = name;  
 }  
 // Получает название голосования.  
 public String getName() {  
 return name;  
 }  
 // Получает список элементов.  
 public List<String> getItem() {  
 return item;  
 }  
 // Устанавливает список элементов.  
 public void setItem(List<String> item ) {  
 this.item = item;  
 }  
 // Получает пользователей.  
 public String getUsers() {  
 return users;  
 }  
 // Устанавливает пользователей.  
 public void setUsers(String users) {  
 this.users = users;  
 }  
 // Устанавливает идентификатор голосования.  
 public void setId(String id) {  
 this.id = id;  
 }  
 // Получает идентификатор голосования.  
 public String getId() {  
 return id;  
 }  
}

Код класса user\_info

package com.boom.voice.ui.data;  
// Этот класс представляет информацию о пользователе, включая электронную почту, имя, идентификатор и год рождения.  
public class user\_info {  
 // Адрес электронной почты пользователя.  
 String email;  
 // Имя пользователя.  
 String name;  
 // Уникальный идентификатор пользователя.  
 String id;  
 // Год рождения пользователя.  
 String year;  
 // Получает уникальный идентификатор пользователя.  
 public String getId(){  
 return id;  
 }  
 // Устанавливает уникальный идентификатор пользователя.  
 public void setId(String id){  
 this.id = id;  
 }  
 // Получает год рождения пользователя.  
 public String getYear() {  
 return year;  
 }  
 // Устанавливает год рождения пользователя.  
 public void setYear(String year) {  
 this.year = year;  
 }  
 // Получает адрес электронной почты пользователя.  
 public String getEmail() {  
 return email;  
 }  
 // Устанавливает адрес электронной почты пользователя.  
 public void setEmail(String email) {  
 this.email = email;  
 }  
 // Получает имя пользователя.  
 public String getName() {  
 return name;  
 }  
 // Устанавливает имя пользователя.  
 public void setName(String name) {  
 this.name = name;  
 }  
}

Код класса profile\_info

package com.boom.voice.ui.data;  
import java.util.ArrayList;  
// Этот класс представляет информацию о профиле, включая информацию о голосе, информацию о пользователе и др.  
public class profile\_info {  
  
 // Единственный экземпляр информации о профиле.  
 private static profile\_info dataObject = null;  
  
 // Список объектов информации о голосе.  
 private ArrayList<VoiceInfo> mInfos = new ArrayList<>();  
  
 // Список пользовательской информации о голосе.  
 private ArrayList<MyvoiceInfo> myVoiceInfos = new ArrayList<>();  
  
 // Объект информации о пользователе.  
 private user\_info myUserInfo = new user\_info();  
  
 // Получает список объектов информации о голосе.  
 public ArrayList<VoiceInfo> getVoiceInfos() {  
 return mInfos;  
 }  
  
 // Устанавливает список объектов информации о голосе.  
 public void setVoiceInfos(ArrayList<VoiceInfo> mapMarketInfos) {  
 this.mInfos = mapMarketInfos;  
 }  
 // Получает список пользовательской информации о голосе.  
 public ArrayList<MyvoiceInfo> getMyVoice() {  
 return myVoiceInfos;  
 }  
  
 // Устанавливает список пользовательской информации о голосе.  
 public void setMyVoice(ArrayList<MyvoiceInfo> Infos) {  
 this.myVoiceInfos = Infos;  
 }  
  
 // Получает объект информации о пользователе.  
 public user\_info getMy\_user\_info() {  
 return myUserInfo;  
 }  
  
 // Устанавливает объект информации о пользователе.  
 public void setMy\_user\_info(user\_info my\_user\_info) {  
 this.myUserInfo = my\_user\_info;  
 }  
  
 // Получает единственный экземпляр информации о профиле.  
 public static profile\_info getInstance() {  
 if (dataObject == null)  
 dataObject = new profile\_info();  
 return dataObject;  
 }  
 // Получает список пользовательской информации о голосе.  
 public ArrayList<MyvoiceInfo> getMyvoice() {  
 return myVoiceInfos;  
 }  
}

Код класса MyvoiceInfo

package com.boom.voice.ui.data;  
// Этот класс представляет информацию о голосе, включая его имя, идентификатор и тип.  
public class MyvoiceInfo {  
 // Уникальный идентификатор голоса.  
 private String id;  
 // Имя голоса.  
 private String name;  
 // Тип или характеристики голоса.  
 private String voice;  
 // Устанавливает идентификатор голоса.  
 public void setId(String id) {  
 this.id = id;  
 }  
 // Получает идентификатор голоса.  
 public String getId() {  
 return id;  
 }  
 // Устанавливает имя голоса.  
 public void setName(String name) {  
 this.name = name;  
 }  
 // Получает имя голоса.  
 public String getName() {  
 return name;  
 }  
 // Устанавливает тип или характеристики голоса.  
 public void setVoice(String voice) {  
 this.voice = voice;  
 }  
 // Получает тип или характеристики голоса.  
 public String getVoice() {  
 return voice;  
 }  
}

Код класса FireBaseLoad

package com.boom.voice.ui.data;  
import static android.content.ContentValues.TAG;  
import android.app.Activity;  
import android.content.Context;  
import android.util.Log;  
import androidx.annotation.NonNull;  
import com.google.android.gms.tasks.OnCompleteListener;  
import com.google.android.gms.tasks.Task;  
import com.google.firebase.auth.FirebaseAuth;  
import com.google.firebase.auth.FirebaseUser;  
import com.google.firebase.firestore.DocumentReference;  
import com.google.firebase.firestore.DocumentSnapshot;  
import com.google.firebase.firestore.FirebaseFirestore;  
import com.google.firebase.firestore.QueryDocumentSnapshot;  
import com.google.firebase.firestore.QuerySnapshot;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
public class FireBaseLoad {  
 Context context;  
 user\_info user = new user\_info();  
 ArrayList<VoiceInfo> voiceInfos;  
 ArrayList<MyvoiceInfo> myInfos;  
 private FirebaseAuth mAuth= FirebaseAuth.getInstance();  
 FirebaseUser userDB= mAuth.getCurrentUser();  
 FirebaseFirestore db = FirebaseFirestore.getInstance();  
 profile\_info profileInfo = new profile\_info();  
 // Конструктор класса FireBaseLoad  
 public FireBaseLoad(Context context){  
 this.context=context;  
 }  
 // Метод для получения информации о пользователе  
 public user\_info getUserInfo(String UserToken) {  
 user\_info userInfo = new user\_info();  
 DocumentReference docRef = db.collection("users").document(UserToken);  
 docRef.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<DocumentSnapshot> task) {  
 if (task.isSuccessful()) {  
 DocumentSnapshot document = task.getResult();  
 if (document.exists()) {  
 Log.d(TAG, "DocumentSnapshot data: " + document.getData());  
 String email = notNullStrForUsers(document, "email");  
 String name = notNullStrForUsers(document, "name");  
 String year = notNullStrForUsers(document, "year");  
 // Установка полученных данных о пользователе  
 userInfo.setEmail(email);  
 userInfo.setName(name);  
 userInfo.setYear(year);  
 userInfo.setId(document.getId());  
 profileInfo.getInstance().setMy\_user\_info(userInfo);  
 Log.d("map", "setMy\_user\_info");  
 } else {  
 Log.d(TAG, "No such document");  
 }  
 } else {  
 Log.d(TAG, "get failed with ", task.getException());  
 }  
 }  
 });  
 return userInfo;  
 }  
 // Метод для получения информации о голосах  
 public void getVoiceInfo () {  
 db.collection("voice")  
 .get()  
 .addOnCompleteListener((Activity) context, new OnCompleteListener<QuerySnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<QuerySnapshot> task) {  
 voiceInfos = new ArrayList<>();  
 if (task.isSuccessful()) {  
 for (QueryDocumentSnapshot document : task.getResult()) {  
 VoiceInfo voiceInfo = new VoiceInfo();  
 voiceInfo.setId(document.getId()); // 5ADD4A60-236B-471A-9CCF-CD7325E53462  
 Map<String, Object> responseObj = new HashMap<String, Object>();  
 Object arr = new String[10];  
 String name = notNullStr (document , "name" );  
 String users = notNullStr (document , "users" );  
 String item = (notNullStr (document , "item" ));  
 String protiv = (notNullStr (document , "protiv" ));  
 String za= (notNullStr (document , "za" ));  
 // Установка полученных данных о голосах  
 voiceInfo.setName(name);  
 voiceInfo.setItem(stringToArrList(item));  
 voiceInfo.setUsers(users);  
 voiceInfo.setProtiv(stringToArrList(protiv));  
 voiceInfo.setZa(stringToArrList(za));  
 Log.d("RESP", arr.toString());  
 voiceInfos.add(voiceInfo);  
 Log.d(TAG, document.getId() + " => " + document.getData());  
 }  
 profileInfo.getInstance().  
 setVoiceInfos(voiceInfos);  
 } else {  
 Log.d(TAG, "Error getting documents: ", task.getException());  
 }  
 }  
 });  
 }  
 // Метод для получения информации о голосах пользователя  
 public void getMyVoiceInfo() {  
 myInfos = new ArrayList<>();  
 String uid = userDB.getUid();  
 db.collection("myvoice")  
 .whereEqualTo("id", uid)  
 .get()  
 .addOnCompleteListener((Activity) context, new OnCompleteListener<QuerySnapshot>() {  
 @Override  
 public void onComplete(@NonNull Task<QuerySnapshot> task) {  
 if (task.isSuccessful()) {  
 for (QueryDocumentSnapshot document : task.getResult()) {  
 MyvoiceInfo pInfo = new MyvoiceInfo();  
 Map<String, Object> responseObj = new HashMap<String, Object>();  
 String name = notNullStr(document, "name");  
 String voice = notNullStr(document, "myvoice");  
 Log.d("PART", name);  
 // Установка полученных данных о голосах пользователя  
 pInfo.setName(name);  
 pInfo.setVoice(voice);  
 myInfos.add(pInfo);  
 Log.d(TAG, document.getId() + " => " + document.getData());  
 }  
 profileInfo.getInstance().setMyvoice(myInfos);  
 Log.d("map", "setPartnerInfos");  
 } else {  
 Log.d(TAG, "Error getting documents: ", task.getException());  
 Log.d("map", "Error getting documents: ", task.getException());  
 }  
 }  
 });  
 }  
 // Преобразование строки в список строк  
 public List<String> stringToArrList(String s){  
 s = s.replaceAll("[\\[\\]]", "");  
 List<String> arr = new ArrayList<String>(Arrays.asList(s.split(",")));  
 return arr;  
 }  
 // Получение значения из поля документа, обработка исключений  
 public String notNullStr(QueryDocumentSnapshot document, String tokenName) {  
 String dd = "";  
 try {  
 dd = document.getData().get(tokenName).toString();  
 } catch (Exception e) {  
 }  
 return dd;  
 }  
 public String notNullStrForUsers(DocumentSnapshot document, String tokenName) {  
 String dd = "";  
 try {  
 dd = document.getData().get(tokenName).toString();  
 } catch (Exception e) {  
 }  
 return dd;  
 }  
 public void removeVoice (String s) {  
 db.collection("voice").document(s).delete();  
 }  
}

Код класса AdapterVoice

package com.boom.voice.Adapter;  
import android.annotation.SuppressLint;  
import android.content.Context;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.TextView;  
import androidx.recyclerview.widget.RecyclerView;  
import com.boom.voice.R;  
import com.boom.voice.ui.data.VoiceInfo;  
import java.util.List;  
public class AdapterVoice extends RecyclerView.Adapter<AdapterVoice.ViewHolder>{  
 // Интерфейс для обработки нажатий на элементы RecyclerView  
 public interface OnStateClickListener{  
 void onStateClick(VoiceInfo partners , int position);  
 }  
 private final OnStateClickListener onClickListener; // Обработчик нажатий  
 private final LayoutInflater inflater; // LayoutInflater для инфляции макета XML  
 private final List<VoiceInfo> voiceInfoList; // Список для хранения объектов VoiceInfo  
 // Конструктор класса AdapterVoice  
 public AdapterVoice(OnStateClickListener onClickListener, Context context, List<VoiceInfo> voiceInfoList) {  
 this.onClickListener = onClickListener; // Установка обработчика нажатий  
 this.voiceInfoList = voiceInfoList; // Инициализация списка объектов VoiceInfo  
 this.inflater = LayoutInflater.from(context); // Инициализация LayoutInflater  
 }  
 // Создание новых представлений (вызывается менеджером макета)  
 @Override  
 public AdapterVoice.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
 // Надуть макет для каждого элемента в RecyclerView  
 View view = inflater.inflate(R.layout.voice\_list, parent, false);  
 return new ViewHolder(view); // Возвращаем новый экземпляр ViewHolder  
 }  
 // Замена содержимого представления (вызывается менеджером макета)  
 @Override  
 public void onBindViewHolder(AdapterVoice.ViewHolder holder, @SuppressLint("RecyclerView") int position) {  
 VoiceInfo partners = voiceInfoList.get(position); // Получить объект VoiceInfo по заданной позиции  
 holder.nameView.setText(partners.getName()); // Установить имя в TextView  
 // Установка обработчика нажатий на элемент списка  
 holder.itemView.setOnClickListener(new View.OnClickListener(){  
 @Override  
 public void onClick(View v) {  
 onClickListener.onStateClick(partners, position); // Вызов метода из интерфейса для обработки нажатия  
 }  
 });  
 }  
 // Возвращает размер набора данных (вызывается менеджером макета)  
 @Override  
 public int getItemCount() {  
 return voiceInfoList.size(); // Возвращаем размер списка VoiceInfo  
 }  
 // Класс ViewHolder для хранения ссылок на представления для каждого элемента в RecyclerView  
 public static class ViewHolder extends RecyclerView.ViewHolder {  
 final TextView nameView, descriptionView; // TextView для отображения имени и описания  
 ViewHolder(View view){  
 super(view); // Вызов конструктора суперкласса  
 nameView = view.findViewById(R.id.nameList); // Инициализация TextView для имени  
 descriptionView = view.findViewById(R.id.List); // Инициализация TextView для описания  
 }  
 }  
}

Код класса AdapterMyVoice

package com.boom.voice.Adapter;  
import android.content.Context;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.TextView;  
import androidx.recyclerview.widget.RecyclerView;  
import com.boom.voice.R;  
import com.boom.voice.ui.data.MyvoiceInfo;  
import java.util.List;  
public class AdapterMyVoice extends RecyclerView.Adapter<AdapterMyVoice.ViewHolder>{  
 private final LayoutInflater inflater; // LayoutInflater для инфляции макета XML  
 private final List<MyvoiceInfo> myvoiceInfoList; // Список для хранения объектов MyvoiceInfo  
 // Конструктор класса AdapterMyVoice  
 public AdapterMyVoice(Context context, List<MyvoiceInfo> myvoiceInfoList) {  
 this.myvoiceInfoList = myvoiceInfoList; // Инициализация списка объектов MyvoiceInfo  
 this.inflater = LayoutInflater.from(context); // Инициализация LayoutInflater  
 }  
 // Создание новых представлений (вызывается менеджером макета)  
 @Override  
 public AdapterMyVoice.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
 // макет для каждого элемента в RecyclerView  
 View view = inflater.inflate(R.layout.my\_voice\_list, parent, false);  
 return new ViewHolder(view); // Возвращаем новый экземпляр ViewHolder  
 }  
 // Замена содержимого представления   
 @Override  
 public void onBindViewHolder(AdapterMyVoice.ViewHolder holder, int position) {  
 MyvoiceInfo partners = myvoiceInfoList.get(position); // Получить объект MyvoiceInfo по заданной позиции  
 holder.nameView.setText(partners.getName()); // Установить имя в TextView  
 holder.descriptionView.setText(partners.getVoice()); // Установить описание голоса в TextView  
 }  
 // Возвращает размер набора данных   
 @Override  
 public int getItemCount() {  
 return myvoiceInfoList.size(); // Возвращаем размер списка MyvoiceInfo  
 }  
 // Класс ViewHolder для хранения ссылок на представления для каждого элемента в RecyclerView  
 public static class ViewHolder extends RecyclerView.ViewHolder {  
 final TextView nameView, descriptionView; // TextView для отображения имени и описания  
 ViewHolder(View view){  
 super(view); // Вызов конструктора суперкласса  
 nameView = view.findViewById(R.id.nameList); // Инициализация TextView для имени  
 descriptionView = view.findViewById(R.id.List); // Инициализация TextView для описания  
 }  
 }  
}

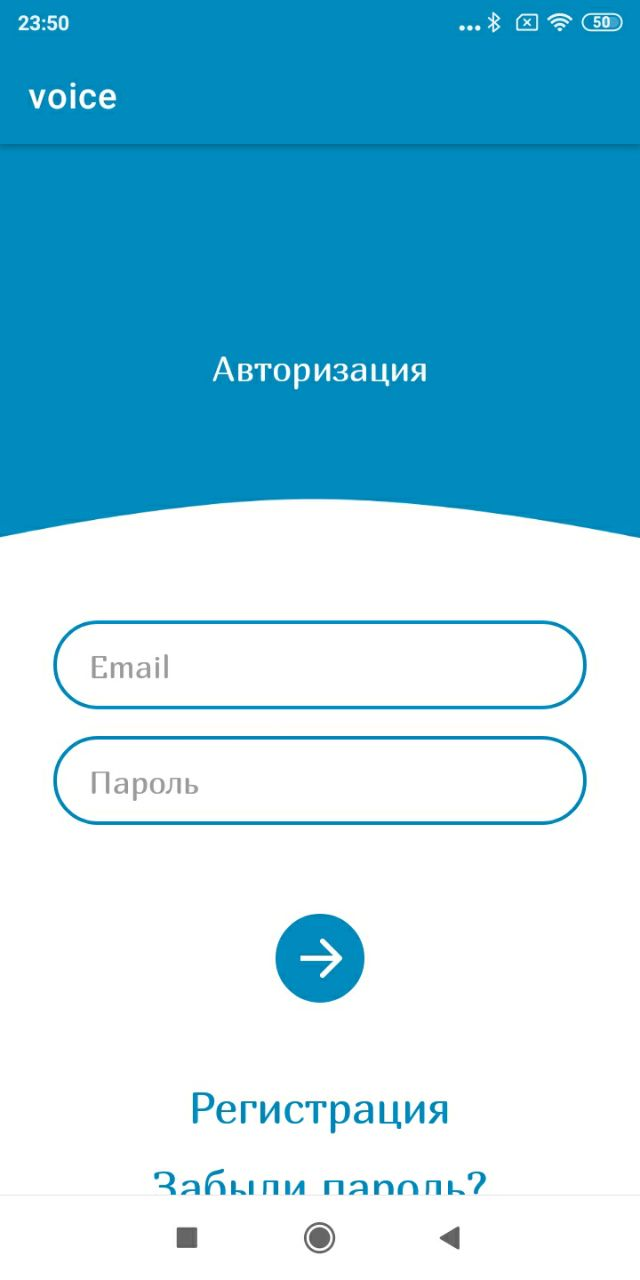
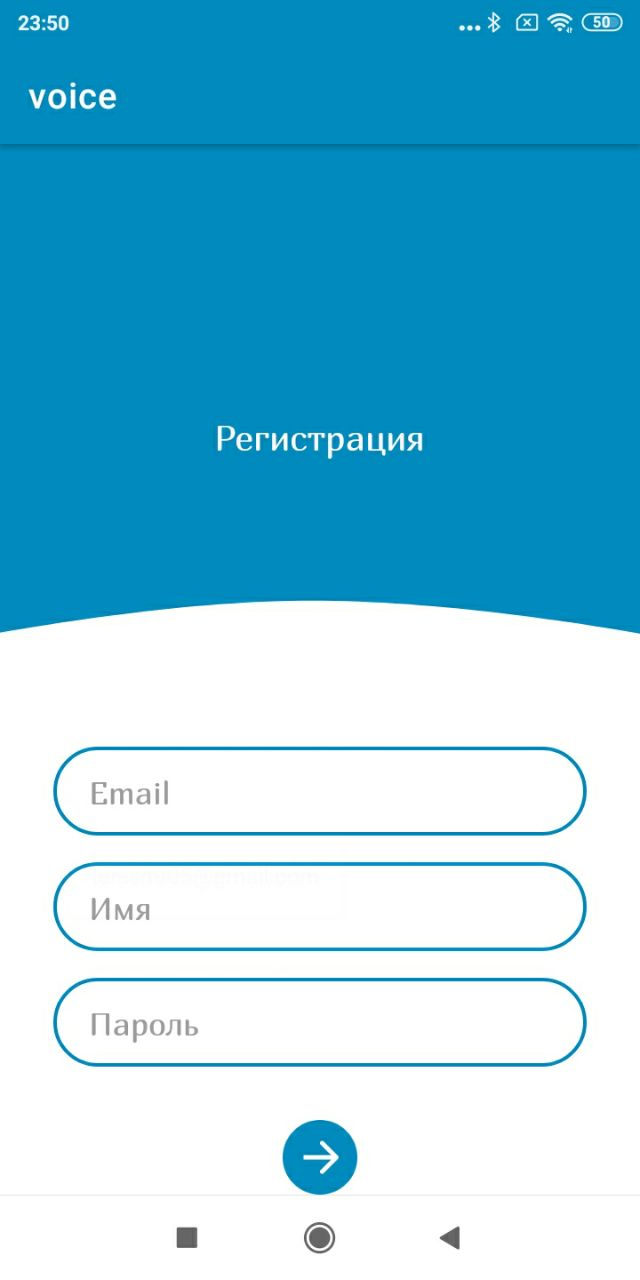
Код AndroidManifest.xml

*<?*xml version="1.0" encoding="utf-8"*?>*<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
 <uses-permission android:name="android.permission.INTERNET" />  
 <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  
 <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  
 <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />  
 <application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/Theme.Voice"  
 android:usesCleartextTraffic="true"  
 tools:targetApi="m">  
 <activity  
 android:name=".ui.ui.ProfileRenameActivity"  
 android:exported="false" />  
 <activity  
 android:name=".ui.ui.GivingVoiceActivity"  
 android:exported="false" />  
 <activity  
 android:name=".ui.ui.CreatVoiceActivity"  
 android:exported="false" />  
 <activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:label="@string/app\_name" />  
 <activity  
 android:name=".ui.ui.AuthActivity"  
 android:exported="true"  
 android:label="@string/app\_name" />  
 <activity  
 android:name=".ui.ui.SplashActivity"  
 android:exported="true"  
 android:label="@string/app\_name">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 <activity  
 android:name=".ui.ui.RegistrUserActivity"  
 android:exported="true"  
 android:label="@string/app\_name" />  
 </application>  
</manifest>

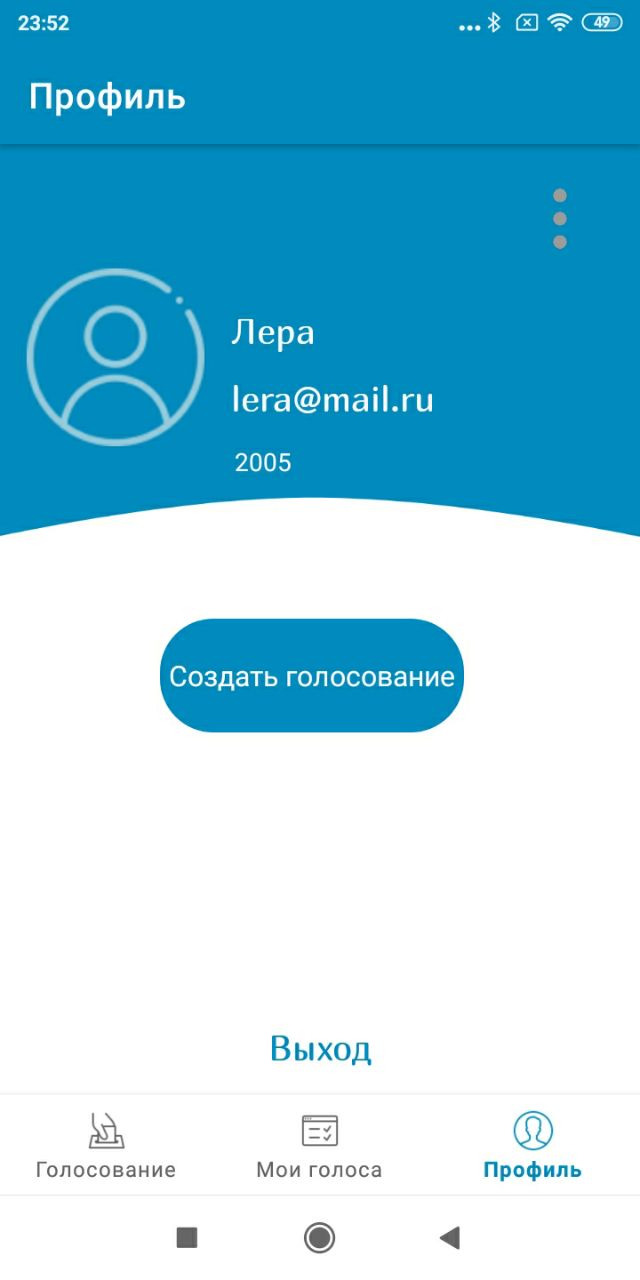
# Приложение Б. Руководство пользователя

Для начала работы с приложением «Voice» необходимо открыть его на своем мобильном устройстве. Приложение доступно для установки при помощи .apk установщика для устройств на базе операционной системы Android.

После установки и запуска приложения пользователю необходимо войти или зарегистрироваться в системе. При регистрации пользователю необходимо ввести данные, такие как email, имя и придумать пароль, после чего необходимо заново ввести данные уже при входе в систему.



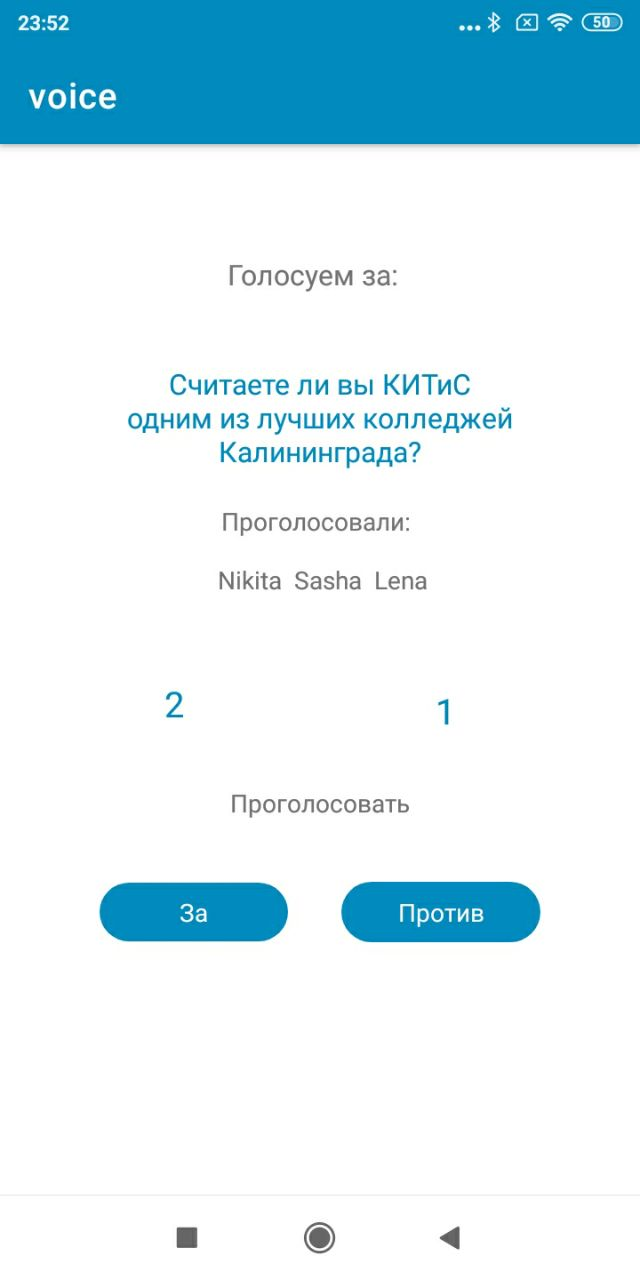
После того как пользователь авторизовался в системе, он попадает на свою личную страницу, где он может нажать на три точки в правом верхнем углу и заполнить свои данные, такие как имя, год рождения и email.



Так же в этом разделе перейдя по кнопке создать голосование, пользователь может придумать название для своего личного голосования. После нажатия кнопки создать, голосование будет видно другим пользователям.



В разделе «Голосования» пользователь может выбрать понравившиеся голосование и принять в нем участие, нажатием по нему. После чего он попадет в голосование, где может выбрать за или против, а также увидеть сколько пользователей участвовали именно в этом опросе.



Когда пользователь принял участие в голосовании и выбрал, за или против, в разделе «Мои голоса» он увидит список опросов, в которых он принял участие.

