МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра математического обеспечения ЭВМ

Разработка мобильного приложения для отслеживания расписания на платформе Android

Курсовая работа по дисциплине
Б1.О.31 Проектирование информационных систем

02.03.02 Фундаментальная информатика и информационные технологии

Инженерия программного обеспечения

Зав. кафедрой _____
д.т.н., профессор Абрамов Г.В. _____

Обучающийся _____
Вафин А.Р. _____

Руководитель _____
к.ф.-м.н., доцент Болотова С.Ю. _____

Воронеж 2024

Содержание

Введение

В современном мире мобильные приложения стали неотъемлемой частью повседневной жизни, охватывая все новые аспекты человеческой деятельности. Особую значимость приобретает их использование в образовательной сфере, в частности для организации учебного процесса в высших учебных заведениях. Разработка специализированных приложений для отслеживания расписания занятий представляет собой актуальную задачу, обусловленную возрастающей цифровизацией образования и повсеместным использованием смартфонов студенческой аудиторией.

Актуальность разработки подобного приложения для платформы Android обусловлена несколькими ключевыми факторами. Во-первых, Android занимает лидирующие позиции на рынке мобильных операционных систем, что обеспечивает широкий охват пользовательской аудитории. Во-вторых, современные технологии разработки позволяют реализовать эффективную синхронизацию с университетскими базами данных через специализированные API. В-третьих, существует возможность внедрения дополнительных функций, таких как интеллектуальные уведомления, аналитика учебной нагрузки и персонализированные рекомендации, что способствует оптимизации образовательного процесса. [**?**]

Таким образом, создание мобильного приложения для управления расписанием в вузе представляет собой научно-практическую задачу, решение которой может значительно повысить эффективность организации учебной деятельности. Перспективным направлением дальнейших исследований является интеграция технологий искусственного интеллекта для прогнозирования и адаптации учебного процесса под индивидуальные потребности студентов.

1. Постановка цели и задач

Целью данной работы является разработка мобильного приложения для отслеживания расписания в вузе на платформе Android. Для достижения этой цели необходимо решить следующие задачи:

1. Проектирование и реализация приложения, включая создание интерфейса, использование баз данных для хранения расписания и механизмов синхронизации для обновления данных в реальном времени.
2. Реализация выбранного API, интеграция его в приложение и тестирование работы, включая написание кода для взаимодействия с API, обработку данных и обеспечение корректной работы всех функций приложения.

2. Анализ средств реализации

Для достижения поставленных целей и решения задач будут использованы следующие методы анализа и исследования:

1. Анализ существующих приложений для отслеживания расписания и API.
2. Проектирование пользовательского интерфейса с использованием методов UX/UI дизайна.
3. Разработка и тестирование программного кода приложения.

2.1. Обзор существующих приложений

При анализе приложения ”Расписание занятий”был выявлен ряд преимуществ. Одним из достоинств является возможность синхронизации данных с Google Sheets API, что обеспечивает актуальное отображение расписания. Еще одним преимуществом выступает функция добавления и редактирования заметок к каждому занятию, повышающая удобство использования. Простота интеграции с Google Sheets и быстрая загрузка данных также относятся к сильным сторонам данного решения. Однако приложение имеет существенные недостатки, среди которых отсутствие поддержки на новых версиях Android и невозможность отображения занятий по числителю/знаменателю. [**?**] Приложение ”Учебный календарь”демонстрирует другие преимущества. Важным достоинством является не только синхронизация с Google Sheets API, но и возможность персонализации интерфейса через изменение цветовой схемы. Особого внимания заслуживает реализация заметок о занятии с функцией их включения и отключения, что предоставляет пользователям гибкость в настройке отображения информации. Еще одним значимым преимуществом стала поддержка отображения занятий по числителю/знаменателю. Однако приложение страдает от перегруженного интерфейса и медленной загрузки данных, а также не предоставляет возможности самостоятельного выбора числителя/знаменателя. На основании проведенного анализа было принято решение объединить сильные стороны обоих приложений, дополнив их новыми функциями. Ключевыми особенностями разрабатываемого решения станут: возможность изменения цветовой схемы интерфейса, реализация системы заметок о занятии с функцией их отображения/скрытия,

а также введение выбора роли (студент/преподаватель) и группы для студентов. Особое внимание уделено реализации самостоятельного выбора числителя/знаменателя, что обеспечит большую гибкость в работе с расписанием независимо от номера недели.

2.2. Обзор существующих языков программирования

При сравнении Java и Kotlin выявлены существенные различия в подходах к разработке. Java, будучи зрелым языком, требует больше ручного управления кодом, включая написание геттеров, сеттеров и других шаблонных конструкций, что увеличивает объем работы и вероятность ошибок. В то же время Kotlin предлагает более высокоуровневый подход с автоматической генерацией boilerplate-кода, например, в data-классах, что делает разработку быстрее и удобнее. В плане производительности оба языка компилируются в байт-код JVM, однако Kotlin предоставляет встроенные оптимизации, такие как null-безопасность и корутины, которые упрощают асинхронное программирование и снижают риск ошибок. Java, хотя и остается высокопроизводительным, требует ручной оптимизации, особенно при работе с памятью и многопоточностью. Гибкость Kotlin проявляется в поддержке функционального программирования, включая лямбда-выражения и расширения функций, что позволяет писать более лаконичный и модульный код. Java, несмотря на добавление Stream API и лямбд, все еще уступает в этом отношении, требуя большего количества шаблонного кода. Поддержка и сообщество Java остаются одними из самых больших, что делает язык надежным выбором для legacy-проектов. Однако Kotlin активно развивается, особенно в Android-разработке, и полностью совместим с Java, что позволяет использовать существующие библиотеки и постепенно переходить на более современный стек. Тестирование в Kotlin упрощается благодаря лаконичному синтаксису и встроенным возможностям, таким как корутины, тогда как в Java для сложных сценариев часто требуются дополнительные библиотеки. В итоге Kotlin был выбран как более современный, безопасный и удобный язык, сочетающий производительность Java с улучшенным синтаксисом и инструментами.

2.3. Обзор существующих API

Использование Google Sheets API для синхронизации данных имеет ряд существенных ограничений, что делает разработку собственного сервера более предпочтительным решением. Одним из основных недостатков является строгое ограничение на количество запросов в минуту, что может привести к задержкам или блокировке доступа при высокой нагрузке, тогда как собственный сервер позволяет гибко масштабировать ресурсы в зависимости от числа пользователей. Еще одной проблемой Google Sheets API является сложность расширения функциональности, поскольку он не поддерживает такие возможности, как уведомления или аналитика, в то время как собственный сервер предоставляет полный контроль над логикой приложения и позволяет легко интегрировать дополнительные функции. Изменение данных через Google Sheets также оказывается неудобным, так как требует ручного редактирования таблиц, что повышает риск ошибок, тогда как собственный сервер может быть оснащен удобной административной панелью для управления расписанием. Производительность Google Sheets API оставляет желать лучшего при работе с большими объемами данных, вызывая задержки, в отличие от собственного сервера, который можно оптимизировать для быстрой обработки запросов. Кроме того, Google Sheets API ограничен табличной структурой, что затрудняет работу со сложными данными, в то время как собственный сервер поддерживает различные форматы хранения, включая реляционные базы, обеспечивая большую гибкость. Таким образом, собственный сервер предлагает значительные преимущества, включая полный контроль над данными, масштабируемость, удобство управления и расширяемость, что делает его оптимальным выбором для обеспечения стабильной и функциональной работы приложения.

2.4. Обзор существующих фреймворков для интерфейса

При сравнении подходов к созданию пользовательского интерфейса явно прослеживаются ключевые различия между традиционной XML-верстkой и современным Jetpack Compose. XML-верстка, несмотря на свою зрелость и обширную базу существующих решений, демонстрирует ряд существенных ограничений, таких как жесткое разделение логики и интер-

фейса, необходимость ручного управления View-элементами и сложности с динамическим обновлением контента. В противоположность этому, Jetpack Compose предлагает единый программный подход на Kotlin, где интерфейс описывается декларативно непосредственно в коде, что значительно упрощает разработку и поддержку. С точки зрения производительности Compose обладает явными преимуществами благодаря своей системе ”рекомпозиции которая интеллектуально обновляет только измененные элементы интерфейса. В отличие от традиционной View-системы с ее глубокими иерархиями и ручной оптимизацией, Compose обеспечивает более эффективное использование ресурсов устройства, что особенно важно для современных требовательных приложений. Гибкость Compose проявляется в простоте создания сложных анимированных интерфейсов и кастомизации компонентов. В то время как XML-верстка требует написания значительного объема кода для реализации нестандартных решений, Compose предоставляет встроенные инструменты для анимаций и позволяет легко создавать переиспользуемые компоненты. Несмотря на то, что XML-верстка имеет более обширное сообщество и документацию из-за своей долгой истории, Jetpack Compose активно развивается при поддержке Google и постепенно становится новым стандартом в Android-разработке. Его возможность сосуществования с традиционными View позволяет осуществлять плавный переход на новые технологии. В области тестирования Compose также предлагает более удобные решения со встроенными инструментами тестирования, что упрощает процесс проверки UI по сравнению с необходимостью использования дополнительных библиотек в случае XML-верстки. Таким образом, Jetpack Compose был выбран как современный, производительный и гибкий фреймворк, который значительно ускоряет разработку интерфейсов за счет сокращения шаблонного кода и предоставления мощных инструментов для создания отзывчивых и интерактивных пользовательских интерфейсов.

2.5. Обзор существующих фреймворков базы данных

При выборе подхода к работе с базами данных в Android-приложениях рассматривались два основных решения: нативный SQLite и библиотека Room. Нативный SQLite, являясь стандартным решением для Android, требует значительных усилий при реализации - разработчику необходимо вруч-

ную создавать таблицы, писать SQL-запросы и управлять соединениями, что приводит к большому объему шаблонного кода. В отличие от этого, Room предоставляет удобную абстракцию над SQLite, автоматически генерируя необходимый код на основе аннотаций, что значительно сокращает время разработки и уменьшает вероятность ошибок. С точки зрения производительности оба подхода демонстрируют сопоставимые результаты, так как Room в конечном итоге использует тот же SQLite. Однако Room предлагает дополнительные оптимизации, такие как кэширование запросов и проверка их корректности на этапе компиляции, что позволяет избежать распространенных ошибок, характерных для ручного написания SQL-запросов. При этом нативный SQLite сохраняет преимущество при работе с особо сложными запросами, где может потребоваться тонкая ручная оптимизация. Гибкость нативного SQLite проявляется в возможности выполнения произвольных запросов и полном контроле над структурой базы данных, однако это требует от разработчика глубоких знаний SQL и тщательного управления миграциями. Room, хотя и ограничивает некоторые низкоуровневые возможности, предоставляет удобные механизмы для работы с транзакциями и миграциями данных, существенно упрощая эти процессы. Особенно ценным является встроенная в Room поддержка LiveData и Flow, позволяющая легко реализовать реактивное обновление интерфейса при изменениях в базе данных. Несмотря на то, что нативный SQLite имеет более обширную базу знаний и примеров благодаря своей долгой истории, Room активно развивается как часть Android Jetpack и становится стандартом для новых проектов. Его популярность постоянно растет, а сообщество разработчиков расширяется, что обеспечивает хорошую поддержку и наличие актуальных решений для типовых задач. Важным преимуществом Room являются встроенные инструменты тестирования, такие как возможность создания временных баз данных в памяти, что существенно упрощает процесс написания и выполнения unit-тестов по сравнению с нативным SQLite, где требуется дополнительная настройка тестового окружения. Таким образом, Room был выбран в качестве основного инструмента для работы с базами данных благодаря своей простоте использования, безопасности типов, отличной интеграции с другими компонентами Android Jetpack и возможности сосредоточиться на бизнес-логике приложения, а не на низкоуровневых деталях работы с SQLite. [**?**]

2.6. Обзор существующих фреймворков для работы с сетью

При разработке сетевого слоя приложения рассматривались два основных подхода: низкоуровневый OkHttp и высокоуровневый Retrofit. OkHttp предоставляет полный контроль над HTTP-запросами, позволяя тонко настраивать все параметры соединения, однако требует значительного объема ручного кода для реализации типовых сценариев. В отличие от него, Retrofit предлагает декларативный подход через аннотированные интерфейсы, автоматизируя создание клиентов и обработку ответов, что существенно сокращает объем шаблонного кода. С точки зрения производительности оба решения демонстрируют отличные результаты, поскольку Retrofit использует OkHttp в качестве транспортного уровня. OkHttp обеспечивает продвинутые оптимизации, включая поддержку HTTP/2, кэширование и сжатие данных, в то время как Retrofit добавляет удобную абстракцию поверх этих возможностей без потери эффективности. Гибкость OkHttp проявляется в возможности создания кастомных интерцепторов и полного контроля над запросами, что критично для сложных интеграций. Retrofit, хотя и ограничивает некоторые низкоуровневые возможности, сохраняет доступ к базовым настройкам OkHttp и предоставляет удобные механизмы для адаптации ответов и обработки ошибок. Оба решения имеют отличную поддержку и активное сообщество благодаря принадлежности к экосистеме Square. OkHttp, как фундаментальная библиотека, используется в большинстве Android-приложений, тогда как Retrofit стал стандартом де-факто для работы с REST API благодаря своей простоте и элегантности API. В области тестирования Retrofit предлагает более удобный подход, позволяя легко создавать mock-серверы и тестировать API-контракты, в то время как тестирование чистого OkHttp требует больше усилий по настройке тестового окружения. Таким образом, Retrofit был выбран в качестве основного инструмента для сетевых запросов благодаря простоте реализации типовых сценариев через аннотированные интерфейсы, автоматической сериализации/десериализации данных, гибкой системе адаптеров и конвертеров, полной совместимости с OkHttp, удобным механизмам обработки ошибок и простому процессу тестирования. При этом сохраняется возможность тонкой настройки через кастомные OkHttp-клиенты и интерцепторы, когда это необходимо для сложных случаев интеграции.

3. Средства реализации

В качестве языка программирования был выбран Kotlin, потому что это современный, лаконичный и мощный язык, который устраняет многие недостатки Java. Kotlin предлагает такие функции, как null-безопасность, корутины, data-классы и расширения, которые значительно упрощают разработку и делают код более читаемым и поддерживаемым. Кроме того, Kotlin полностью совместим с Java, что позволяет использовать существующие библиотеки и постепенно мигрировать на новый язык. Этот выбор позволяет мне создавать более эффективные, безопасные и современные приложения с минимальными усилиями. В качестве среды разработки используется Android Studio. Она предоставляет все необходимые инструменты для создания, отладки и развертывания приложений для платформы Android. В ней можно создавать макеты пользовательского интерфейса, писать код на Kotlin, проводить отладку и тестирование приложения.

Внутри Android Studio используется Gradle – это система управления зависимостями и сборкой проектов, используемая в Android Studio. С помощью Gradle будут определяться зависимости и настройки проекта. [**?**]

Для удобства разработки используется система контроля версий Git, которая используется для отслеживания изменений в файловой системе проекта. Она позволяет сохранять историю изменений, возвращаться к предыдущим версиям проекта, совместно работать над кодом с другими участниками и управлять кодом в разных ветках разработки. Git работает на уровне файловой системы и фиксирует изменения в файлах и папках, создавая так называемые ”коммиты”. Каждый коммит содержит информацию о том, какие файлы были изменены, кто и когда внес изменения, а также комментарии к изменениям. [**?**]

Для написания интерфейса был выбран Jetpack Compose, так как это современный и мощный инструмент для создания пользовательских интерфейсов в Android. Compose позволяет описывать UI прямо в коде на Kotlin, что делает разработку более интуитивной и эффективной. Благодаря своей декларативной природе, Compose упрощает создание динамических и отзывчивых интерфейсов, а также поддерживает современные функции, такие как анимации и Material Design 3. Этот выбор позволяет мне создавать красивые и производительные приложения с минимальным количеством шаблонного

кода. [**?**][**?**]

Для работы с базой данных был выбран Room, потому что это высокоуровневая библиотека, которая значительно упрощает взаимодействие с SQLite. Room автоматизирует многие процессы, такие как создание таблиц, выполнение запросов и миграции, что позволяет сосредоточиться на логике приложения, а не на низкоуровневых деталях работы с базой данных. Благодаря интеграции с LiveData и Flow, Room также упрощает работу с данными в реальном времени, что делает его идеальным выбором для современных Android-приложений.

Для работы с сетевыми запросами был выбран Retrofit, так как это одна из самых популярных и удобных библиотек для взаимодействия с REST API. Retrofit позволяет описывать API с помощью интерфейсов и аннотаций, что делает код чистым и легко читаемым. Благодаря своей интеграции с OkHttp, Retrofit обеспечивает высокую производительность и гибкость, а также поддерживает множество функций, таких как автоматическая сериализация/десериализация данных и обработка ошибок. Этот выбор позволяет мне эффективно работать с сетевыми запросами и сосредоточиться на бизнес-логике приложения.

Для тестирования сервера используется online-инструмент Postman, который позволяет отправлять HTTP-запросы к серверу и проверять ответы. Он позволяет создавать и отправлять запросы различных типов (GET, POST, PUT, DELETE и т. д.), настраивать заголовки и параметры запроса, а также автоматизировать тестирование с помощью коллекций запросов и сценариев. Это будет полезно при работе с внешним API для интеграции с расписанием.

4. Реализация

Этапы реализации могут быть представлены в виде схемы (рис. 1).

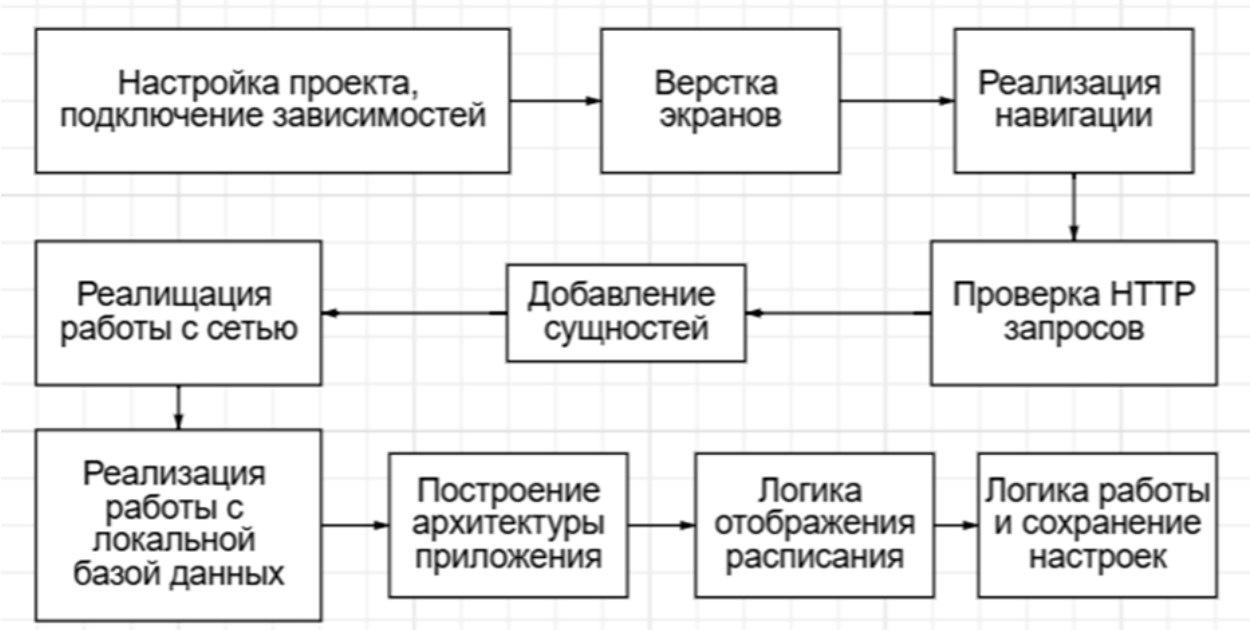

Рис. 1. Схема этапов реализации приложения

4.1. Подключение зависимостей

Для написания приложения нужно добавить необходимые библиотеки в Gradle файле на уровне приложения:

1. Retrofit – это библиотека для работы с HTTP-запросами в Android-приложениях. Она используется для взаимодействия с собственным API, обеспечивая удобный и эффективный способ отправки запросов и обработки ответов. Конвертер Gson позволяет автоматически преобразовывать JSON-ответы от API в объекты Kotlin, что упрощает работу с данными. [**?**][**?**]
2. Glide – это библиотека для загрузки и отображения изображений. Она используется для отображения картинок котиков на интерфейсе, а также их кэширования для повышения производительности. [**?**]
3. AndroidX Libraries предоставляет современные компоненты и инструменты для разработки Android-приложений, улучшая совместимость и функциональность. [**?**]
4. Jetpack Compose для декларативной верстки экранов приложения. [**?**]

5. Jetpack Navigation: обеспечивает удобный способ навигации между экранами приложения, упрощая управление компонентами и навигационными действиями.
6. Koin – это библиотека для внедрения зависимостей, которая упрощает создание и управление зависимостями в приложении, улучшая модульность и тестируемость кода.
7. Room – это библиотека для работы с базой данных SQLite, предоставляющая удобный и безопасный способ хранения и управления данными в приложении. Благодаря ей будет происходить хранение информации о треках, плейлистах и избранном.

4.2. Верстка экранов

Все экраны приложения верстаются с помощью Kotlin Declarative подхода Jetpack Compose. На этом этапе вынесены отдельно строковые и цветовые ресурсы, необходимые шрифты и векторные изображения для иконок. Для всех частей интерфейса используются стандартные UI-элементы Android SDK.[2] Приложение состоит из двух основных экранов: расписание и настройки. Смена состояния основного Activity с Bottom Bar осуществляется при помощи помещения одного из двух фрагментов в Scaffold. Расписание занятий отображается с помощью Column, где каждый элемент списка – это контейнер для события, куда записывается информация о занятии. Пока не начата работа с сетью, оставлены изображения-заглушки и тестовый текст, чтобы убедиться в корректности отображения интерфейса. [**?**] Для похожих элементов UI необходимо вынести одинаковые атрибуты в отдельный стиль для дальнейшего повторного использования. Также необходимо добавить поддержку светлой и темной темы, чтобы пользователи могли выбирать наиболее удобный для себя вариант отображения интерфейса. Ниже на рис. 2 и рис. 3 изображена итоговая верстка экранов с тестовыми данными.

4.3. Реализация навигации

На данном этапе необходимо связать сверстанные экраны в единое приложение. Для реализации логики Bottom Bar будет использоваться Jetpack Navigation. При нажатии на элементы расписания необходимо обес-

печить переход на соответствующие детальные экраны или выполнить другие действия, такие как открытие дополнительной информации о занятии. [?]

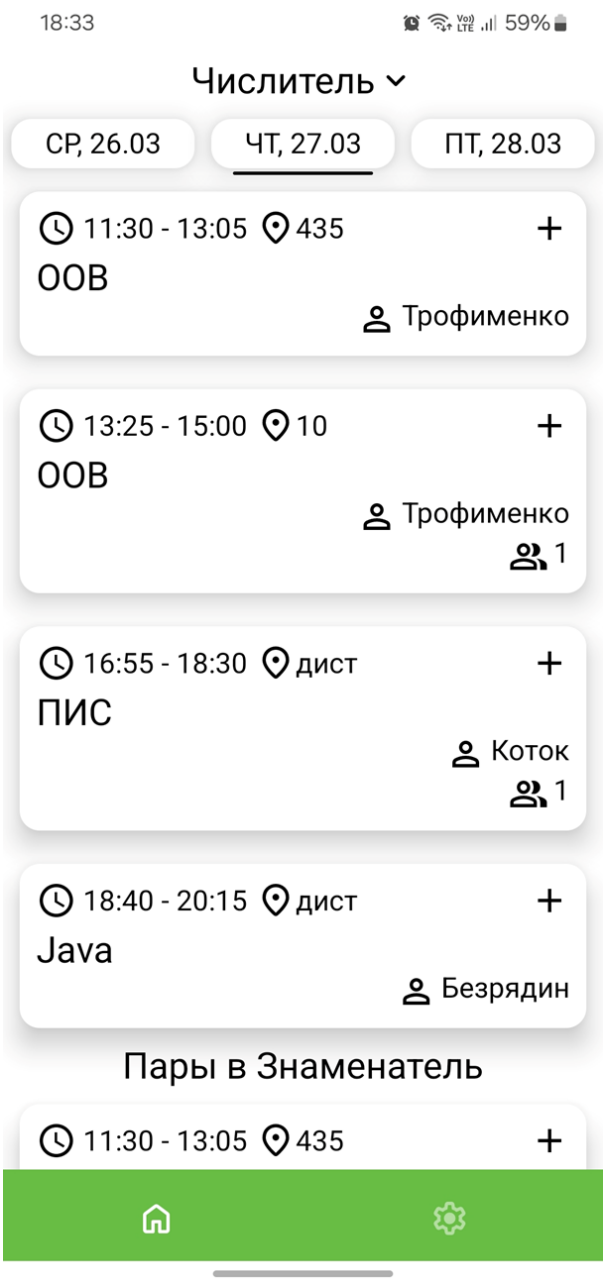

Рис. 2. Макет главного экрана

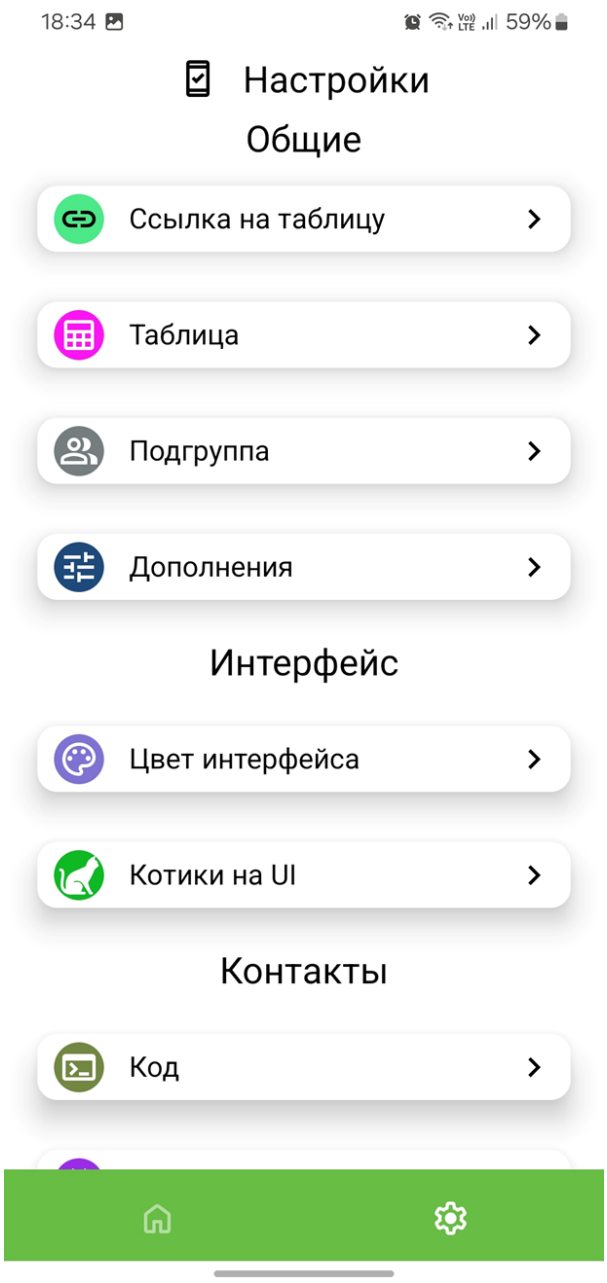

Рис. 3. Макет экрана настроек

На экране настроек при нажатии кнопки «Назад» необходимо вернуться на предыдущий экран. Это реализовано с помощью стандартных механизмов навигации в Android, например, NavController. [?] До реализации взаимодействия с базой данных полная функциональность некоторых экранов будет ограничена. Однако, основная логика навигации и переходов между экранами будет реализована для обеспечения плавного пользовательского опыта.

4.4. Проверка HTTP-запросов

Ниже, на рис. 4, рис. 5, рис. 6, рис. 7, изображены тестовые обращения для проверки запросов к собственному API через Postman. Все данные, необходимые для отображения пользователю, существуют и корректны. [**?**]

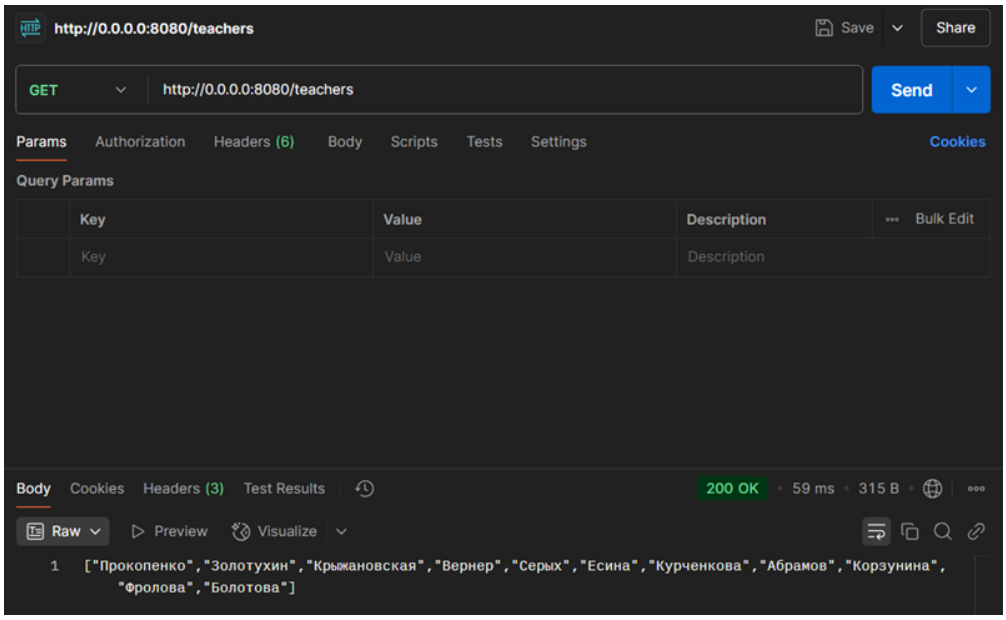

Рис. 4. Запрос получения преподавателей

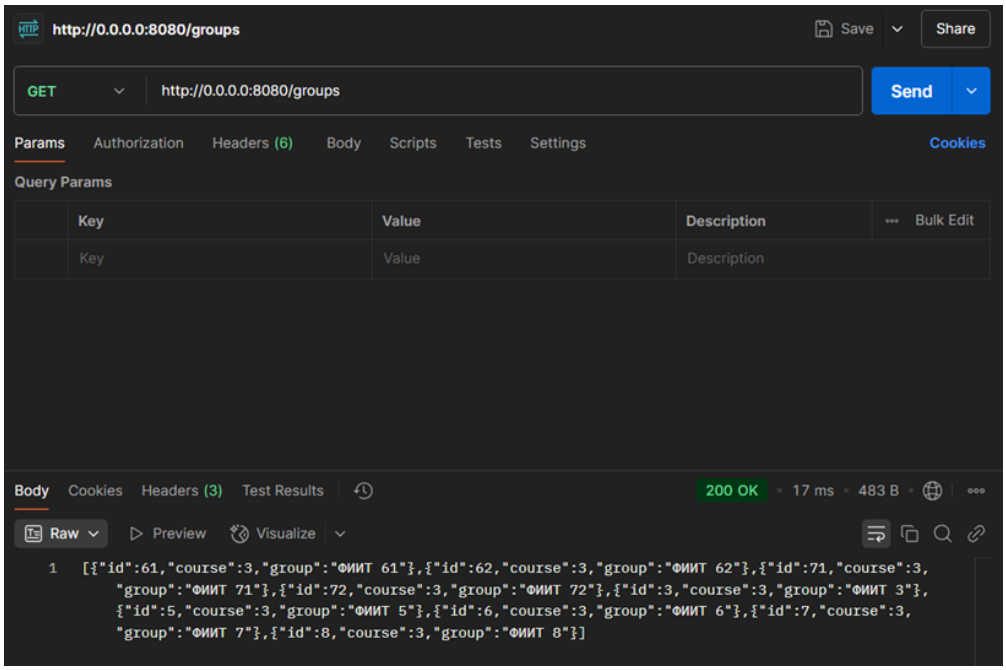

Рис. 5. Запрос получения групп

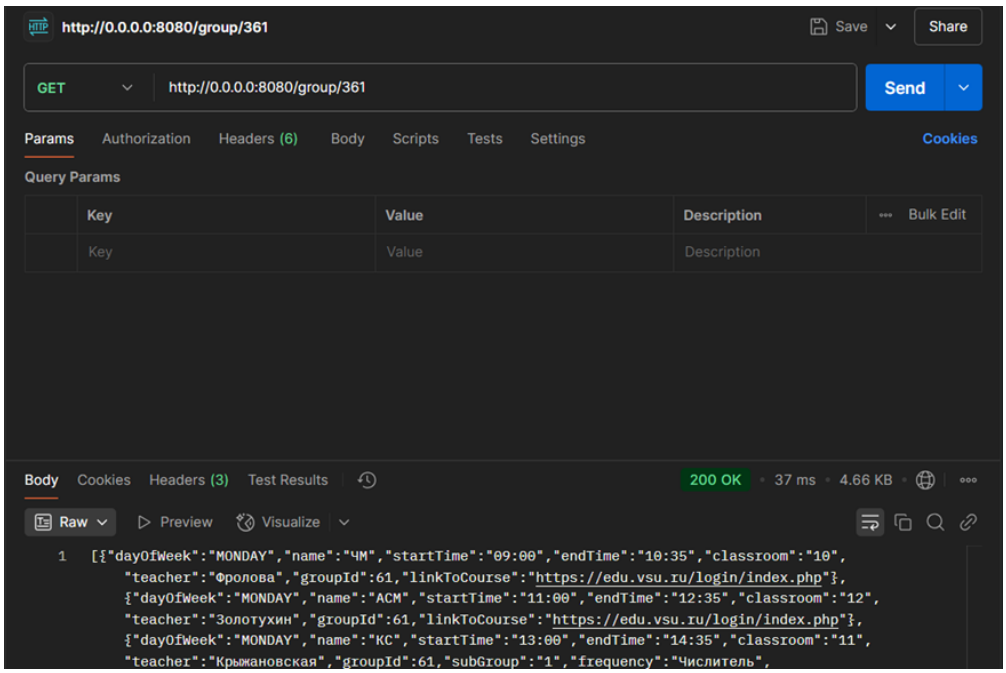

Рис. 6. Запрос получения расписания конкретной группы

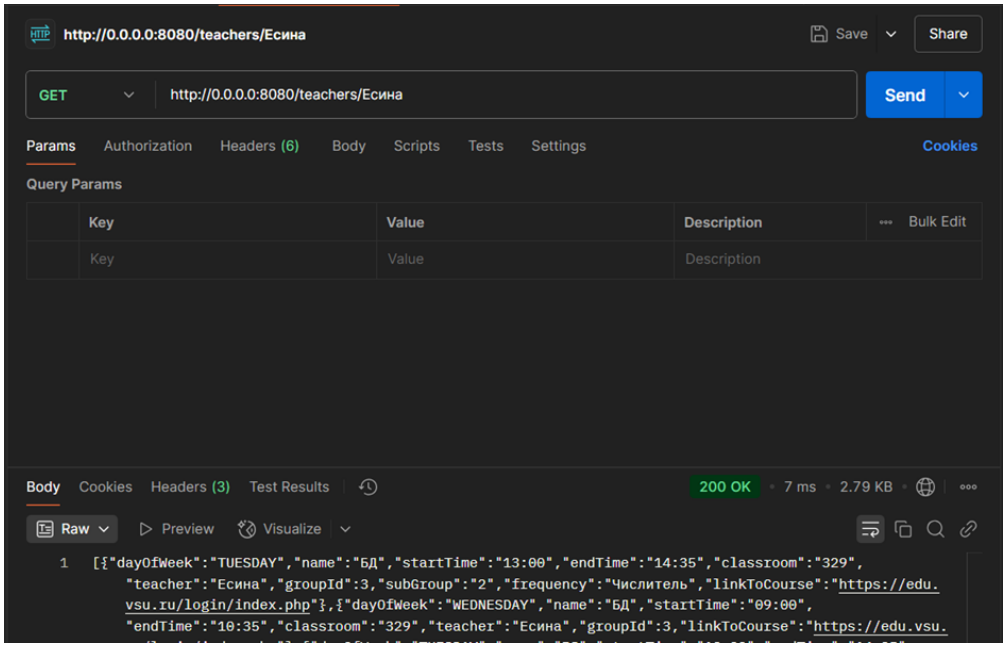

Рис. 7. Запрос получения расписания конкретного преподавателя

4.5. Модели данных

Следующие модели данных, изображенные на рис. 8, используются для конвертации в них HTTP-ответов от сервера в формат JSON и обработки для дальнейшего помещения в базу данных и отображения на интерфейсе.

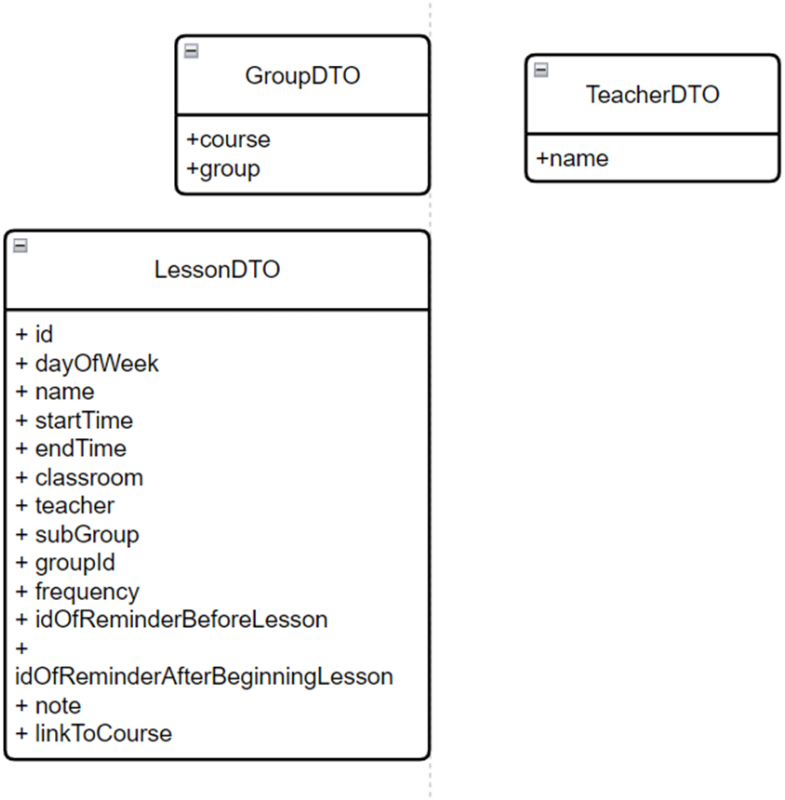

Рис. 8. Диаграмма классов для моделей данных сетевых репозиториев

1. LessonDto: модель для представления информации о занятии.
2. TeacherDto: модель для представления информации о преподавателе.
3. GroupDto: модель для представления информации о группе студентов.

4.6. Логика работы с сетью

Основная цель работы с сетью — обеспечение взаимодействия приложения с API, для получения информации о расписании по запросу пользователя. [**?**]

Для работы с сетью в приложении используются следующие ключевые компоненты, диаграмма которых изображена на рис. 9:

1. Интерфейсы LessonRemoteRepository, GroupRemoteRepository, Teacher-RemoteRepository. Определяют методы, которые принимает объект запроса и возвращает объект ответа. Эти интерфейсы позволяет абстрагироваться от конкретных деталей реализации сетевого взаимодействия, обеспечивая гибкость и возможность замены реализации при необходимости.
2. Классы LessonRemoteRepositoryImpl, GroupRemoteRepositoryImpl, TeacherRemoteRepositoryImpl. Реализуют интерфейсы Lesson-

RemoteRepository, GroupRemoteRepository, TeacherRemoteRepository и используют библиотеку Retrofit для выполнения сетевых запросов. Здесь проверяется сетевое подключение – прежде чем отправить запрос, проверяется наличие активного интернет-соединения. Если соединение отсутствует, возвращается ответ с соответствующим кодом ошибки. А также обрабатывается сам запрос: методы принимают параметры запроса и выполняют сетевой запрос к API. В случае успешного выполнения запроса возвращается ответ с кодом 200 и полученными данными. В случае ошибки (например, при отсутствии соединения или неверном типе запроса) возвращаются соответствующие коды ошибок.[**?**]

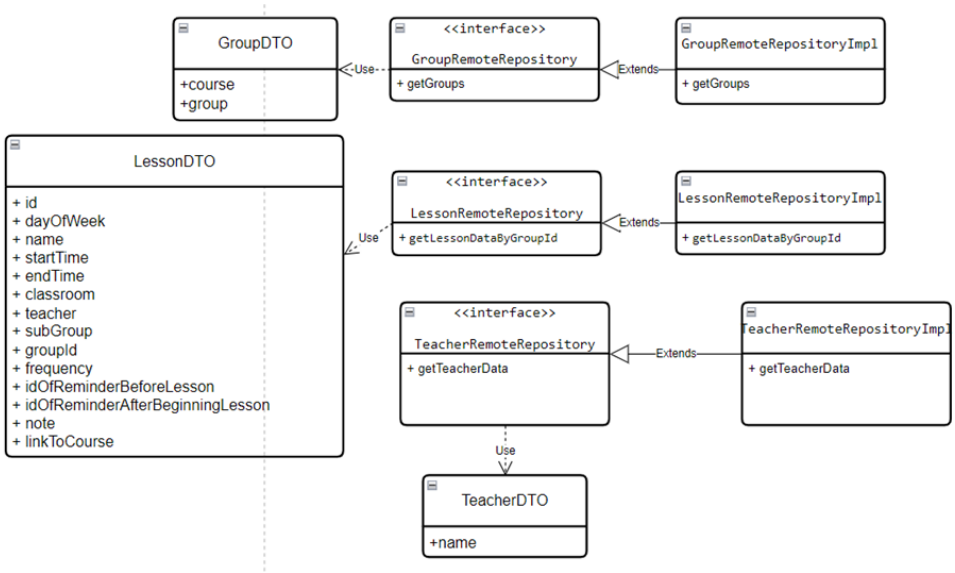

Рис. 9. Диаграмма классов для сетевых репозиториев

4.7. Логика работы с базой данных

Для обеспечения надежного и эффективного хранения данных о занятиях в приложении используется библиотека Room, которая предоставляет удобный интерфейс для работы с SQLite базой данных в Android-приложениях. Для реализации функциональности работы с базой данных в приложении используются следующие основные компоненты:

1. Entity-классы представляют собой модели данных, которые будут сохраняться в базе данных. В приложении используются следующие Entity-классы, которые изображены ниже на рис. 10:
    - LessonEntity: представляет информацию о занятии, включая его

идентификатор, название, время начала и конца, аудиторию, преподавателя и т. д.

- TeacherEntity: представляет информацию о преподавателе, включая его идентификатор, имя, должность и т. д.
- GroupEntity: представляет информацию о группе студентов, включая ее идентификатор, название и т. д.
- ReminderEntity: представляет информацию о напоминании о занятии

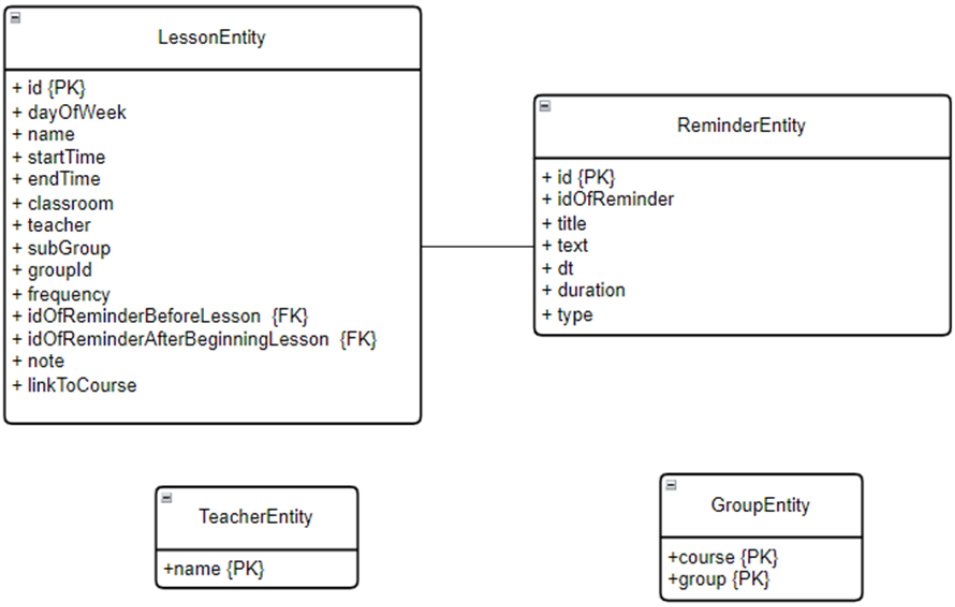

Рис. 10. Схема базы данных

2. DAO (Data Access Object) интерфейсы определяют методы для взаимодействия с базой данных. В нашем приложении используются три DAO-интерфейса, которые изображены на рис 11:
   - LessonDao: определяет методы для вставки, удаления и получения занятий из базы данных.
   - TeacherDao: определяет методы для вставки, удаления и получения преподавателей из базы данных.
   - GroupDao: определяет методы для вставки, удаления и получения групп из базы данных.

Эти интерфейсы аннотированы с использованием аннотаций Room, таких как `@Insert`, `@Delete` и `@Query`, которые определяют SQL-запросы, выполняемые для соответствующих операций. [**?**]

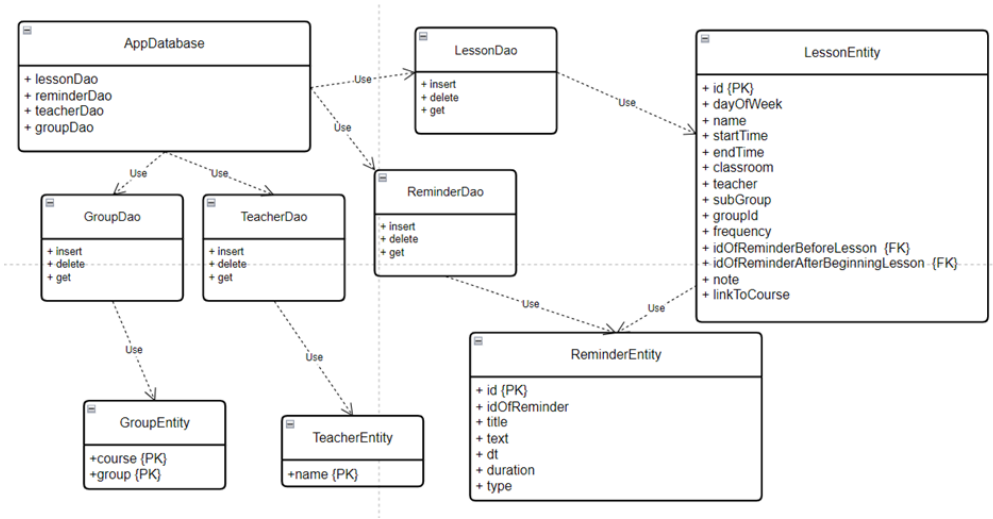

Рис. 11. Диаграмма классов базы данных

3. Абстрактный класс базы данных наследуется от `RoomDatabase` и представляет собой точку входа для взаимодействия с базой данных. В нашем приложении определены следующие классы:
    - AppDatabase: абстрактный класс для работы с таблицей занятий. Содержит абстрактные свойства `lessonDao`, `groupDao`, `teacherDao`, возвращающие экземпляры интерфейсов `LessonDao`, `GroupDao` и `TeacherDao` соответственно.
4. Клиент базы данных предоставляет методы для взаимодействия с базой данных через DAO-интерфейсы. В нашем приложении используется класс `AppDatabase_Impl`, который реализует интерфейс `AppDatabase`. Этот класс определяет методы для сохранения, удаления и получения занятий. Класс `AppDatabase_Impl` инкапсулирует логику работы с базой данных, обеспечивая удобный интерфейс для других компонентов приложения. Он использует инъекцию зависимости для получения экземпляра базы данных и обеспечивает выполнение CRUD-операций через соответствующие DAO-интерфейсы.

4.8. Архитектура приложения

Приложение будет разработано с соблюдением принципов SOLID и чистой архитектуры. Применение этих принципов обеспечивает высокую степень модульности, гибкости и тестируемости кода, а также упрощает поддержку и расширение функциональности. [**?**]

Чистая архитектура разделяет систему на несколько слоев, каждый из

которых имеет чётко определённые задачи и зависимости. Это позволяет легко изменять и масштабировать приложение, а также улучшает его тестируемость. В соответствии с чистой архитектурой в приложении выделены следующие основные слои:

1. **Внешний слой (UI Layer)**: включает все компоненты, связанные с пользовательским интерфейсом, такие как фрагменты и активности. Основная задача UI Layer - отображение данных и взаимодействие с пользователем. Этот слой получает данные из ViewModel и обновляет интерфейс в соответствии с изменениями в данных [**?**].
2. **Слой данных (Data Layer)**: содержит компоненты, связанные с управлением данными, включая DAO-интерфейсы для работы с Room, реализации сетевых запросов с использованием Retrofit, а также клиентов базы данных и сетевых клиентов. Data Layer отвечает за получение и хранение данных, предоставляя их в Domain Layer через репозитории [**?**].
3. **Слой домена (Domain Layer)**: содержит бизнес-логику и бизнес-модели. Domain Layer не зависит от других слоев и содержит основные бизнес-правила и интерфейсы репозиториев [**?**].
4. **Слой приложений (Application Layer)**: содержит ViewModel, которые объединяют бизнес-логику с логикой представления и управления состоянием. ViewModel взаимодействуют с репозиториями для получения данных и обработки пользовательских действий.

Для разделения представления и логики управления данными в приложении используется паттерн MVVM (Model-View-ViewModel). Этот паттерн обеспечивает чёткое разделение обязанностей между компонентами приложения:

1. **Model**: включает бизнес-логику и данные приложения. Модель получает данные из Data Layer и предоставляет их ViewModel.
2. **View**: состоит из пользовательского интерфейса и отвечает за отображение данных. View наблюдает за изменениями в ViewModel и обновляет интерфейс в соответствии с этими изменениями.
3. **ViewModel**: посредник между View и Model. ViewModel запрашивает данные у Model и предоставляет их View, а также обрабатывает пользовательские действия и обновляет Model.

Для управления зависимостями и их внедрения в приложении используется библиотека Koin. В модулях Koin определяются зависимости и способы их создания. Например, можно определить зависимости для сетевого клиента, базы данных и ViewModel. Это позволяет централизованно управлять зависимостями и облегчает их модификацию. Koin автоматически предоставляет экземпляры классов, когда они требуются, что упрощает тестирование и модульность. Например, ViewModel могут получать необходимые репозитории через Koin, что устраняет жёсткие зависимости[**?**]

5. Основная логика работы приложения

Основную логику работы приложения, которая изображена ниже на рис. 12, можно разделить на 2 части:

1. Просмотр информации о занятиях
2. Изменение настроек приложения

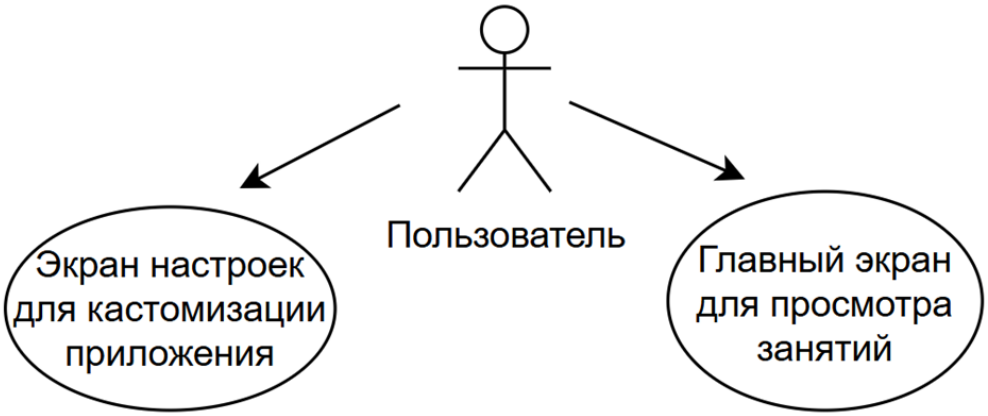

Рис. 12. Диаграмма вариантов использования приложения

5.1. Логика работы компонента «Главный экран»

Логика работы просмотра расписания в приложении реализована через несколько ключевых компонентов: интерфейсы, реализации и клиенты. Функциональность главного экрана можно увидеть на рис. 13. Эти компоненты взаимодействуют друг с другом для обеспечения функциональности отображения актуального расписания на сегодня, на другие дни, редактирования заметок. При разработке логики получения расписания важно учитывать несколько ключевых моментов, таких как получение данных от удаленного сервера, обработка ответа, управление локальными данными и обеспечение удобного пользовательского интерфейса. В нашем приложении мы используем MVVM архитектуру и DI с помощью Koin для организации кода.

Весь процесс можно разбить на следующие основные компоненты [**?**]:

1. Репозитории данных об учебных занятиях, преподавателях и группах (`LessonRemoteRepository`, `GroupRemoteRepository`, `TeacherRemoteRepository`): эти компоненты отвечают за взаимодействие с данными от удаленного сервера об учебных занятиях, преподавателях и группах. Они осуществляют запросы к удаленному серверу.
2. Локальные репозитории данных об учебных занятиях, преподавателях

и группах (`LessonLocalRepository`, `GroupLocalRepository`, `TeacherLocalRepository`): эти компоненты отвечают за взаимодействие с локальной базой данных и реализуют операции чтения/записи данных, которые приходят с сервера для кэширования и работы приложения без интернета.

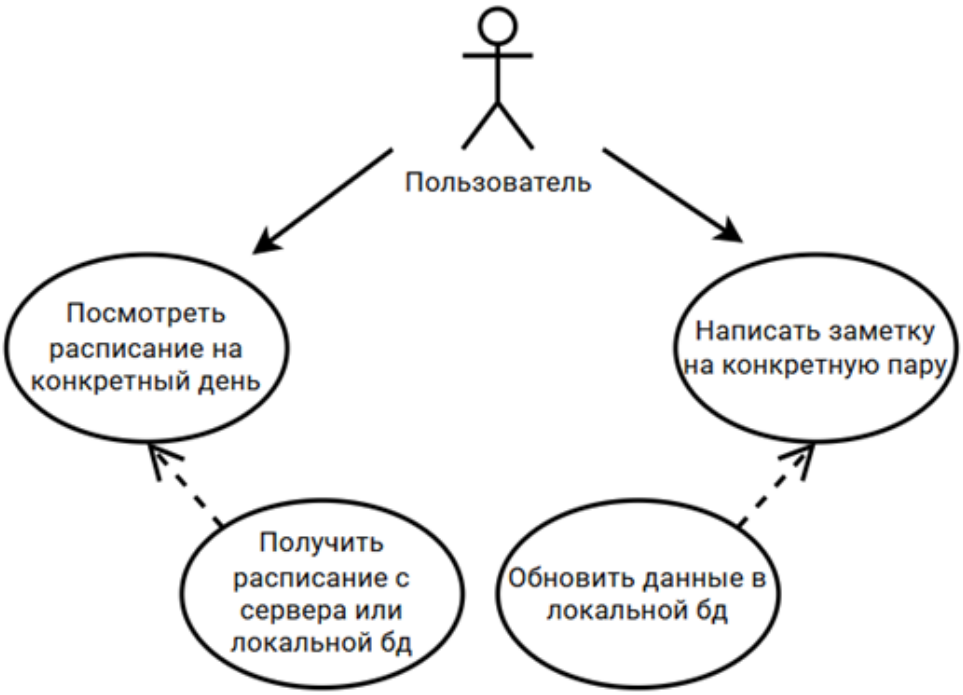

Рис. 13. Диаграмма вариантов использования компонента «Главный экран»

5.2. Логика работы компонента «Настройки»

Компонент настроек (SettingsScreen) отвечает за предоставление пользователю возможности изменять параметры приложения, такие как тема оформления, а также за доступ к некоторым внешним ресурсам, например поддержке.

Весь процесс можно разбить на следующие основные компоненты:

1. Компонент «Настройки» (`SettingsScreen`): фрагмент, отвечающий за отображение и управление настройками приложения. Он содержит интерфейс для изменения темы, отправки отзывов, просмотра пользовательского соглашения и т. д.
2. `SettingsScreenViewModel`: ViewModel, который обрабатывает бизнес-логику, связанную с настройками. Он взаимодействует с репозиторием и юзкейсами для получения и сохранения данных.

3. Менеджер настроек (`SettingsManager`): интерфейс, определяющий методы для доступа к хранилищу данных настроек. Включает методы для сохранения и получения данных настроек, а также для изменения темы.
4. Хранилище настроек (`SharedPreferences`): реализация хранилища настроек, которую использует `SettingsManager` для сохранения настроек.

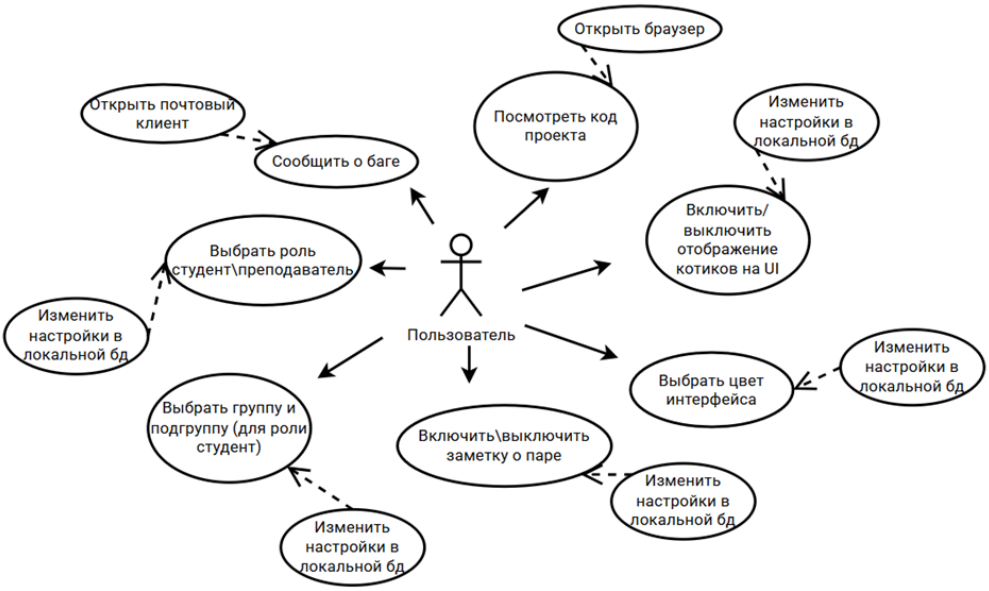

Рис. 14. Диаграмма вариантов использования компонента «Настройки»

Взаимодействие с компонентом настроек, изображенное на рис. 14, производится в соответствии со следующим планом:

- При открытии компонента на экране создается пользовательский интерфейс, и он связывается с ViewModel. Это обеспечивает связь между отображаемыми данными и бизнес-логикой. При загрузке компонента текущие настройки (например, состояние переключателя темы) запрашиваются из ViewModel. ViewModel получает данные от `SettingsManager`, который взаимодействует с хранилищем. Текущие настройки отображаются в интерфейсе.
- Пользователь взаимодействует с элементами интерфейса, такими как переключатель темы, кнопки для отправки отзывов или просмотра кода приложения, а также выбор роли/группы/подгруппы. Когда пользователь изменяет состояние переключателя темы, новое состояние

сохраняется через ViewModel и `SettingsManager`. Тема приложения немедленно обновляется, чтобы отразить новое состояние. При нажатии на соответствующие элементы интерфейса ViewModel вызывает методы интерактора для выполнения этих действий. Юзкейс использует внешний навигатор для отправки письма, открытия ссылки или браузера. После изменения настроек интерфейс обновляется, чтобы отразить текущие параметры. Например, после переключения темы приложение немедленно переключается в дневной или ночной режим.

- Все изменения настроек сохраняются в хранилище данных через репозиторий. Это гарантирует, что при следующем запуске приложения будут применены последние сохранённые настройки.

Заключение

Было разработано мобильное приложение, которое позволяет просматривать расписание университета. Также была реализована возможность добавлять заметки к учебным занятиям, изменять цвет интерфейса, сообщать об ошибках и просматривать код приложения.

Данное решение может быть масштабировано путём добавления расширенной функциональности. В первую очередь это настройка персонализированного отображения занятий, например, не списком, а в виде таблицы. Также может быть добавлена система аккаунтов для сохранения настроек на сервере, что позволит синхронизировать их между устройствами одного пользователя.

Список используемых источников

Приложение 1

Главный класс приложения

```
class MainActivity : ComponentActivity() {
    private val viewModel: MainActivityViewModel by viewModel()


    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MainTheme {
                viewModel.navController = rememberNavController()
                RequestNotificationPermission()
                NavigationRoot(viewModel = viewModel)
            }
        }
    }
}
```

Приложение 2

Компонент с логикой для главного класса приложения

```
internal class MainActivityViewModel(
    private val getLatestReleaseUseCase: GetLatestReleaseUseCase,
    private val rebootingRemindersUseCase: RebootingRemindersUseCase,
    context: Context,
    private val settingsManager: SettingsManager,
    private val refresher: Refresher,
) : ViewModel(), BottomBarNavigator {

    var release: Release? = null
        private set

    val versionCode = context.getVersionCode()
    val versionName = context.getVersionName()

    suspend fun checkUpdates(): Release? {
        val localRelease = getLatestReleaseUseCase.invoke()
        return if (localRelease != null && versionName != null &&
            localRelease.tagName.substringAfter("v") != versionName
        ) {
            saveSettingsToSharedPreferences {
                it.copy(releaseBody = localRelease.body)
            }
            release = localRelease
            release
        } else null
    }

    fun update() {
        release?.let {
            viewModelScope.launch(Dispatchers.IO) {
                refresher.refresh(viewModelScope, it.apkUrl, RefresherInfo.
                    ↪ APK_FILE_NAME)
            }
        }
    }

    val isUpdateInProcessFlow: SharedFlow<Boolean> = refresher.progressFlow.
```

```
    ↪ map {
    it !is DownloadStatus.Error && it !is DownloadStatus.Success
}.shareIn(viewModelScope, SharingStarted.Lazily)

val percentageFlow: SharedFlow<Float> = refresher.progressFlow.map {
    when (it) {
        is DownloadStatus.InProgress -> it.percentage
        DownloadStatus.Success -> 100f
        else -> 0f
    }
}.shareIn(viewModelScope, SharingStarted.Lazily)

val settings = settingsManager.settingsFlow

fun saveSettingsToSharedPreferences(saving: (Settings) -> Settings) {
    settingsManager.save(saving)
}

private fun registerGeneralExceptionCallback(context: Context) {
    Thread.setDefaultUncaughtExceptionHandler { thread, throwable ->
        context.copyTextToClipBoard(
            label = "Error",
            text = "Contact us about this problem: ${Link.EMAIL}\n\n
                ↪ Exception in ${thread.name} thread\n${throwable.
                ↪ stackTraceToString()}"
        )
        Log.e("GeneralException", "Exception in thread ${thread.name}",
            ↪ throwable)
        exitProcess(0)
    }
}

init {
    registerGeneralExceptionCallback(context = context)
}

val startScreen = Screen.Main

override var navController: NavHostController? = null

private val _currentScreen: MutableStateFlow<Screen> = MutableStateFlow(
```

```
        ↪ startScreen)
    override val currentScreen: StateFlow<Screen> = _currentScreen.
        ↪ asStateFlow()

    private fun emitCurrentScreen() {
        viewModelScope.launch(Dispatchers.Main) {
            navController?.currentBackStackEntryFlow?.collect {
                ↪ backStackEntry ->
                val destination = backStackEntry.destination
                when {
                    destination.hasRoute(Screen.Main::class) -> _currentScreen
                        ↪ .emit(Screen.Main)
                    destination.hasRoute(Screen.Settings::class) ->
                        ↪ _currentScreen.emit(Screen.Settings)
                }
            }
        }
    }

    override fun back() {
        navController?.popBackStack()
        emitCurrentScreen()
    }

    override fun navigateTo(screen: Screen) {
        if (screen != Screen.Main)
            navController?.navigate(screen)
        else navController?.popBackStack()
        emitCurrentScreen()
    }
}
```

Приложение 3

Главный экран

```
@Composable
internal fun MainScreen(bottomBarNavigator: BottomBarNavigator) {
    val lifecycleState by LocalLifecycleOwner.current.lifecycle.
        ↪ currentStateFlow.collectAsState()
    val viewModel: MainScreenViewModel = koinViewModel()
    val context = LocalContext.current

    val settings by viewModel.settingsFlow.collectAsState()
    val defaultColor = Theme.colors.mainColor

    val dark = isSystemInDarkTheme()
    val mainColor by remember {
        mutableStateOf(
            settings.getMainColorForThisTheme(isDark = dark) ?: defaultColor
        )
    }

    var isFrequencyInChanging by remember { mutableStateOf(false) }
    val cor = rememberCoroutineScope()
    var localTime by remember { mutableStateOf(LocalTime.now()) }
    var localDate by remember { mutableStateOf(LocalDate.now()) }

    val weekOfYear by remember {
        derivedStateOf {
            localDate.getFrequencyByLocalDate()
                .changeFrequencyIfDefinedInSettings(settings = settings)
        }
    }

    val lessons by viewModel.lessonsFlow.collectAsState(listOf())
    val cardsWithDateState = rememberLazyListState()

    fun chooseTypeOfDefinitionFrequencyDependsOn(selectedFrequency:
        ↪ Frequency?) {
        viewModel.saveSettingsToSharedPreferences {
            it.copy(isSelectedFrequencyCorrespondsToTheWeekNumbers =
                ↪ selectedFrequency?.let { localDate.getFrequencyByLocalDate
```

```
            ↪ () == it })
    }
    isFrequencyInChanging = false
}

val pagerState = rememberPagerState(pageCount = { viewModel.pageNumber
    ↪ }, initialPage = 0)

BackHandler {
    when {
        pagerState.currentPage == 0 -> {
            bottomBarNavigator.back()
            (context as Activity).finish()
        }

        else -> {
            cor.launch(Dispatchers.Main) {
                pagerState.animateScrollToPage(0)
            }
        }
    }
}

LaunchedEffect(key1 = null) {
    withContext(Dispatchers.Main) {
        while (true) {
            localTime = LocalTime.now()
            delay(timeMillis = 1000L)
        }
    }
}

LaunchedEffect(key1 = pagerState.currentPage) {
    localDate = viewModel.todayDate.plusDays(pagerState.currentPage.
        ↪ toLong())
    cardsWithDateState.animateScrollToItem(if (pagerState.currentPage >
        ↪ 0) pagerState.currentPage - 1 else pagerState.currentPage)
}

LaunchedEffect(lifecycleState) {
    when (lifecycleState) {
```

```
            Lifecycle.State.DESTROYED -> {}
            Lifecycle.State.INITIALIZED -> {}
            Lifecycle.State.CREATED -> {}
            Lifecycle.State.STARTED -> {}
            Lifecycle.State.RESUMED -> {}
        }
    }

    Column(modifier = Modifier.fillMaxSize()) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = 10.dp),
            horizontalArrangement = Arrangement.Center,
            verticalAlignment = Alignment.CenterVertically,
        ) {
            Box {
                Row(
                    verticalAlignment = Alignment.CenterVertically,
                    modifier = Modifier.clickable { isFrequencyInChanging =
                        ↪ true }) {
                    Text(
                        text = stringResource(id = weekOfYear.resourceName),
                        fontSize = FontSize.big22,
                        color = Theme.colors.oppositeTheme,
                    )
                    Icon(
                        painter = painterResource(id = if (
                            ↪ isFrequencyInChanging) R.drawable.
                            ↪ keyboard_arrow_up else R.drawable.
                            ↪ keyboard_arrow_down),
                        contentDescription = stringResource(R.string.
                            ↪ icon_fold_or_unfold_list_with_frequency),
                        tint = Theme.colors.oppositeTheme,
                    )
                }

                DropdownMenu(
                    modifier = Modifier
                        .background(Theme.colors.singleTheme)
                        .border(BorderStroke(width = 2.dp, color = Theme.
```

```
            ↪ colors.oppositeTheme)),
        expanded = isFrequencyInChanging,
        onDismissRequest = { isFrequencyInChanging = false },
    ) {
        DropdownMenuItem(
            text = {
                Row {
                    TextForThisTheme(
                        text = stringResource(id = Frequency.
                            ↪ Numerator.resourceName),
                        fontSize = FontSize.medium19
                    )
                    if (settings.
                        ↪ isSelectedFrequencyCorrespondsToTheWeekNumbers
                        ↪  != null
                        && weekOfYear == Frequency.Numerator
                    )
                        Icon(
                            painterResource(id = R.drawable.done),
                            contentDescription = stringResource(R.
                                ↪ string.this_is_selected_or_not),
                            tint = Theme.colors.oppositeTheme
                        )
                }
            },
            onClick = { chooseTypeOfDefinitionFrequencyDependsOn(
                ↪ selectedFrequency = Frequency.Numerator) })

        Spacer(
            modifier = Modifier
                .height(2.dp)
                .padding(horizontal = 15.dp)
                .background(color = Theme.colors.oppositeTheme)
        )

        DropdownMenuItem(
            text = {
                Row {
                    TextForThisTheme(
                        text = stringResource(id = Frequency.
                            ↪ Denominator.resourceName),
```

```
                    fontSize = FontSize.medium19
                )
                if (settings.
                    ↪ isSelectedFrequencyCorrespondsToTheWeekNumbers
                    ↪  != null
                    && weekOfYear == Frequency.Denominator
                )
                    Icon(
                        painterResource(id = R.drawable.done),
                        contentDescription = stringResource(R.
                            ↪ string.this_is_selected_or_not),
                        tint = Theme.colors.oppositeTheme
                    )
            }
        },
        onClick = { chooseTypeOfDefinitionFrequencyDependsOn(
            ↪ selectedFrequency = Frequency.Denominator) })

    Spacer(
        modifier = Modifier
            .height(2.dp)
            .padding(horizontal = 15.dp)
            .background(color = Theme.colors.oppositeTheme)
    )

    DropdownMenuItem(
        text = {
            Row {
                TextForThisTheme(
                    text = stringResource(id = R.string.auto),
                    fontSize = FontSize.medium19
                )
                if (settings.
                    ↪ isSelectedFrequencyCorrespondsToTheWeekNumbers
                    ↪  == null)
                    Icon(
                        painterResource(id = R.drawable.done),
                        contentDescription = stringResource(R.
                            ↪ string.this_is_selected_or_not),
                        tint = Theme.colors.oppositeTheme
                    )
```

```
                    }
                },
                onClick = { chooseTypeOfDefinitionFrequencyDependsOn(
                    ↪ selectedFrequency = null) })
            }
        }
    }

    LazyRow(
        state = cardsWithDateState,
        modifier = Modifier.fillMaxWidth(),
        horizontalArrangement = Arrangement.SpaceEvenly,
    ) {
        items(count = viewModel.pageNumber) { index ->
            val day by remember { mutableStateOf(viewModel.todayDate.
                ↪ plusDays(index.toLong())) }
            Column {
                Card(
                    modifier = Modifier
                        .fillParentMaxWidth(1 / 3f)
                        .padding(horizontal = 5.dp),
                    colors = CardDefaults.cardColors(
                        containerColor = if (day == viewModel.todayDate)
                            ↪ mainColor else Theme.colors.buttonColor,
                        contentColor = (if (day == viewModel.todayDate)
                            ↪ mainColor else Theme.colors.buttonColor).
                            ↪ suitableColor(),
                    ),
                    elevation = CardDefaults.cardElevation(
                        ↪ defaultElevation = 10.dp),
                ) {
                    Text(
                        text = day.getDateStringWithWeekOfDay(context =
                            ↪ context),
                        fontSize = FontSize.small17,
                        modifier = Modifier
                            .fillMaxWidth()
                            .clickable {
                                cor.launch(Dispatchers.Main) {
                                    pagerState.animateScrollToPage(index)
                                }
```

```
                    }
                        .padding(vertical = 5.dp, horizontal = 10.dp),
                    textAlign = TextAlign.Center,
                )
            }

            if (day == localDate)
                Card(
                    modifier = Modifier
                        .fillParentMaxWidth(1 / 3f)
                        .padding(top = 2.dp)
                        .padding(horizontal = 18.dp)
                        .height(2.dp),
                    colors = CardDefaults.cardColors(containerColor =
                        ↪ Theme.colors.oppositeTheme)
                ) {}
        }
    }
}

HorizontalPager(
    state = pagerState,
    modifier = Modifier
        .fillMaxWidth()
        .weight(10f),
) { page ->
    val dateOfThisLesson = viewModel.todayDate.plusDays(page.toLong()
        ↪ )
    val weekOfYearOfThisDay = dateOfThisLesson.
        ↪ getFrequencyByLocalDate()
        .changeFrequencyIfDefinedInSettings(settings = settings)

    val lessonsOfThisDay = lessons.filter {
        it.dayOfWeek == dateOfThisLesson.dayOfWeek &&
                (it.frequency == null || it.frequency ==
                    ↪ weekOfYearOfThisDay) &&
                if (settings.role == Role.Student) (it.subGroup ==
                    ↪ settings.subgroup || settings.subgroup == null
                    ↪ || it.subGroup == null)
                else it.teacher == settings.teacherName
    }.sorted()
```

```
val lessonsInOppositeNumAndDenDay = lessons.filter {
    it.dayOfWeek == dateOfThisLesson.dayOfWeek &&
            it.frequency == weekOfYearOfThisDay.getOpposite() &&
            if (settings.role == Role.Student) (it.subGroup ==
                ↪ settings.subgroup || settings.subgroup == null
                ↪ || it.subGroup == null)
            else it.teacher == settings.teacherName
}.sorted()

Box {
    Column(
        modifier = Modifier
            .fillMaxSize()
            .verticalScroll(rememberScrollState())
            .padding(horizontal = 10.dp),
    ) {
        if (lessonsOfThisDay.isNotEmpty()) {
            viewModel.nowIsLesson = false

            lessonsOfThisDay.forEach { lesson ->
                if (lesson.nowIsLesson(localTime) && viewModel.
                    ↪ todayDate == dateOfThisLesson) {
                    viewModel.nowIsLesson = true

                    lesson.StringForSchedule(
                        colorBack = mainColor,
                        dateOfThisLesson = dateOfThisLesson,
                        viewModel = viewModel,
                        isNoteAvailable = settings.notesAboutLesson,
                        isNotificationsAvailable = settings.
                            ↪ notificationsAboutLesson,
                    )
                } else if (viewModel.todayDate == dateOfThisLesson
                    ↪ && lessonsOfThisDay.any { it.startTime >
                    ↪ localTime } && lesson == lessonsOfThisDay.
                    ↪ filter { it.startTime > localTime }[0] && !
                    ↪ viewModel.nowIsLesson) {
                    CardOfNextLesson(colorOfCard = mainColor) {
                        lesson.StringForSchedule(
                            paddingValues = PaddingValues(),
```

```
                                colorBack = Theme.colors.buttonColor,
                                dateOfThisLesson = dateOfThisLesson,
                                viewModel = viewModel,
                                isNoteAvailable = settings.
                                    ↪ notesAboutLesson,
                                isNotificationsAvailable = settings.
                                    ↪ notificationsAboutLesson,
                            )
                        }
                    } else lesson.StringForSchedule(
                        colorBack = Theme.colors.buttonColor,
                        dateOfThisLesson = dateOfThisLesson,
                        viewModel = viewModel,
                        isNoteAvailable = settings.notesAboutLesson,
                        isNotificationsAvailable = settings.
                            ↪ notificationsAboutLesson,
                    )
                }
            } else WeekDay(
                isCatShowed = settings.weekendCat,
                viewModel = viewModel,
                context = context,
                modifier = Modifier.let {
                    var modifier =
                        Modifier.padding(vertical = 100.dp); if (
                            ↪ lessonsInOppositeNumAndDenDay.isEmpty())
                            ↪ modifier =
                    Modifier.weight(1f); modifier
                })

            if (lessonsInOppositeNumAndDenDay.isNotEmpty()) {
                TextForThisTheme(
                    text = "${stringResource(id = R.string.
                        ↪ other_lessons_in_this_day)} ${
                            stringResource(
                                id = weekOfYearOfThisDay.getOpposite().
↪ resourceName
                            )
                        }",
                    modifier = Modifier.align(Alignment.
                        ↪ CenterHorizontally),
```

```
                        fontSize = FontSize.big22,
                    )

                    lessonsInOppositeNumAndDenDay.forEach { lesson ->
                        lesson.StringForSchedule(
                            colorBack = Theme.colors.buttonColor,
                            dateOfThisLesson = null,
                            viewModel = viewModel,
                            isNoteAvailable = settings.notesAboutLesson,
                            isNotificationsAvailable = settings.
                                ↪ notificationsAboutLesson,
                        )
                    }
                }
            }


        }
    }
}
```

Приложение 4

Экран настроек

```
@Composable
internal fun SettingsScreen(bottomBarNavigator: BottomBarNavigator) {
    val viewModel: SettingsScreenViewModel = koinViewModel()
    val context = LocalContext.current
    val dark = isSystemInDarkTheme()
    val subgroupList by viewModel.subgroupFlow.collectAsState(listOf())
    val groupList by viewModel.groupFlow.collectAsState(listOf())
    val teachersList by viewModel.teachersFlow.collectAsState(listOf())
    val settings by viewModel.settings.collectAsState()

    var colorIsEditable by remember { mutableStateOf(false) }
    var isFeaturesEditable by remember { mutableStateOf(false) }
    var isGroupChanging by remember { mutableStateOf(false) }
    var isSubGroupChanging by remember { mutableStateOf(false) }
    var isTeacherChanging by remember { mutableStateOf(false) }
    var catsOnUIIsChanging by remember { mutableStateOf(false) }
    var isRoleChanging by remember { mutableStateOf(false) }
    val networkState by viewModel.gSheetsServiceRequestStatusFlow.
        ↪ collectAsState()

    BackHandler {
        bottomBarNavigator.back()
    }

    Column(
        modifier = Modifier.fillMaxSize()
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = 10.dp),
            horizontalArrangement = Arrangement.Center,
            verticalAlignment = Alignment.CenterVertically
        ) {
            Row {
                Icon(
                    painter = painterResource(
```

```
                id = getIconByRequestStatus(
                    networkState = networkState
                )
            ),
            contentDescription = stringResource(R.string.
                ↪ icon_data_updating_state),
            tint = Theme.colors.oppositeTheme
        )
        Spacer(modifier = Modifier.width(15.dp))
        TextForThisTheme(
            text = stringResource(R.string.settings),
            fontSize = FontSize.big22
        )
    }
}

if (colorIsEditable) {
    ColorPickerDialog(
        context = context,
        firstColor = settings.getMainColorForThisTheme(isDark = dark)
            ?: Theme.colors.mainColor,
        onDismissRequest = { colorIsEditable = false }
    ) {
        viewModel.saveSettingsToSharedPreferences { settings ->
            if (dark) settings.copy(
                darkThemeColor = it
            ) else settings.copy(lightThemeColor = it)
        }
    }
}

Column(
    modifier = Modifier
        .weight(1f)
        .padding(horizontal = 20.dp)
        .verticalScroll(rememberScrollState())
) {
    Box(modifier = Modifier.fillMaxWidth()) {
        TextForThisTheme(
            modifier = Modifier.align(Alignment.Center),
            fontSize = FontSize.big22,
```

```
            text = stringResource(R.string.general)
        )
        if (settings.catInSettings) {
            GifPlayer(
                size = 80.dp,
                modifier = Modifier
                    .align(Alignment.CenterEnd)
                    .clickable { viewModel.meow() },
                imageUri = Uri.parse(AssetsInfo.FUNNY_SETTINGS_CAT)
            )
        }
    }

    CardOfSettings(
        text = stringResource(R.string.role),
        icon = {
            Icon(
                painter = painterResource(id = R.drawable.role),
                contentDescription = stringResource(R.string.role),
                tint = it.suitableColor()
            )
        },
        onClick = {
            isRoleChanging = !isRoleChanging
        },
        additionalContentIsVisible = isRoleChanging,
        additionalContent = {
            LazyRow(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = it)
            ) {
                items(Role.entries) { role ->
                    AssistChip(
                        leadingIcon = {
                            if (role == settings.role) {
                                Icon(
                                    painter = painterResource(id = R.
                                        ↪ drawable.done),
                                    contentDescription = stringResource(R
                                        ↪ .string.this_is_user_role),
```

```
                                tint = Theme.colors.oppositeTheme
                            )
                        }
                    },
                    modifier = Modifier.padding(horizontal = 3.dp),
                    onClick = {
                        viewModel.saveSettingsToSharedPreferences {
                            ↪ settings ->
                            settings.copy(
                                role = role
                            ).let { s ->
                                if (role == Role.Teacher) s.copy(
                                    subgroup = null,
                                    groupId = null
                                ) else s.copy(teacherName = null)
                            }
                        }
                        viewModel.sync()
                    },
                    label = { TextForThisTheme(text = stringResource
                        ↪ (role.resourceName)) }
                )
            }

        }
    }
)

if (groupList.isNotEmpty() && settings.role == Role.Student) {
    CardOfSettings(
        text = stringResource(R.string.group),
        icon = {
            Icon(
                painter = painterResource(id = R.drawable.group),
                contentDescription = stringResource(R.string.group)
                    ↪ ,
                tint = it.suitableColor()
            )
        },
        onClick = { isGroupChanging = !isGroupChanging },
        additionalContentIsVisible = isGroupChanging,
```

```
        additionalContent = {
            LazyRow(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = it)
            ) {
                items(groupList) { group ->

                    AssistChip(
                        leadingIcon = {
                            if (group.id == settings.groupId) {
                                Icon(
                                    painter = painterResource(id = R.
                                        ↪ drawable.done),
                                    contentDescription =
                                        ↪ stringResource(R.string.
                                        ↪ this_is_user_group),
                                    tint = Theme.colors.oppositeTheme
                                )
                            }
                        },
                        modifier = Modifier.padding(horizontal = 3.
                            ↪ dp),
                        onClick = {
                            viewModel.saveSettingsToSharedPreferences
                                ↪  { settings ->
                                settings.copy(
                                    groupId = if (settings.groupId !=
                                        ↪  group.id) group.id else
                                        ↪ null
                                )
                            }
                            viewModel.sync()
                        },
                        label = { TextForThisTheme(text = group.
                            ↪ groupCourseString()) }
                    )
                }
            }
        }
    )
```

```
}

if (subgroupList.isNotEmpty() && settings.role == Role.Student) {
    CardOfSettings(
        text = stringResource(R.string.subgroup),
        icon = {
            Icon(
                painter = painterResource(id = R.drawable.subgroup)
                    ↪ ,
                contentDescription = stringResource(R.string.
                    ↪ subgroup),
                tint = it.suitableColor()
            )
        },
        onClick = { isSubGroupChanging = !isSubGroupChanging },
        additionalContentIsVisible = isSubGroupChanging,
        additionalContent = {
            LazyRow(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = it)
            ) {
                items(subgroupList) { subgroup ->
                    AssistChip(
                        leadingIcon = {
                            if (subgroup == settings.subgroup) {
                                Icon(
                                    painter = painterResource(id = R.
                                        ↪ drawable.done),
                                    contentDescription =
                                        ↪ stringResource(R.string.
                                        ↪ this_is_user_subgroup),
                                    tint = Theme.colors.oppositeTheme
                                )
                            }
                        },
                        modifier = Modifier.padding(horizontal = 3.
                            ↪ dp),
                        onClick = {
                            viewModel.saveSettingsToSharedPreferences
                                ↪  { settings ->
```

```
                                settings.copy(
                                    subgroup = if (settings.subgroup
                                        ↪ != subgroup) subgroup else
                                        ↪ null
                                )
                            }
                        },
                        label = { TextForThisTheme(text = subgroup)
                            ↪ }
                    )
                }
            }
        }
    )
}

if (teachersList.isNotEmpty() && settings.role == Role.Teacher) {
    CardOfSettings(
        text = stringResource(R.string.teacher),
        icon = {
            Icon(
                painter = painterResource(id = R.drawable.teacher),
                contentDescription = stringResource(R.string.
                    ↪ icon_teacher),
                tint = it.suitableColor()
            )
        },
        onClick = { isTeacherChanging = !isTeacherChanging },
        additionalContentIsVisible = isTeacherChanging,
        additionalContent = {
            LazyRow(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(horizontal = it)
            ) {
                items(teachersList) { teacher ->
                    AssistChip(
                        leadingIcon = {
                            if (teacher.name == settings.teacherName
                                ↪ ) {
                                Icon(
```

```
                                painter = painterResource(id = R.
                                    ↪ drawable.done),
                                contentDescription =
                                    ↪ stringResource(R.string.
                                    ↪ selected_teacher),
                                tint = Theme.colors.oppositeTheme
                            )
                        }
                    },
                    modifier = Modifier.padding(horizontal = 3.
                        ↪ dp),
                    onClick = {
                        viewModel.saveSettingsToSharedPreferences
                            ↪  { settings ->
                            settings.copy(
                                teacherName = if (settings.
                                    ↪ teacherName != teacher.name
                                    ↪ ) teacher.name else null
                            )
                        }
                        viewModel.sync()
                    },
                    label = { TextForThisTheme(text = teacher.
                        ↪ name) }
                )
            }
        }
    )
}

CardOfSettings(
    text = stringResource(id = R.string.features),
    icon = {
        Icon(
            painter = painterResource(id = R.drawable.tune),
            contentDescription = stringResource(R.string.features)
                ↪ ,
            tint = it.suitableColor()
        )
    },
```

```
        onClick = { isFeaturesEditable = !isFeaturesEditable },
        additionalContentIsVisible = isFeaturesEditable
    ) {
        FeatureOfSettings(
            onClick = {
                viewModel.saveSettingsToSharedPreferences { settings
                    ↪ ->
                    settings.copy(notificationsAboutLesson = !settings.
                        ↪ notificationsAboutLesson)
                }
            },
            padding = it,
            text = stringResource(R.string.
                ↪ notification_about_lesson_before_time),
            checked = settings.notificationsAboutLesson
        )
        FeatureOfSettings(
            onClick = {
                viewModel.saveSettingsToSharedPreferences { settings
                    ↪ ->
                    settings.copy(
                        notesAboutLesson = !settings.notesAboutLesson
                    )
                }
            },
            padding = it,
            text = stringResource(R.string.note),
            checked = settings.notesAboutLesson
        )
    }

    TextForThisTheme(
        modifier = Modifier
            .padding(10.dp)
            .align(Alignment.CenterHorizontally),
        fontSize = FontSize.big22,
        text = stringResource(R.string.interface_str)
    )

    CardOfSettings(
        text = stringResource(R.string.interface_color),
```

```
            icon = {
                Icon(
                    painter = painterResource(id = R.drawable.palette),
                    contentDescription = stringResource(R.string.
                        ↪ change_color_of_interface),
                    tint = it.suitableColor()
                )
            },
            onClick = { colorIsEditable = true }
        )

        CardOfSettings(
            text = stringResource(R.string.cats_on_ui),
            icon = {
                Icon(
                    painter = painterResource(id = R.drawable.cat),
                    contentDescription = stringResource(R.string.
                        ↪ cats_in_interface),
                    tint = it.suitableColor()
                )
            },
            onClick = { catsOnUIIsChanging = !catsOnUIIsChanging },
            additionalContentIsVisible = catsOnUIIsChanging,
            additionalContent = {
                Column {
                    FeatureOfSettings(
                        onClick = {
                            viewModel.saveSettingsToSharedPreferences {
                                ↪ settings ->
                                settings.copy(weekendCat = !settings.
                                    ↪ weekendCat)
                            }
                        },
                        padding = it,
                        text = stringResource(R.string.weekend_cat),
                        checked = settings.weekendCat
                    )
                    FeatureOfSettings(
                        onClick = {
                            viewModel.saveSettingsToSharedPreferences {
                                ↪ settings ->
```

```
                        settings.copy(catInSettings = !settings.
                            ↪ catInSettings)
                    }
                },
                padding = it,
                text = stringResource(R.string.cat_in_settings),
                checked = settings.catInSettings
            )
        }
    }
)

TextForThisTheme(
    modifier = Modifier
        .padding(10.dp)
        .align(Alignment.CenterHorizontally),
    fontSize = FontSize.big22,
    text = stringResource(R.string.contacts)
)

CardOfSettings(
    text = stringResource(R.string.code),
    icon = {
        Icon(
            painter = painterResource(id = R.drawable.terminal),
            contentDescription = stringResource(R.string.view_code
                ↪ ),
            tint = it.suitableColor()
        )
    },
    onClick = { context.openLink(link = Link.CODE) }
)

CardOfSettings(
    text = stringResource(R.string.report_a_bug),
    icon = {
        Icon(
            painter = painterResource(id = R.drawable.bug_report),
            contentDescription = stringResource(R.string.
                ↪ report_a_bug),
            tint = it.suitableColor()
```

```
                    )
                }, onClick = {
                    context.sendEmail(email = Link.EMAIL)
                })
            TextForThisTheme(
                modifier = Modifier
                    .padding(10.dp)
                    .padding(bottom = 20.dp)
                    .align(Alignment.End),
                fontSize = FontSize.small17,
                text = "${stringResource(R.string.version)} ${LocalContext.
                    ↪ current.getVersionName()}"
            )
        }
    }
}
```