МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка Android-приложения для напоминаний о приеме медицинских препаратов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-341.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ П.Г. Верман

Автор работы,
студент группы КЭ-403
_____ Я.А. Землянкина

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студентке группы КЭ-403
Землянкиной Яне Александровне,
обучающейся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка Android-приложения для напоминаний о приеме медицинских препаратов.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. Android Studio Documentation. [Электронный ресурс] URL: https://developer.android.com/docs (дата обращения: 16.02.2024 г.).
3.2. Java Documentation. [Электронный ресурс] URL: https://docs.oracle.com/en/java/ (дата обращения: 16.02.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Произвести анализ предметной области.
4.2. Выполнить проектирование мобильного приложения.
4.3. Реализовать мобильное приложение.
4.4. Протестировать мобильное приложение.
5. **Дата выдачи задания:** 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП П.Г. Верман

Задание принял к исполнению Я.А. Землянкина

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном обществе смартфоны стали неотъемлемой частью нашей повседневной жизни. На сегодняшний день почти половина обладателей смартфонов использует их более 5 часов в сутки [1]. Столь высокая статистика использования смартфонов свидетельствует о том, что эти устройства за все время их существования превратились в универсальные инструменты, способные удовлетворить множество потребностей человека.

В свете постоянного совершенствования смартфонов наблюдается стремительный рост популярности мобильных приложений. Например, согласно статистике информационного портала «42 Matters», в 2024 году в «App Store» уже насчитывается более 1,8 миллиона приложений, а в «Google Play» почти в два раза больше – около 3,5 миллионов [2].

Мобильные приложения облегчают множество задач: от общения с близкими до управления финансами. Все больше людей обращаются к мобильным приложениям, чтобы решать разнообразные задачи, и это вполне логично. Ведь они предоставляют удобство, доступность и функциональность на расстоянии одного нажатия. Технологии мобильных устройств стали не просто средством связи, но и мощным инструментом для оптимизации и улучшения различных аспектов нашего быта. В этом контексте забота о здоровье через инновационные подходы становится более доступной и эффективной.

С развитием медицинской технологии возникает важная потребность в создании мобильных приложений, способных дополнительно облегчить процесс контроля своего здоровья. Одной из наиболее актуальных задач в этой области является соблюдение регулярного и своевременного приема медицинских препаратов, что, в свою очередь, может влиять на эффективность лечения и общее состояние здоровья пациента. Именно здесь вступает в игру приложение для напоминания о приеме медицинских препаратов, ко-

торое не только учитывает современные тенденции в использовании смартфонов, но и призвано улучшить практику приема лекарств.

Постановка задачи

Целью выпускной квалификационной работы является разработка Android-приложения для напоминаний о приеме медицинских препаратов. Для достижения поставленной цели необходимо решить следующие задачи:

1) произвести анализ предметной области;
2) выполнить проектирование приложения;
3) реализовать приложение;
4) протестировать приложение.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 47 страниц, объем списка литературы – 20 источников.

В первой главе «Анализ предметной области» рассматриваются существующие решения со схожей тематикой.

Во второй главе «Проектирование» приведены функциональные и нефункциональные требования к системе, построена диаграмма вариантов использования, описана архитектура системы и базы данных, а также спроектирован интерфейс приложения.

В третьей главе «Реализация» представлено описание реализации Android-приложения и особенности составляющих его компонентов.

В четвертой главе «Тестирование» содержатся результаты функционального тестирования, по которым сделаны вывод о достижении поставленных целей.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область

Примечательно, что в случае непоследовательного приема предписанных препаратов пациент может не только не избавиться от симптомов своего заболевания, но и не достичь полного выздоровления. Кроме того, такое несоблюдение режима лечения может повлечь за собой серьезные последствия и значительные расходы. Каждый год более 125 000 смертей от сердечно-сосудистых заболеваний, например, инфаркта миокарда и инсульта, приписываются неправильному приему лекарств. Также предполагается, что до 23% госпитализаций в лечебно-реабилитационные центры, 10% стационарных госпитализаций, а также множество посещений врачей, диагностических тестов и курсов лечения можно было бы избежать при более дисциплинированном приеме препаратов согласно предписаниям [3].

Неправильный прием лекарств не только увеличивает расходы на медицинское обслуживание, но также серьезно ухудшает качество жизни. Например, пропуск доз соответствующих препаратов может привести к повреждению зрительного нерва и слепоте у пациентов с глаукомой, нарушениям сердечного ритма и остановке сердца у больных сердечно-сосудистыми заболеваниями, а также инсультам у людей с повышенным артериальным давлением. Несоблюдение предписанных доз антибиотиков может вызвать вспышку инфекции и появление проблем, связанных с развитием устойчивых к антибиотикам бактерий.

Таким образом, своевременный прием препаратов способен значительно повысить эффективность лечения и содействовать полному выздоровлению. Приложение для напоминаний о принятии препаратов становится весьма полезным инструментом, помогающим пациентам следить за регулярностью приема лекарств. Это позволяет предотвращать пропуски доз, уменьшая риск возникновения серьезных осложнений и снижая необходимость в дорогостоящем медицинском лечении.

1.2. Обзор аналогов и существующих решений

В качестве аналогов и существующих решений были рассмотрены некоторые приложения из магазина «Google Play» со схожим функционалом, такие как MyTherapy, Medisafe и Pills Time.

MyTherapy

Приложение MyTherapy [4] может использоваться для комплексного напоминания о необходимой медицинской процедуре в определенное время, будь то прием лекарств, измерения показания здоровья или выполнение физических упражнений. На рисунке 1 представлен скриншот приложения.

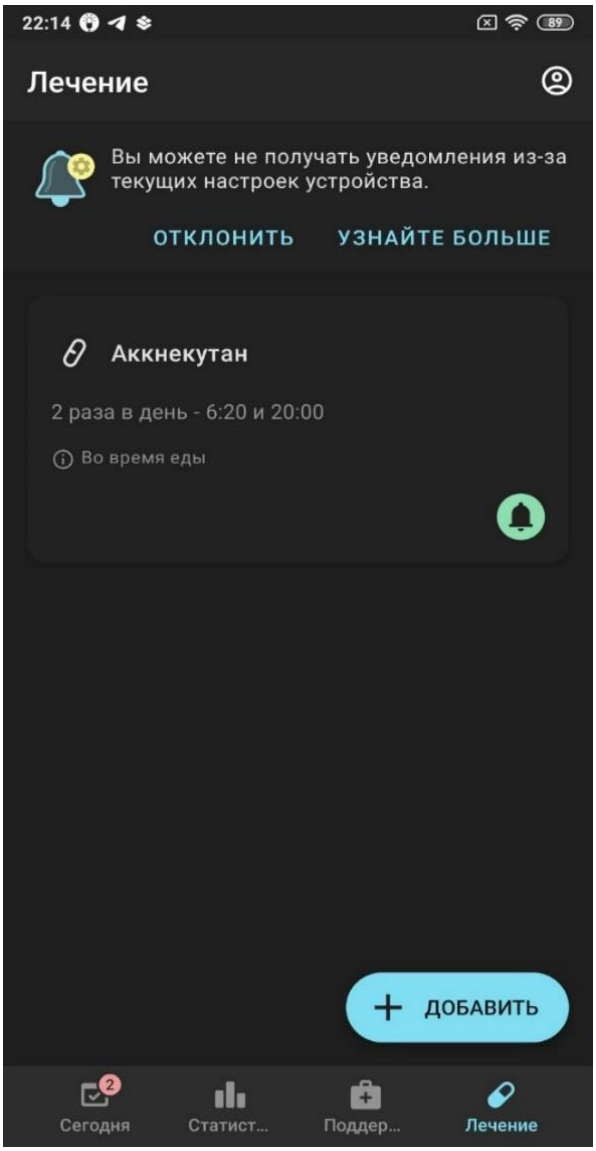

Рисунок 1 – Скриншот интерфейса «MyTherapy»

Пользователь может отслеживать прием лекарств по журналу, отмечать симптомы и делать заметки о состоянии здоровья, что может быть полезно при ведении журнала для медицинских консультаций или анализа динамики заболевания. В приложении возможно добавлять информацию о своих врачах и предстоящие приемы, также есть возможность создать отчет о состоянии лечения, которые можно отправить по почте и показать врачу во время приема. Одной из особенностей приложения является приглашение члена семьи или врача, которые могут видеть прогресс лечения.

Medisafe

Medisafe – еще одно функциональное приложение для напоминаний о приеме медицинских препаратов [5]. На рисунке 2 представлен скриншот приложения.

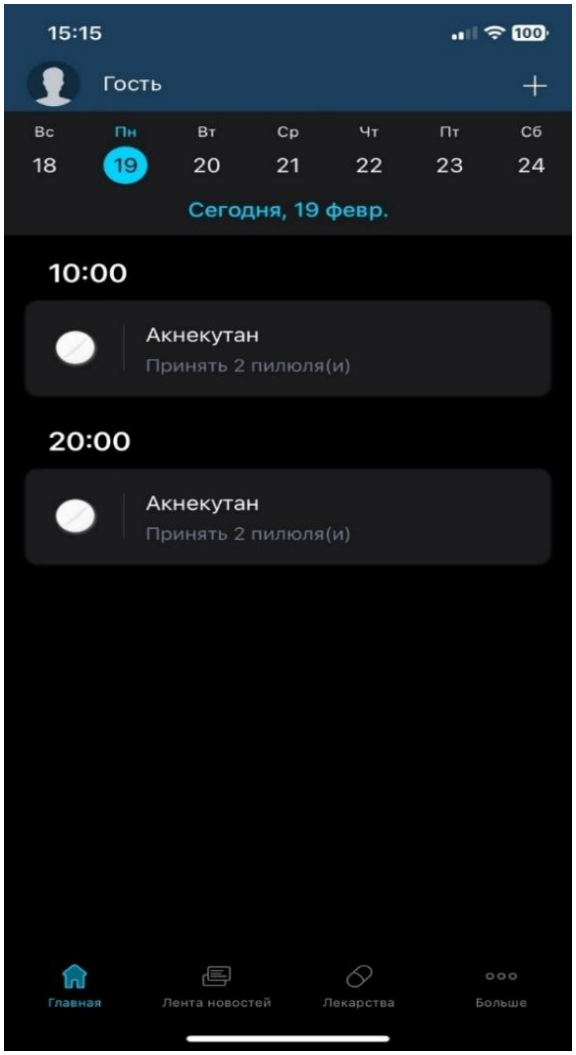

Рисунок 2 – Скриншот приложения «Medisafe»

Это приложение является довольно популярным и имеет более 5 млн. скачиваний. Medisafe подходит для людей, страдающих диабетом, сердечными заболеваниями, так как есть возможность отслеживать глюкозу в крови и артериальное давление, а также является удобным средством для напоминания о приеме противозачаточных средств. Преимуществом приложения можно выделить напоминание о пополнении лекарств, запись приема препарата по необходимости и напоминание о приеме к врачу.

Pills Time

Pills Time – это простое мобильное приложение для напоминания о приеме лекарств [6]. На рисунке 3 представлен скриншот приложения.

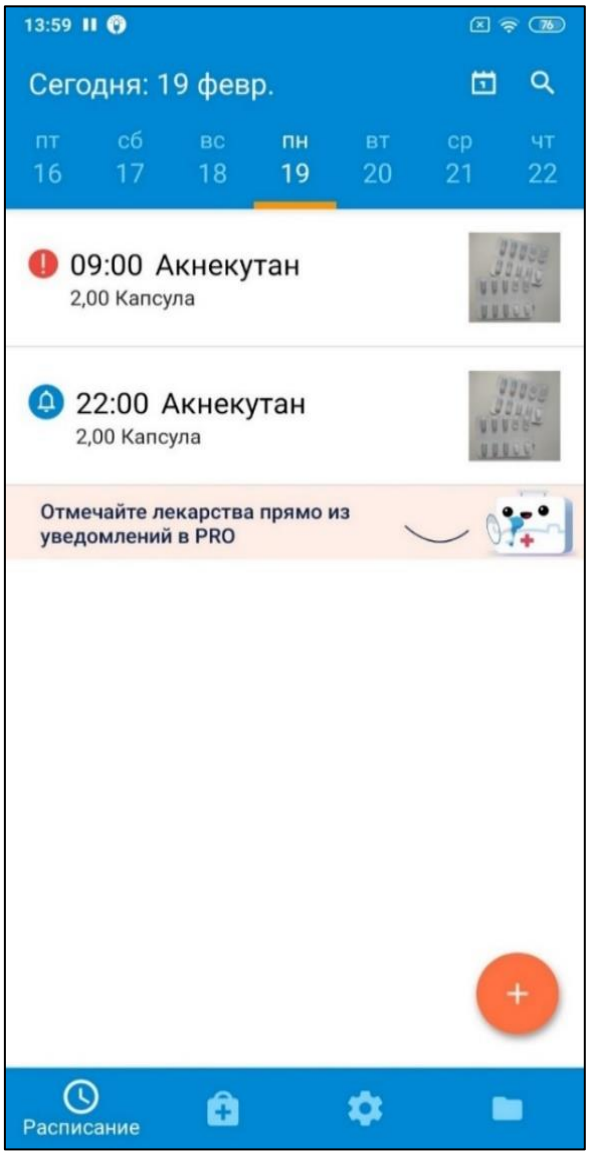

Рисунок 3 – Скриншот приложения «Pills Time»

С помощью данного приложения можно отслеживать только принятые лекарства и ставить напоминания о принятии пищи. Функция напоминаний о приеме врача является платной. Основным преимуществом приложения является возможность загрузить фотографию препаратов, а недостатком можно считать отсутствие составления отчета.

1.3. Сравнительный анализ

Для дальнейшей разработки собственного приложения необходимо сравнить существующие решения, чтобы выявить основные плюсы и минусы приложений. В таблице 1 представлена сравнительная характеристика возможностей рассмотренных приложений.

Таблица 1 – Сравнение приложений

Возможности приложений	**Приложения**		
	MyTherapy	**Medisafe**	**Pills Time**
Журнал жизненно важных показателей	Да	Да	Нет
Просмотр графиков	Да	Да	Нет
Загрузка фотографий	Нет	Нет	Да
Подтверждение приема из уведомлений	Да	Нет	Нет
Функция задержки уведомлений	Да	Да	Нет
Совместное использование	Да	Нет	Нет
Изменение темы	Да	Да	Нет
Платные функции	Нет	Да	Да

Из приведенной таблицы можно увидеть, что приложение MyTherapy позволяет вести журнал жизненно важных показателей, просматривать графики, подтверждать прием из уведомлений, использовать функцию задержки уведомлений и совместное использование. Кроме того, оно не имеет платных функций, что делает его доступным для широкого круга пользователей.

Medisafe также поддерживает журнал жизненно важных показателей и просмотр графиков, а также позволяет подтверждать прием из уведомлений. Однако оно не предлагает функции задержки уведомлений и совместного использования. В отличие от MyTherapy, Medisafe имеет платные

функции, что может ограничить его использование для некоторых пользователей.

Pills Time выделяется возможностью загрузки фотографий, но его функционал ограничен отсутствием журнала жизненно важных показателей, просмотра графиков, подтверждения приема из уведомлений, функции задержки уведомлений и совместного использования. Кроме того, оно включает платные функции.

Проанализировав структуру и функционал рассмотренных приложений, а также изучив предметную область, было принято решение сделать основным функционалом приложения напоминание о приеме препаратов в определенное время в нужных дозировках, предоставляя пользователям эффективный способ следить за своим лечением. Также следует добавить загрузку фотографий, просмотр графиков приема и функцию задержки уведомлений. Для удобства пользователей также стоит добавить возможность подтверждать прием прямо из уведомлений и возможность изменить тему приложения.

Вывод по первой главе

В данном разделе была рассмотрена предметная область приложения и были приведены существующие решения. А также на основе сравнительного анализа был выявлен функционал, который требуется разработать в своем приложении.

2. ПРОЕКТИРОВАНИЕ

Для качественного проектирования мобильного приложения требуется сформулировать функциональные и нефункциональные требования, описать архитектуру приложения, сформировать базу данных и разработать макеты страниц.

2.1. Требования

Функциональные требования

Функциональные требования определяют функциональность программного обеспечения, то есть описывают, какое поведение должна предоставлять разрабатываемая система. В результате анализа предметной области и обзора существующих решений были сформированы следующие требования. Были определены следующие функциональные требования:

1) приложение должно уведомлять пользователя о приеме лекарственного препарата;

2) приложение должно предоставлять пользователю возможность добавления лекарства;

3) приложение должно предоставлять пользователю возможность редактирования лекарства;

4) приложение должно предоставлять пользователю возможность просматривать и редактировать историю;

5) приложение должно предоставлять пользователю просматривать диаграммы приема лекарства;

6) приложение должно предоставлять пользователю изменять настройки;

7) приложение должно предоставлять пользователю возможность подтверждать или задерживать прием из уведомлений.

Нефункциональные требования

Нефункциональные требования представляют собой описание ограничений и свойств, применяемых к системе. Для реализации приложения

были сформулированы следующие нефункциональные требования:

1) приложение должно быть написано на языке программирования Kotlin;

2) приложение должно иметь интерфейс, разработанный при помощи языка разметки XML;

3) приложение должно работать на операционной системе Android, начиная с версии 6.

2.2. Варианты использования

Для проектирования описанных выше функциональных требований была создана диаграмма вариантов использования с помощью языка графического описания для объектного моделирования UML [7], показывающая отношения между актерами и прецедентами. На рисунке 4 приведена диаграмма вариантов использования мобильного приложения.

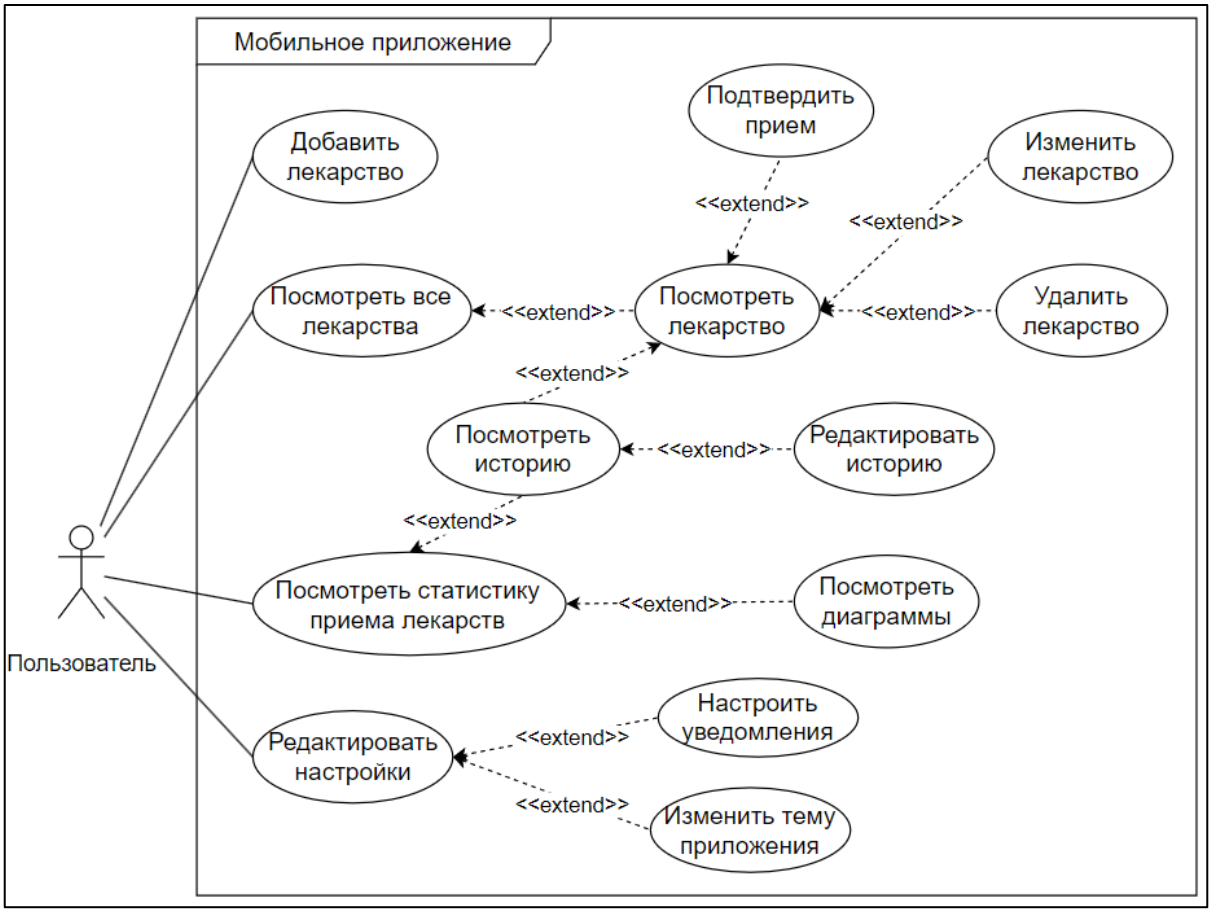

Рисунок 4 – Диаграмма вариантов использования

С приложением взаимодействует один актер – пользователь.

Пользователь может добавить лекарство в свой список напоминаний. Для добавления препарата необходимо указать название, выбрать цвет иконки для лекарства, настроить время напоминаний и указать количество таблеток. При желании можно загрузить фотографию и добавить описание.

На главном экране приложения пользователь может посмотреть все лекарства, ранее добавленные в приложение.

Пользователь может посмотреть конкретное лекарство из списка для получения более детальной информации. При просмотре пользователь может изменить лекарство, удалить только лекарство или же удалить лекарство и его историю. В окне просмотра определенного лекарства пользователь может посмотреть историю приема, а также подтвердить прием таблеток, если пришло время.

Пользователь может посмотреть историю приема, где указаны даты и время всех подтвержденных и пропущенных приемов. Пользователь может редактировать записи в истории для корректировки информации.

Пользователь может посмотреть статистику приема лекарств, что включает в себя просмотр истории приема всех лекарств, либо каждого по отдельности. Также пользователь может посмотреть диаграммы, на которых отражена информация о пропущенных и подтвержденных приемах.

Пользователь может редактировать различные настройки приложения. Пользователь может настроить уведомления, выбрав время для отложенных напоминаний, а также настраивать параметры уведомлений для конкретного препарата. В этом же окне пользователь может изменить тему приложения на светлую или темную.

2.3. Проектирование базы данных приложения

Для работы приложения была спроектирована локальная реляционная база данных, которая хранит всю пользовательскую информацию. Схема базы данных приведена на рисунке 5.

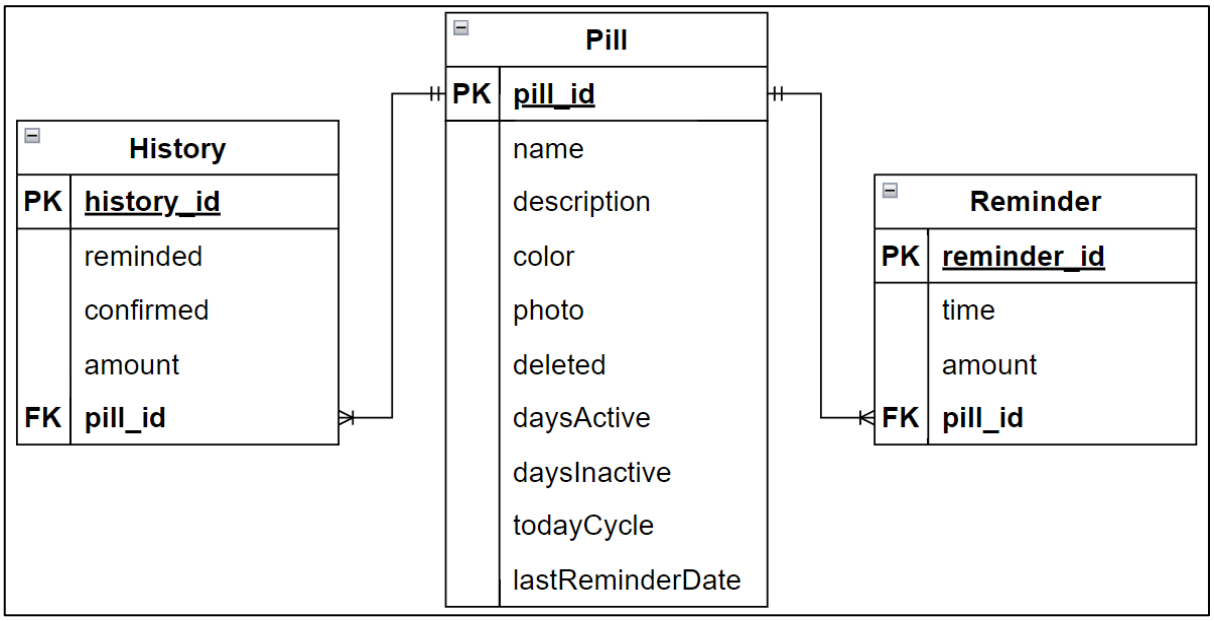

Рисунок 5 – Схема базы данных

База данных содержит 3 таблицы: Pill, History и Reminder.

Таблица Pill содержит информацию о препарате и имеет следующие поля:

1) pill_id – поле, в котором хранится уникальный идентификатор для каждой записи о препарате, является первичным ключом;

2) description – поле, в котором хранится описание лекарства;

3) color – поле, в котором хранится цвет иконки для препарата;

4) photo – поле, в котором хранится изображение препарата;

5) deleted – поле-флаг, указывающее, был ли удален препарат;

6) daysActive – поле, в котором хранится количество дней, когда напоминания активны;

7) daysInactive – поле, в котором хранится количество дней, когда напоминания неактивны;

8) todayCycle – поле, в котором хранится текущий день цикла напоминаний;

9) lastReminderDate – поле, в котором хранится дата последнего напоминания.

Таблица History содержит информацию об истории напоминаний и имеет следующие поля:

1) history_id – поле, в котором хранится уникальный идентификатор для каждой записи истории, является первичным ключом;

2) reminded – поле, которое хранит время, когда было получено напоминание;

3) confirmed – поле, которое хранит время, когда напоминание было подтверждено;

4) amount – поле, которое хранит количество таблеток, которое было принято;

5) pill_id – внешний ключ, который ссылается на таблицу Pill и связывает запись истории с конкретным лекарством.

Таблица Reminder содержит информацию о напоминаниях о приеме препаратов и содержит следующие поля:

1) Reminder_id – поле, в котором хранится уникальный идентификатор для каждого напоминания, является первичным ключом;

2) time – поле, которое содержит время напоминания;

3) amount – поле, которое содержит количество таблеток, которое необходимо принять;

4) pill_id – внешний ключ, который ссылается на таблицу Pill и связывает напоминание с конкретным препаратом.

2.4. Архитектура приложения

При разработке данного мобильного приложения будет применяться паттерн MVVM (Model-View-ViewModel) [8], схема которого приведена на рисунке 6.

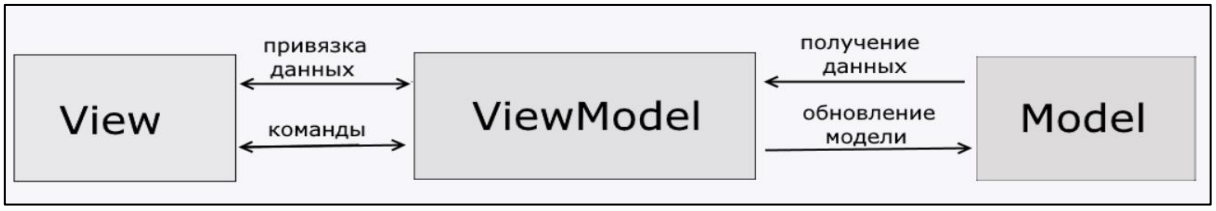

Рисунок 6 – Архитектура MVVM

MVVM, или Model-View-ViewModel, является архитектурным паттерном, широко используемым в разработке приложений для операционной системы Android. Паттерн был разработан для разделения разработки пользовательского интерфейса и бизнес-логики, что облегчает создание, тестирование и поддержку приложений.

В паттерне MVVM присутствуют три сущности.

1. Model представляет собой данные и бизнес-логику приложения. Это может быть база данных, веб-сервисы или любая другая логика, работающая с данными. Model не знает ничего о пользовательском интерфейсе и не зависит от него.

2. View отвечает за отображение данных и взаимодействие с пользователем. Важно, что View напрямую не взаимодействует с Model. View отвечает за визуальное представление информации и отправку пользовательских команд в ViewModel.

3. ViewModel служит посредником между Model и View. Он обрабатывает данные, полученные от Model, и подготавливает их для отображения в View. ViewModel также обрабатывает команды от View, управляя ими и вызывая соответствующие методы в Model. ViewModel обычно реализует механизм уведомлений, позволяя View автоматически обновляться при изменении данных.

Основные преимущества MVVM включают улучшение тестируемости кода. Поскольку ViewModel не зависит от конкретной реализации View, его можно легко тестировать с использованием модульных тестов. Это также способствует улучшению структурированности кода. Четкое разделение ответственности между компонентами Model, View и ViewModel помогает поддерживать код чистым и организованным. Повторное использование кода становится проще благодаря независимости компонентов. Логику, реализованную в ViewModel, можно использовать повторно в других View, что уменьшает дублирование кода.

Были выделены основные компоненты системы, которые представлены на рисунке 7.

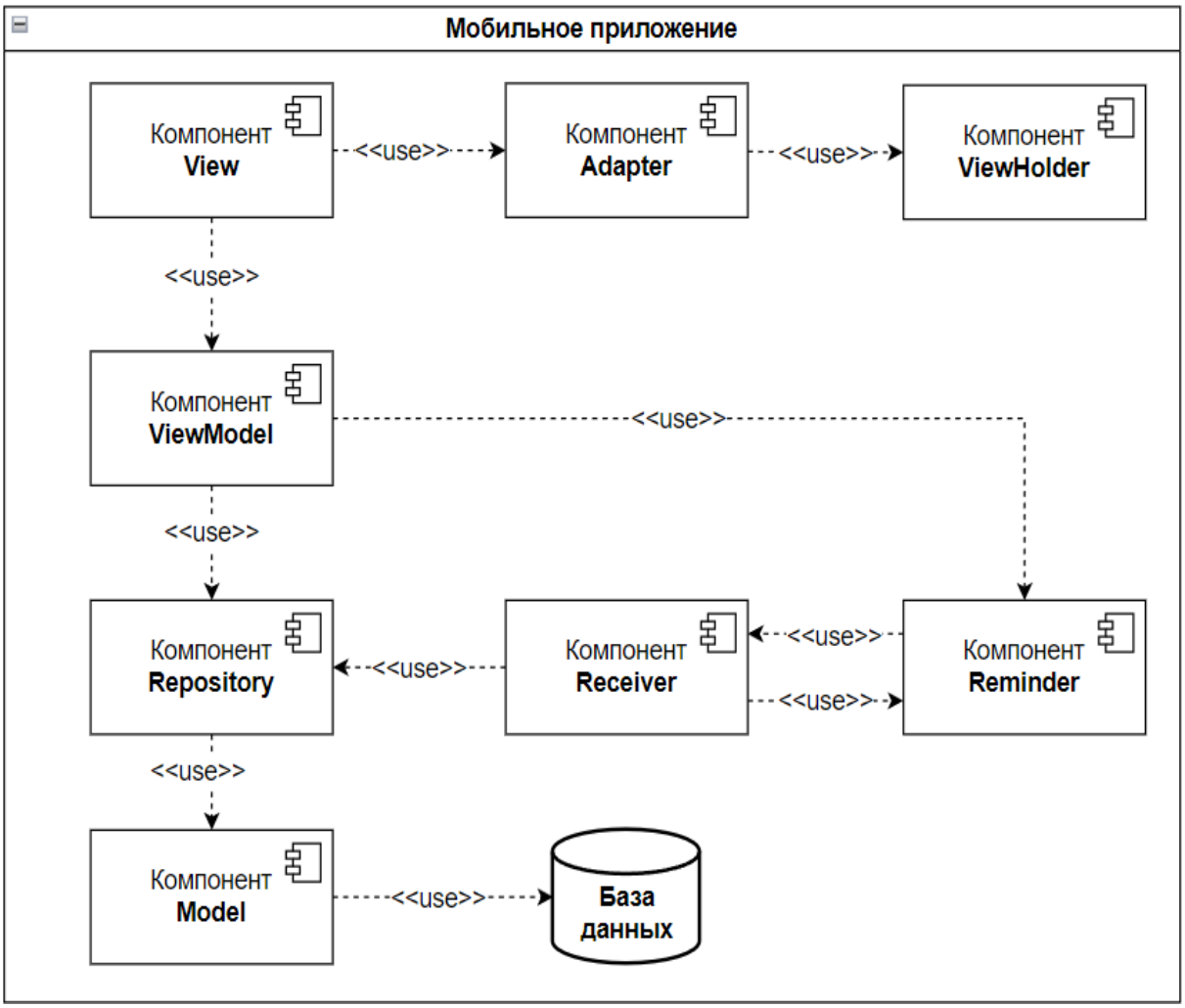

Рисунок 7 – Диаграмма компонентов мобильного приложения

Компонент Model включает классы, отвечающие за работу с базой данных. Каждой таблице соответствует класс, содержащий методы-запросы для получения, обновления и вставки данных, а также классы-структуры, описывающие объекты базы данных.

Компонент View объединяет классы, отвечающие за отрисовку элементов пользовательского интерфейса, отображающие данные, полученные от компонента ViewModel, и передающие действия пользователя обратно в ViewModel.

Компонент Adapter включает классы, которые служат промежуточным звеном между View и данными, предоставляемыми ViewModel. Adapter

преобразует данные в формат, подходящий для отображения в View, и управляет отображением этих данных в списках или других сложных структурах пользовательского интерфейса. Adapter также отвечает за оптимизацию работы с View, обеспечивая повторное использование элементов интерфейса через ViewHolder.

Компонент ViewHolder используется Adapter для управления представлением отдельных элементов в списке или другой структуре данных. Классы ViewHolder занимаются непосредственно отображением данных в отдельных элементах View, обеспечивая быстрый доступ к элементам интерфейса и минимизируя затраты на создание новых View.

Компонент ViewModel включает классы, отвечающие за подготовку данных для компонента View, обработку действий пользователя и взаимодействие с компонентом Repository для изменения данных. Он также использует компонент Reminder для управления напоминаниями.

Компонент Repository объединяет классы, которые обрабатывают доступ к данным и управляют данными из различных источников, таких как локальная база данных.

Компонент Receiver отвечает за получение и обработку данных из внешних источников, таких как системные уведомления или события. Классы Receiver реагируют на внешние события, такие как получение уведомлений от системы, и передают соответствующие данные в Repository для дальнейшей обработки.

Компонент Reminder включает классы, которые управляют напоминаниями, их созданием, планированием и отображением.

База данных представляет собой локальное место хранения информации об истории, таблетках и напоминаниях.

2.5. Пользовательский интерфейс

Были спроектированы макеты пользовательского интерфейса.

При запуске приложения пользователю открывается главный экран «Лекарства», из которого при просмотре лекарства можно перейти на экран «Просмотр». Макеты экранов представлены на рисунке 8.

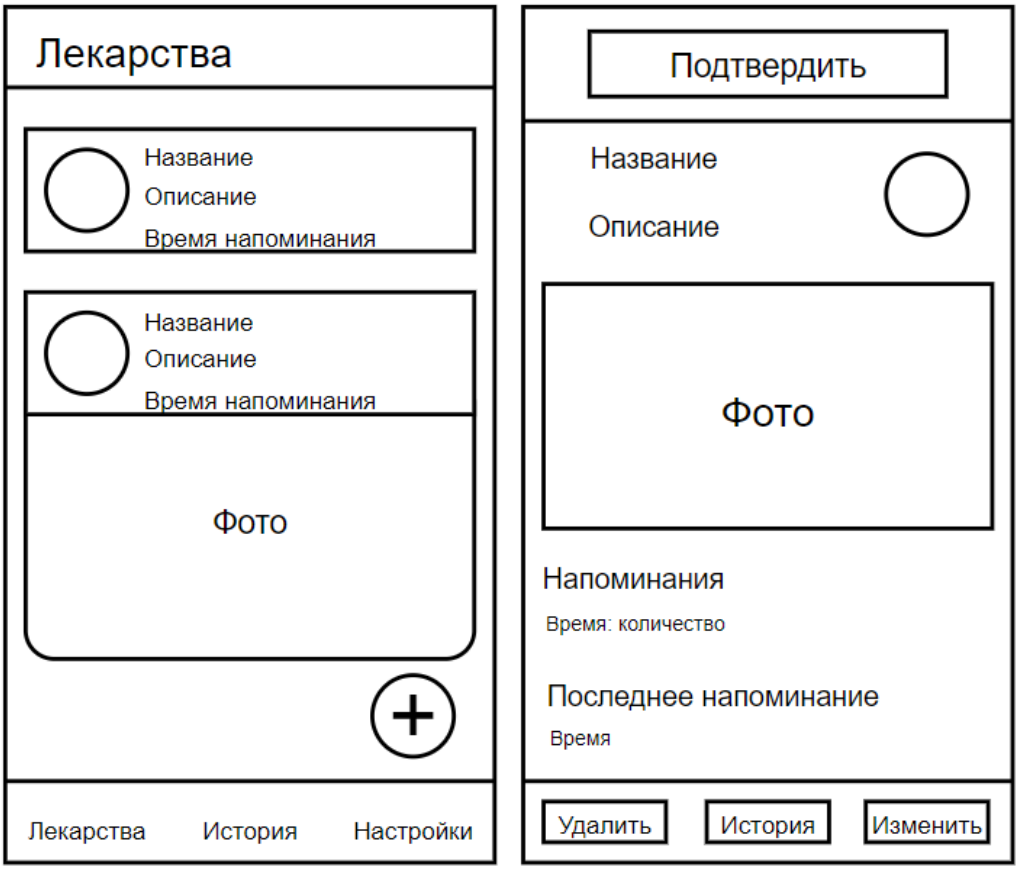

Рисунок 8 – Макеты экранов «Лекарства» и «Просмотр»

На главном экране «Лекарства» будет находиться список всех добавленных лекарств с краткой информацией, цветной иконкой и фотографией. Также внизу будет находиться кнопка добавления нового лекарства. При нажатии на лекарство будет открываться экран просмотра, где выведена полная информация о лекарстве и напоминаниях. На этом экране можно удалить лекарство из списка, изменить его или посмотреть его историю приема. Также здесь можно подтвердить прием лекарства, если пришло время.

Экран добавления лекарства будет содержать поля для ввода названия и описания лекарства, загрузку фотографии. Можно будет выбрать цвет для

иконки и специального оформления. Ниже будет располагаться кнопка «Добавить напоминание» для настройки времени и количества таблеток, которые надо принять. Также можно будет выбрать вариант приема лекарства: сделать его бессрочным, на определенное количество дней, либо же циклично. В правом нижнем углу будет размещена плавающая кнопка сохранения. Макет экрана добавления представлен на рисунке 9.

Рисунок 9 – Макет экрана добавления лекарства

Экран добавления лекарства будет содержать поля для ввода названия и описания лекарства, загрузку фотографии. Можно будет выбрать цвет для иконки и специального оформления. Ниже будет располагаться кнопка «Добавить напоминание» для настройки времени и количества таблеток, которые надо принять. Также можно будет выбрать вариант приема лекарства: сделать его бессрочным, на определенное количество дней, либо же циклично. В правом нижнем углу будет размещена кнопка сохранения.

Экран «Статистика» в приложении будет представлен двумя вкладками «История» и «Диаграммы». В разделе «История» будет доступна для просмотра история приема всех лекарств или одного конкретного. В разделе «Диаграммы» можно посмотреть статистику о приеме в виде трех диаграмм. Диаграммы будут отражать информацию в процентном виде о всех напоминаниях, о пропущенных напоминаниях каждого лекарства и о принятых и пропущенных всего напоминаниях. Макет экрана «Статистика» представлен на рисунке 10.

Рисунок 10 – Макет экрана «Статистика»

При просмотре истории будет выходить всплывающее окно, в котором отображены название лекарства, дата, время, когда пользователь подтвердил прием, время уведомления и количество таблеток. Кнопка «Удалить» будет удалять всю историю приема лекарства. Также каждую запись в истории можно будет отредактировать посредством меню действий. Макет всплывающего окна «История» представлен на рисунке 11.

Рисунок 11 – Макет всплывающего окна «История»

Вывод по второй главе

Во второй главе было выполнено проектирование мобильного приложение. Вначале были сформулированы функциональные и нефункциональные требования к приложению, что позволило определить ключевые возможности и ограничения, которые необходимо учитывать при его разработке. Затем была создана диаграмма вариантов использования, показывающая основные сценарии взаимодействия пользователей с приложением. Были разработаны структура и схема базы данных приложения. Также была спроектирована архитектура приложения, включающая выбор подходящей архитектурной модели и определение компонентов системы. Кроме того, были разработаны макеты экранов пользовательского интерфейса, которые визуализируют будущий внешний вид и функциональность приложения.

3. РЕАЛИЗАЦИЯ

3.1. Средства реализации

При разработке данного приложения использовался язык высокого уровня Kotlin. Kotlin – это современный статически типизированный язык программирования, разработанный компанией JetBrains и полностью совместимый с Java.

Для реализации приложения использовалась среда разработки Android Studio версии 2022.2.1. Android Studio – это интегрированная среда разработки (IDE) для создания приложений под операционную систему Android. Она была создана компанией Google и является наиболее распространенной IDE для разработки Android-приложений. Android Studio обеспечивает удобную работу с кодом, автоматически подсвечивая ошибки и предлагая подсказки для улучшения кода. Интегрированный отладчик позволяет быстро находить и исправлять ошибки в приложении.

Для хранения данных в приложении использовалась база данных SQLite [9]. SQLite – это встроенная реляционная база данных, которая является частью стандартной библиотеки Android. SQLite не требует настроек сервера и подходит для использования в мобильных приложениях.

3.2. Реализация базы данных

Для разработки реляционной локальной базы данных была использована библиотека Room [10], являющаяся частью Android Jetpack и обеспечивающая удобный способ работы с SQLite.

Была создана база данных, состоящая из трех классов хранения данных, таких как `Pill`, `Reminder` и `History`.

Класс `PillEnity` предназначен для хранения информации о лекарствах, добавленных пользователем. Структура таблицы представлена в листинге 1.

Листинг 1 – Структура класса `PillEnity`

```
@Entity(tableName = "pill")
data class PillEntity(
    var name: String,
    var description: String?,
    var photo: Bitmap?,
    var color: PillColor,
    var deleted: Boolean = false,
    var daysActive: Int = NO_DAY_LIMIT,
    var daysInactive: Int = NO_BREAK,
    var todayCycle: Int = 1,
    var lastReminderDate: Calendar? = null,
    @PrimaryKey(autoGenerate = true) @ColumnInfo(name = "pillId") val id:
Long = 0
)
```

Класс `Reminder` содержит информацию о напоминаниях. Связь с таблицей `PillEnity` реализована через внешний ключ `pillId`. Структура таблицы представлена в листинге 2.

Листинг 2 – Структура класса `Reminder`

```
@Entity(
    tableName = "reminder",
    foreignKeys = [ForeignKey(
        entity = PillEntity::class,
        parentColumns = arrayOf("pillId"),
        childColumns = arrayOf("pillId"),
        onDelete = ForeignKey.CASCADE
    )])
data class Reminder(
    var time: Calendar,
    var amount: String,
    @ColumnInfo(index = true)
    var pillId: Long,
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "reminderId")
    val id: Long = 0)
```

Таблица `History` хранит данные о событиях напоминаний и их подтверждениях. Связь с таблицей `PillEnity` также реализована через внешний ключ `pillId`. Структура таблицы представлена в листинге 3.

Листинг 3 – Структура класса `History`

```
@Entity(
    tableName = "history",
    foreignKeys = [ForeignKey(
        entity = PillEntity::class,
        parentColumns = arrayOf("pillId"),
        childColumns = arrayOf("pillId"),
        onDelete = ForeignKey.CASCADE)])
data class History(
        @PrimaryKey(autoGenerate = true)
        @ColumnInfo(name = "historyId")
```

```
    val id: Long = 0,

    var reminded: Calendar,
    var confirmed: Calendar? = null,
    val amount: String = "1",
    @ColumnInfo(index = true)
    var pillId: Long)
```

Также для работы с таблицами базы данных были созданы интерфейсы Data Access Object (DAO), такие как `PillDao`, `ReminderDao`, и `HistoryDao` соответственно для каждой таблицы. Интерфейс DAO предоставляет абстракцию доступа к данным, что позволяет приложению взаимодействовать с базой данных через методы, определённые в DAO, вместо прямого использования SQL-запросов. Интерфейс `PillDao` представлен в листинге 4.

Листинг 4 – Интерфейс `PillDao`

```
@Dao
interface PillDao {
    @Transaction
    @Query("SELECT * FROM pill WHERE deleted = 0 ORDER BY pillId ASC")
    fun getEverythingFlow(): Flow<List<Pill>>

    @Transaction
    @Query("SELECT * FROM pill WHERE deleted = 0 ORDER BY pillId ASC")
    suspend fun getEverything(): List<Pill>

    @Transaction
    @Query("SELECT * FROM pill ORDER BY pillId ASC")
    fun getEverythingIncludingDeletedFlow(): Flow<List<Pill>>

    @Transaction
    @Query("SELECT * FROM pill ORDER BY pillId ASC")
    suspend fun getEverythingIncludingDeleted(): List<Pill>

    @Transaction
    @Query("SELECT * FROM pill WHERE pillId = (:pillId)")
    fun getWithIdFlow(pillId: Long): Flow<Pill>

    @Transaction
    @Query("SELECT * FROM pill WHERE pillId = (:pillId)")
    suspend fun getWitId(pillId: Long): Pill

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertPillEntity(pillEntity: PillEntity): Long

    @Update
    suspend fun updatePillEntity(pillEntity: PillEntity)

    @Delete
    suspend fun deletePillEntity(pillEntity: PillEntity)}
```

3.3. Реализация основного функционала

Главный экран

Когда пользователь открывает главное окно «Лекарства», управляемое классом `HomeFragment`, инициализируется `HomeViewModel`, который отвечает за получение данных о лекарствах. `HomeViewModel` содержит метод `getAllPillsWithHistoryFlow`, который обращается к `PillRepository` для получения списка лекарств вместе с их историей. Эти данные преобразуются в `LiveData` и наблюдаются в `HomeFragment`. `LiveData` – это компонент архитектуры Android, который обеспечивает автоматическое обновлении пользовательского интерфейса в ответ на изменения данных [11].

В `HomeFragment` создается и настраивается адаптер `AppRecyclerAdapter`, который управляет отображением элементов списка в `RecyclerView`. `RecyclerView` – это компонент пользовательского интерфейса Android, предназначенный для отображения наборов данных в виде списков [12].

При создании представления, `HomeFragment` настраивает `RecyclerView` для использования адаптера `AppRecyclerAdapter` и наблюдает за изменениями данных в `HomeViewModel`. Когда данные о лекарствах изменяются, адаптер обновляется методом `submitList`, который передает новый список лекарств для отображения.

В адаптере `AppRecyclerAdapter` реализован метод `onCreateViewHolder`, который создает экземпляры `PillViewHolder` для отображения каждого элемента списка. Метод `onBindViewHolder` связывает данные с элементами пользовательского интерфейса, вызывая метод `bind` в `PillViewHolder`. Фрагменты методов `onCreateViewHolder` и `onBindViewHolder` представлены в листинге 5.

Листинг 5 – Фрагменты методов класса `AppRecyclerAdapter`

```
override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position:
Int) {
    when (holder.itemViewType) {
        ItemType.PILL.ordinal -> if (holder is PillViewHolder) holder.bind(
            getItem(position) as Pill)}
```

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Recy-
clerView.ViewHolder {
    val layoutInflater = LayoutInflater.from(parent.context)
    return when (viewType) {
        ItemType.PILL.ordinal -> PillViewHolder(
            ItemPillBinding.inflate(layoutInflater, parent, false),
            onItemClickListener,
            onPillConfirmClickListener
        )}
```

Класс `PillViewHolder` отвечает за непосредственное отображение данных о лекарстве. Он связывает информацию о лекарстве с соответствующими элементами пользовательского интерфейса, такими как имя лекарства, изображение, описание и напоминания. Также он настраивает слушатели кликов для обработки подтверждений приема лекарства.

Главный экран «Лекарства» представлен на рисунке 12.

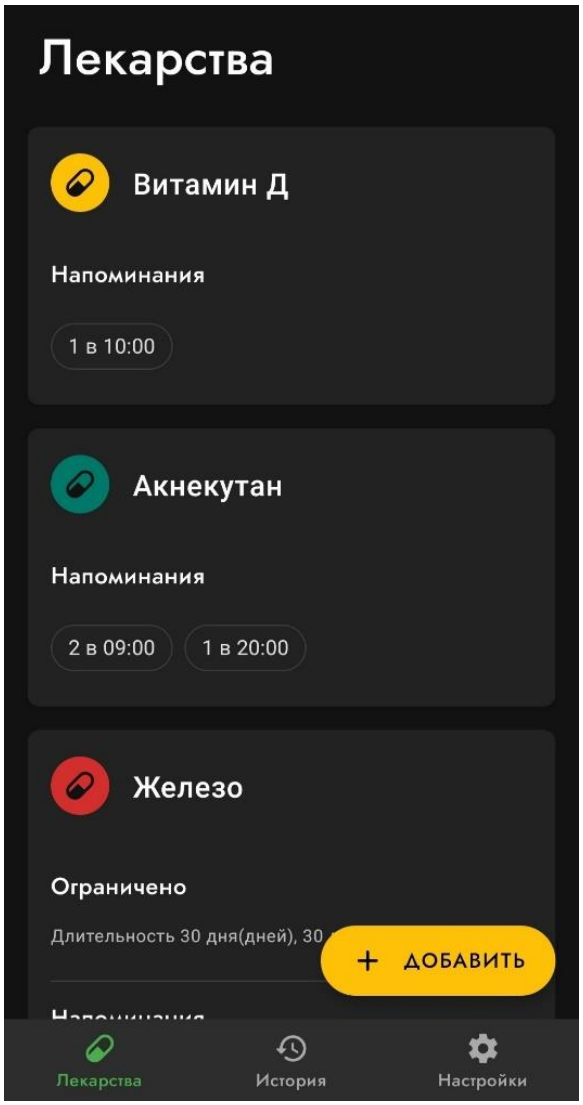

Рисунок 12 – Главный экран «Лекарства»

При нажатии на главном экране на кнопку «Добавить» открывается экран для добавления лекарства. Этот же экран отвечает за изменение лекарства. Экран представлен на рисунке 13.

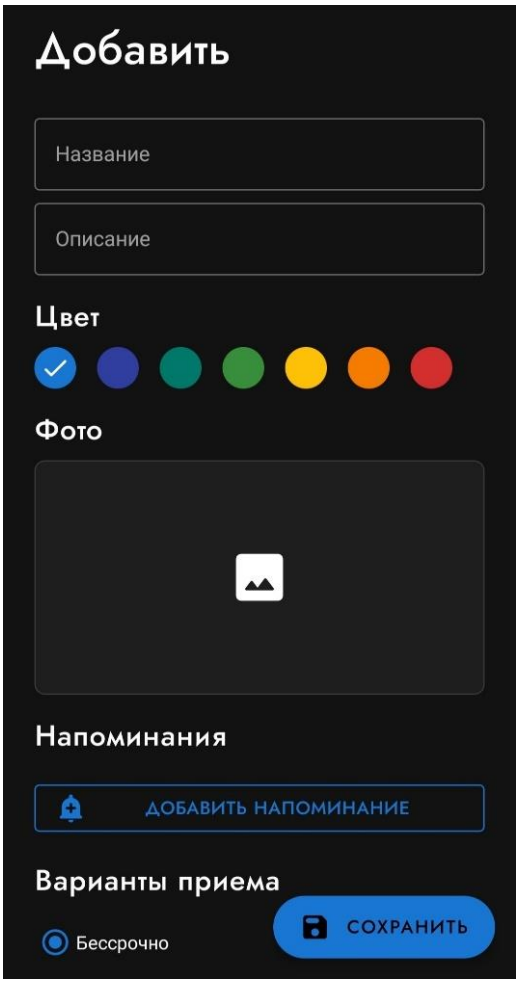

Рисунок 13 – Экран «Добавить»

Пользователь должен ввести название, описание, выбрать цвет и фотографию, также назначить уведомления и режим приема. При нажатии кнопки «Сохранить» в фрагменте `EditFragment` вызывается метод `onPillSave`, в котором происходит проверка на введение имени и хотя бы одного напоминания. Метод `onPillSave` представлен в листинге 6.

Листинг 6 – Метод `onPillSave`

```
private fun onPillSave() = binding.run {
    if (model.pill.name.isBlank()) {
        inputNameLayout.error = getString(R.string.enter_field_name)
        scrollEdit.smoothScrollTo(0, 0)
        return@run}
    if (model.pill.reminders.isEmpty()) {
        showSnackbar(getString(R.string.no_reminders_set))
```

```
        return@run}
    model.pill.options = pillOptionsView.getOptions()
    if (isCreatingNewPill) {
        model.addPill(model.pill, requireActivity().applicationContext)
        exitTransition = Slide().addTarget(R.id.layoutEdit)
    } else {
        model.updatePill(model.pill, requireActivity().applicationContext)
        returnTransition = MaterialSharedAxis(MaterialSharedAxis.Y, false)}
    findNavController().popBackStack()}
```

Если создается новое лекарство, которое определяется условием `isCreatingNewPill`, вызывается метод `addPill` из `EditViewModel`, который добавляет новое лекарство в базу данных. Если лекарство редактируется, вызывается метод `updatePill`, который обновляет информацию о существующем лекарстве в базе данных.

История

На экране истории лекарств отображается информация о приеме лекарств пользователями. Экран «История» представлен на рисунке 14.

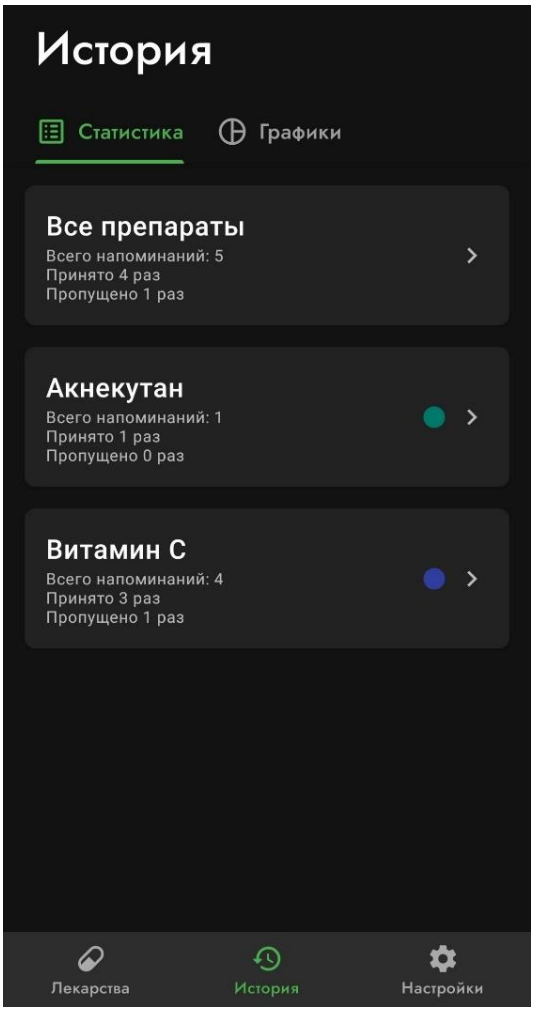

Рисунок 14 – Экран «История»

На экране «История» в приложении реализованы вкладки, позволяющие пользователям переключаться между разными представлениями данных, такими как статистика и диаграммы. Это достигается с помощью фрагмента `HistoryFragment`, который управляет отображением вкладок и их содержимым. В `HistoryFragment` используется макет `fragment_history`, в котором расположены `ViewPager` и `TabLayout`. `ViewPager` отвечает за перелистывание между различными фрагментами, а `TabLayout` отображает сами вкладки. Для связывания `TabLayout` с `ViewPager` используется `TabLayoutMediator` [13]. Он синхронизирует вкладки с содержимым `ViewPager`, так что при переключении между вкладками изменяется соответствующий фрагмент. При создании фрагмента `HistoryFragment` в методе `onCreate` устанавливаются анимации плавных переходов с помощью `MaterialFadeThroug` [14]. В методе `onViewCreated` устанавливается адаптер `HistoryViewPagerAdapter` для `ViewPager`, который управляет фрагментами вкладок. Класс `HistoryFragment` представлен на листинге 7.

Листинг 7 – Класс `HistoryFragment`

```
class HistoryFragment : Fragment(R.layout.fragment_history) {

    private val binding by viewBinding(FragmentHistoryBinding::bind)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enterTransition = MaterialFadeThrough()
        exitTransition = MaterialFadeThrough()}
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding.pager.adapter = HistoryViewPagerAdapter(this)
        TabLayoutMediator(binding.tabLayout, binding.pager){tab,position ->
            tab.text = when (position) {
                0 -> getString(R.string.overview)
                1 -> getString(R.string.charts)
                else -> getString(R.string.history)
            }
            tab.icon = ResourcesCompat.getDrawable(
                resources, when (position) {
                    0 -> R.drawable.ic_list_alt
                    1 -> R.drawable.ic_pie_chart
                    else -> R.drawable.ic_history
                }, context?.theme)
        }.attach()}
```

Класс `HistoryOverviewViewModel` отвечает за подготовку данных для отображения на вкладке «Статистика». Он взаимодействует с репозиториями `HistoryRepository` и `PillRepository` для получения и обновления информации об истории приема лекарств.

Для отображения списка элементов истории используется RecyclerView и адаптер `HistoryViewAdapter`. Адаптер связывает данные с представлениями в RecyclerView, обеспечивая корректное отображение карточек историй. Внутри адаптера используется класс `HistoryViewHolder` для отображения списка истории. Он отображает общую историю всех лекарств, а также каждую по отдельности. Класс представлен в листинге 8.

Листинг 8 – Класс `HistoryViewHolder`

```
class HistoryViewHolder(
    private val binding: ItemHistoryPillBinding,
    private val clickListener: (View, BaseModel) -> Unit
) : RecyclerView.ViewHolder(binding.root) {

    fun bind(historyPill: HistoryPillItem) = binding.run {
        when (historyPill.historyType) {

            is HistoryOverallItem -> {
                viewPillColor.isVisible = false
                textHistoryName.text = binding.root.con-
text.getString(R.string.stat_overall)}
            is Pill -> {

                viewPillColor.isVisible = true
                viewPillColor.setBackgroundColorShaped(historyPill.histo-
ryType.colorResource(context))

                textHistoryName.text = historyPill.historyType.name}}

        cardHistoryPill.onClick { view ->
            clickListener(view, historyPill.historyType)}

        textHistoryDescription.text = historyPill.stat.getSummaryText(con-
text)
        textHistoryDescription.isVisible = historyPill.stat.hasStats}}
```

При нажатии на конкретную историю лекарства открывается всплывающее окно, за которое отвечает класс `HistoryViewDialog`. Окно с историей лекарств представлено на рисунке 15.

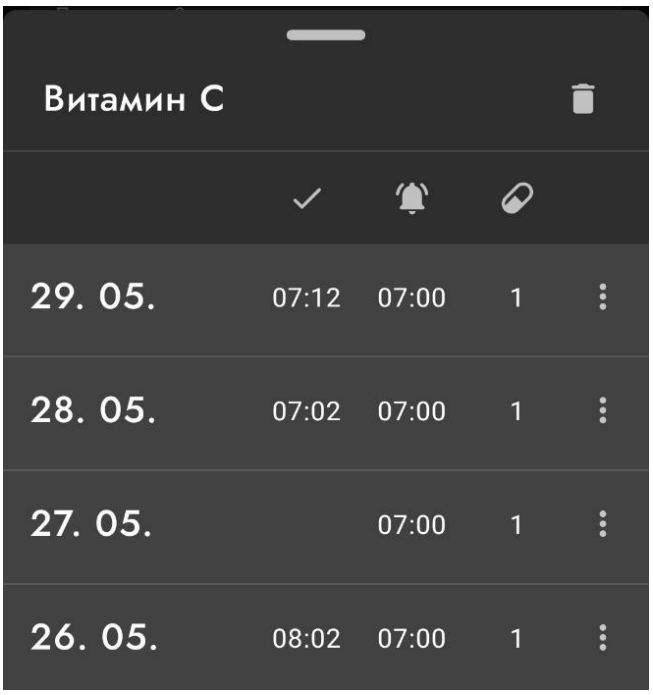

Рисунок 15 – Окно просмотра истории лекарств

В нем содержатся элементы истории приема вместе с датой, временем подтверждения приема и напоминания, количеством таблеток и кнопкой для редактирования истории. Все эти элементы отображает класс `HistoryItemViewHolder`, который представлен на листинге 9.

Листинг 9 – Класс `HistoryItemViewHolder`

```
class HistoryItemViewHolder(
    private val binding: ItemHistoryBinding,
    private val optionsClickListener: (View, History, Int) -> Unit
) : RecyclerView.ViewHolder(binding.root) {
    fun bind(
        history: History, isFirstInList: Boolean, isFirstOfDate: Boolean,
position: Int, showNames: Boolean) = binding.run {
        textHistoryAmount.text = history.amount
        buttonShowMore.onClick {
            optionsClickListener(it, history, position)}
        textHistoryReminded.setTimeText(history.reminded.time)
        textHistoryConfirmed.isInvisible = !history.hasBeenConfirmed
        textHistoryConfirmed.setTimeText(history.confirmed?.time ?: his-
tory.reminded.time)
        textDate.setDateText(history.reminded.time)
        textPillName.text = history.pillName
        textPillName.isVisible = showNames
        divider.isVisible = isFirstOfDate && !isFirstInList
        dividerPill.isVisible = !isFirstOfDate
        if (showNames) {
            textHistoryConfirmed.setVerticalBias(1.0f)
            textDate.isVisible = isFirstOfDate} else {
            textHistoryConfirmed.setVerticalBias(0.5f)
            textDate.isInvisible = !isFirstOfDate}}}
```

При нажатии кнопки buttonShowMore всплывает контекстное меню (PopupMenu) [15], которое позволяет изменять параметры элемента истории или удалить этот элемент. Методы для изменений содержатся в классе HistoryItemViewModel, который в свою очередь обращается к классам HistoryRepository и PillRepository для соответствующих изменений в базе данных. Фрагмент класса HistoryItemViewModel с методами для изменения истории представлен в листинге 10.

Листинг 10 – Фрагмент класса HistoryItemViewModel

```
fun confirmHistory(item: History) = viewModelScope.launch {
    val historyEntity =
        History(item.id, item.reminded, Calendar.getInstance(),
item.amount, item.pillId)
    historyRepository.updateHistoryItem(historyEntity)}
fun markHistoryNotConfirmed(item: History) = viewModelScope.launch {
    val historyEntity = History(item.id, item.reminded, null, item.amount,
item.pillId)
    historyRepository.updateHistoryItem(historyEntity)}
fun setHistoryConfirmTime(item: History, newConfirmTime: Calendar) = view-
ModelScope.launch {
    val historyEntity =
        History(item.id, item.reminded, newConfirmTime, item.amount,
item.pillId)
    historyRepository.updateHistoryItem(historyEntity)}
fun setHistoryAmount(item: History, newAmount: String) = viewMod-
elScope.launch {
    val historyEntity = History(item.id, item.reminded, item.confirmed,
newAmount, item.pillId)
    historyRepository.updateHistoryItem(historyEntity)}
fun deleteHistory(item: History) = viewModelScope.launch {
    historyRepository.deleteHistoryItem(item)}
```

В окне «История» пользователь также может просматривать диаграммы по категориям: «Все лекарства», «Пропущенные лекарства» и «Подтверждённые / Пропущенные лекарства».

Когда пользователь открывает вкладку с диаграммами, отображается фрагмент HistoryChartFragment. Первым делом, в методе onViewCreated этого фрагмента создаётся адаптер AppRecyclerAdapter, который будет использоваться для отображения элементов в RecyclerView. Этот адаптер используется для показа диаграмм и сообщений о пустой истории.

Затем, фрагмент начинает наблюдать за данными, которые предоставляет HistoryChartViewModel, который отвечает за подготовку данных для

диаграмм. Когда данные истории обновляются, модель данных собирает и обрабатывает информацию о подтверждённых и пропущенных дозировках, а также группирует историю по каждому лекарству. Метод `getStatsData` внутри `HistoryChartViewModel` возвращает данные в формате `LiveData`, которые наблюдаются фрагментом. Эти данные включают три набора данных для диаграмм: данные обо всех лекарствах, данные о пропущенных дозировках и данные о подтверждённых и пропущенных дозировках. Метод `getStatsData` представлен на листинге 11.

Листинг 11 – Метод `getStatsData`

```
fun getStatsData(applicationContext: Context) =
    historyRepository.getHistoryFlow().map { history ->
        if (history.isEmpty()) {
            return@map null}
        val totalConfirmed = history.count { it.hasBeenConfirmed }
        val totalMissed = history.size - totalConfirmed
        val colorsAll = mutableListOf<Int>()
        val colorsMissed = mutableListOf<Int>()
        val colorsConfirmed = mutableListOf(
            applicationContext.getColor(R.color.colorGreen),
            applicationContext.getColor(R.color.colorRed))
        val allEntries = ArrayList<PieEntry>()
        val missedEntries = ArrayList<PieEntry>()
        val confirmedEntries: ArrayList<PieEntry> = arrayListOf(
            PieEntry(
                totalConfirmed.toFloat(),
                applicationContext.getString(R.string.confirmed)),
            PieEntry(totalMissed.toFloat(), applicationCon-
text.getString(R.string.missed)))
        val pillsHistory = history.groupBy { it.pillId }.values
        pillsHistory.forEach { pillHistory ->
            val pill = pillRepository.getPill(pillHistory.first().pillId)
            colorsAll.add(pill.colorResource(applicationContext))
            allEntries.add(PieEntry(pillHistory.size.toFloat(), pill.name))
            val pillMissed = pillHistory.count { !it.hasBeenConfirmed }
            if (pillMissed > 0) {
                colorsMissed.add(pill.color.getColor(applicationContext))
                missedEntries.add(PieEntry(pillMissed.toFloat(),
pill.name))}}
        val dataSetAll = getDataSet(allEntries, colorsAll, applicationCon-
text)
        val dataSetMissed = getDataSet(missedEntries, colorsMissed, appli-
cationContext)
        val dataSetConfirmed = getDataSet(confirmedEntries, colorsCon-
firmed, applicationContext)
        val dataAll = getData(dataSetAll)
        val dataMissed = getData(dataSetMissed)
        val dataConfirmed = getData(dataSetConfirmed)
        return@map (listOf(dataAll, dataMissed, dataConfirmed))
    }.asLiveData()
```

Когда данные готовы, они преобразуются в объекты `ChartItem`, каждый из которых содержит заголовок и данные для диаграммы. Эти объекты затем передаются в адаптер `AppRecyclerAdapter`, который отображает диаграммы в `RecyclerView`. Вкладка «Диаграммы» представлена на рисунке 16.

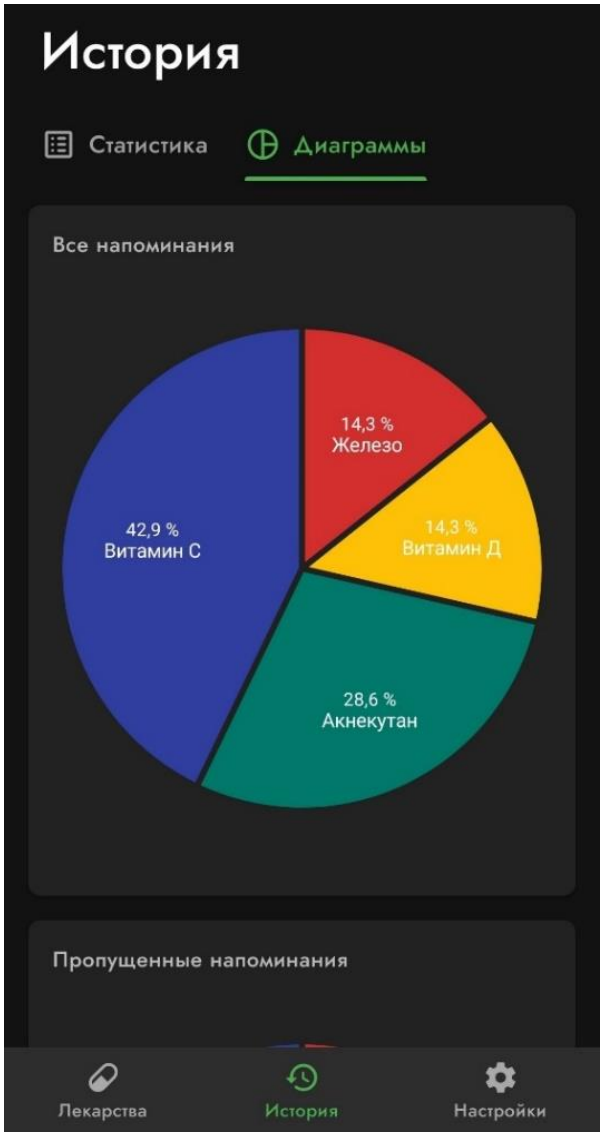

Рисунок 16 – Вкладка «Диаграммы»

Каждый элемент диаграммы отображается с помощью `ChartViewHolder`. Метод `bind` принимает объект `ChartItem` и настраивает `PieChart` из библиотеки MPAndroidChart [16]. Настройки включают отключение описания диаграммы, использование процентных значений, анимацию диаграммы, радиус центрального отверстия и стилизацию элементов

диаграммы. Затем данные PieData назначаются диаграмме, и она обновляется методом invalidate, чтобы отобразить данные на экране. Класс ChartViewHolder представлен на листинге 12.

Листинг 12 – Класс ChartViewHolder

```
class ChartViewHolder(
    private val binding: ItemChartBinding
) : RecyclerView.ViewHolder(binding.root) {
    fun bind(chartItem: ChartItem) = binding.run {
        chartTitle.text = chartItem.title
        pieChart.apply {
            description.isEnabled = false
            setUsePercentValues(true)
            dragDecelerationFrictionCoef = 0.8f
            animateY(1400, Easing.EaseInOutQuad)
            holeRadius = 25f
            isDrawHoleEnabled = false
            transparentCircleRadius = 30f
            setDrawEntryLabels(true)}
        pieChart.legend.apply {
            textColor = context.getAttrColor(R.attr.colorOnSurface)
            isWordWrapEnabled = true
            textSize = 12f
            isEnabled = false}
        chartItem.pieData.setValueFormatter(PercentFormatter(pieChart))
        pieChart.data = chartItem.pieData
        pieChart.invalidate()}}
```

Настройки

Когда пользователь открывает настройки, загружается PreferencesFragment, который наследует от PreferenceFragmentCompat. Компонент Preference от Android нужен для создания экрана настроек, который представлен в виде списка предпочтений [17]. Внутри метода onCreatePreferences вызывается метод setPreferencesFromResource, который загружает предпочтения из XML-файла.

XML-файл включает в себя следующие элементы.

1. SeekBarPreference служит для настройки задержки перед напоминанием в минутах.

2. SwitchPreferenceCompat служит для включения или выключения повторных напоминаний.

3. SeekBarPreference служит для настройки интервала повторных напоминаний в минутах.

4. `Preference` открывает настройки уведомлений приложения в системных настройках.

5. `ListPreference` служит для выбора темы приложения. Пользователь может выбрать одну из тем: «Системная», «Светлая» и «Темная».

Экран «Настройки» представлен на рисунке 17.

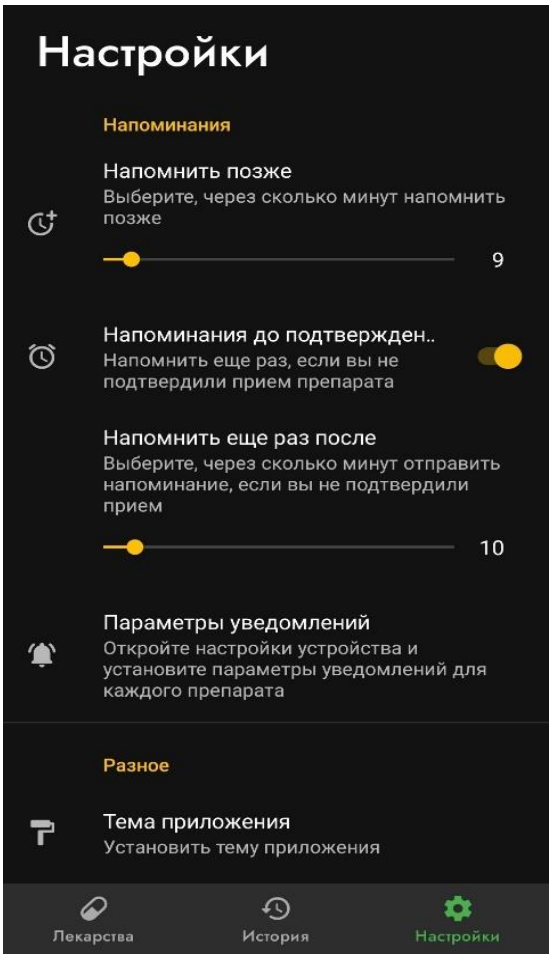

Рисунок 17 – Экран «Настройки»

3.4. Реализация уведомлений

В данном приложении система уведомлений играет ключевую роль, обеспечивая своевременные напоминания пользователям о необходимости приема лекарств. Для создания, управления и отображения уведомлений используются классы, такие как `NotificationManager`, `ReminderManager`, `ReminderUtil` и `ReminderReceiver`.

1. `NotificationManager` содержит методы, которые отвечают за создание и отображение уведомлений, а также за управление каналами уведомлений.

2. `ReminderManager` содержит методы, которые отвечают за планирование напоминаний на основе времени, установленного пользователем для каждой таблетки.

3. `ReminderUtil` предоставляет методы для создания интентов, которые используются в системе уведомлений.

4. `ReminderReceiver` является `BroadcastReceiver` [18], который отвечает за получение и обработку системных широковещательных сообщений, а также за отправление уведомлений.

Когда пользователь устанавливает время для напоминания, приложение вызывает метод `planNextPillReminder` из `ReminderManager` для планирования напоминания для этой таблетки. Метод представлен в листинге 13.

Листинг 13 – Метод `planNextPillReminder`

```
fun planNextPillReminder(context: Context, pill: Pill) {
    planNextReminder(context, pill.reminders)}

private fun planNextReminder(context: Context, reminders: List<Reminder>) {
    val sortedByTime = reminders.sortedBy { it.time.timeInMillis }
    val today = Calendar.getInstance()
    sortedByTime.forEach { reminder ->
        val remindTime = reminder.getTodayMillis()
        if (remindTime > today.timeInMillis) {
            createAlarm(context, reminder)
            return@planNextReminder}}
    val firstTomorrow = sortedByTime.firstOrNull()
    firstTomorrow?.let { reminder ->
        val triggerAtCalendar = Calendar.getInstance()
        triggerAtCalendar.timeInMillis = reminder.getTodayMillis()
        triggerAtCalendar.add(Calendar.DAY_OF_YEAR, 1)
        createAlarm(context, reminder.id, triggerAtCalendar.timeInMillis)}}
```

Этот метод вызывает метод `planNextReminder`, который сортирует напоминания по времени и определяет следующее напоминание. Если оно находится в будущем, создает будильник с помощью системного сервиса

AlarmManager [19], который позволяет приложению присылать уведомления, даже если в данный момент оно не активно или не запущено. Если напоминаний на сегодня нет, планируется первое напоминание на завтра.

Метод `createNotification` из класса `NotificationManager` используется для создания уведомления с помощью системного компонента Notification [20] с заданными параметрами, такими как заголовок, описание, цвет, изображение, а также с интентами для действий, такими как подтверждение приема и задержка напоминания. Метод представлен в листинге 14.

Листинг 14 – Метод `createNotification`

```
fun createNotification(

    context: Context,
    title: String,
    description: String,
    color: Int,
    bitmap: Bitmap?,
    pendingIntent: PendingIntent,
    confirmPendingIntent: PendingIntent,
    delayPendingIntent: PendingIntent,
    fullscreenPendingIntent: PendingIntent,
    channelId: String,
    whenMillis: Long
): Notification {
    val buttonDelay = Pref.buttonDelay

    val builder = NotificationCompat.Builder(context, channelId)
        .setSmallIcon(R.drawable.ic_pill)
        .setContentTitle(title)
        .setContentText(description)
        .setColor(color)
        .setPriority(NotificationCompat.PRIORITY_MAX)
        .setShowWhen(true)
        .setWhen(whenMillis)
        .addAction(
            NotificationCompat.Action(
                R.drawable.ic_check,
                context.getString(R.string.confirm),
                confirmPendingIntent))
        .addAction(

            NotificationCompat.Action(
                R.drawable.ic_delay,
                context.resources.getQuantityString(
                    R.plurals.delay,
                    buttonDelay,
                    buttonDelay),
                delayPendingIntent))return builder.build()}
```

Также в классе `NotificationManager` содержатся методы `createNotificationChannel` и `removeNotificationChannel`, которые создают

и удаляют каналы уведомлений соответственно. Каналы уведомлений были введены в Android 8.0 и представляют собой механизм для управления уведомлениями в приложениях. Каналы позволяют пользователю настраивать параметры уведомлений для каждого канала отдельно.

Когда срабатывает будильник, вызывается метод `createReminderNotification` из класса `ReminderUtil`. Этот метод позволяет задать уведомление с использованием данных о лекарстве и напоминании. Он использует метод `showNotification` из класса `NotificationManager`, который отображает информацию в уведомлении. Метод `createReminderNotification` представлен в листинге 15.

Листинг 15 – Метод `createReminderNotification`

```
fun createReminderNotification(context: Context, pill: Pill, reminder: Re-
minder) {
    NotificationManager.showNotification(
        context,
        title = pill.name,
        description = pill.getNotificationDescription(context, reminder),
        color = pill.color.getColor(context),
        bitmap = pill.photo,
        pendingIntent = getNotificationClickIntent(context, pill.id),
        confirmPendingIntent = getNotificationConfirmIntent(
            context,
            reminder.id,
            pill.id,
            reminder.getTodayMillis()),

        delayPendingIntent = getNotificationDelayIntent(
            context,
            reminder.id,
            Pref.buttonDelay.toLong() * 1000 * 60,
            reminder.getTodayCalendar().timeInMillis),
        fullscreenPendingIntent = getNotificationClickIntent(context,
pill.id),
        notificationId = reminder.id,
        channelId = pill.id.toString(),
        whenMillis = reminder.getTodayMillis())}
```

Также класс содержит методы создания интентов, таких как подтверждение напоминания, задержка напоминания, создание напоминания и повторного напоминания, а также переход по клику на уведомление в экран просмотра лекарства.

Все интенты и метод `createReminderNotification` используются в приемнике системных событий `ReminderReceiver`, который и создает уведомления.

Процесс создания уведомлений в системе представлен в виде диаграммы деятельности на рисунке 18.

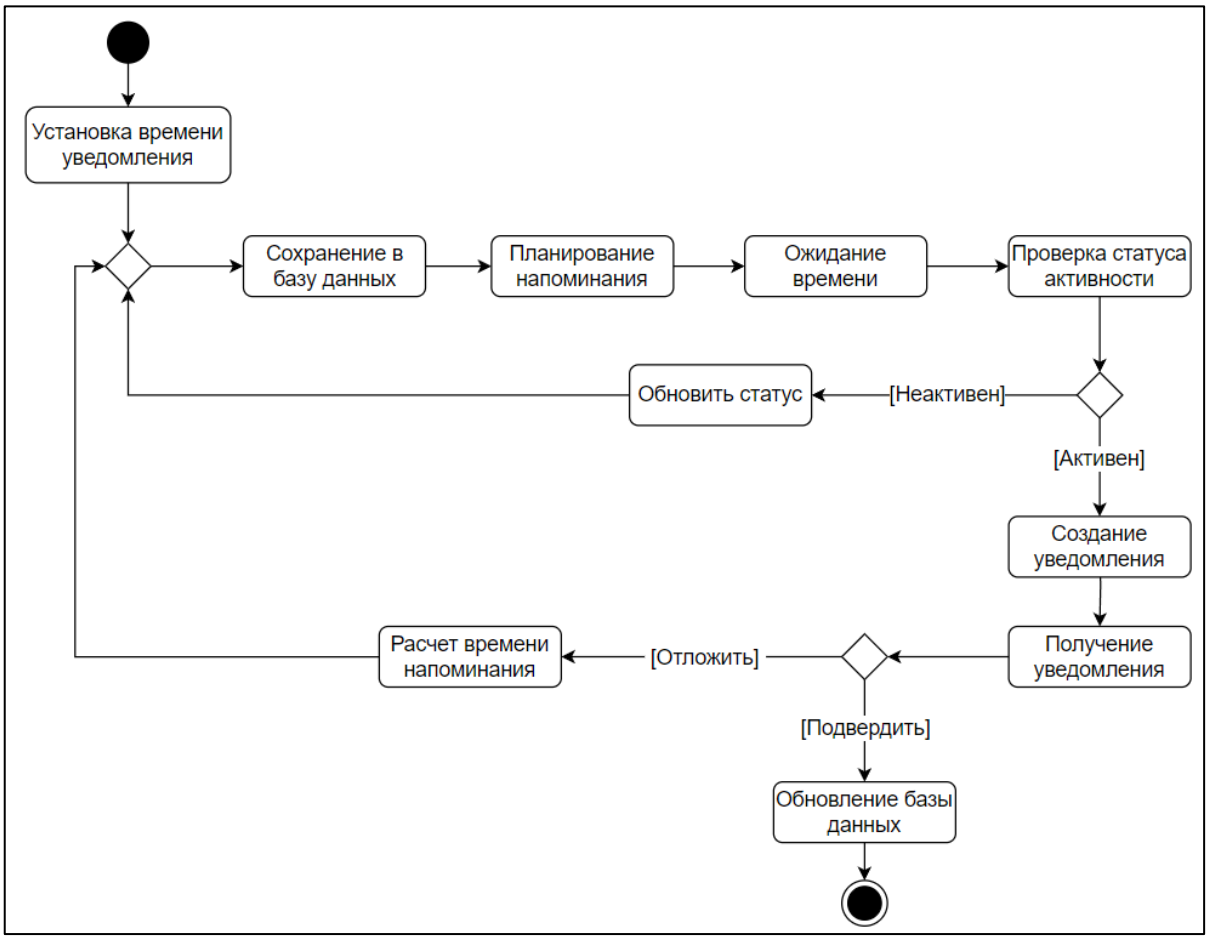

Рисунок 18 – Диаграмма деятельности создания уведомлений

Вывод по третьей главе

В третьей главе были приведены выбранные средства реализации и подробно описана реализация структуры базы данных, основного функционала и уведомлений. Разработанное приложение полностью соответствует сформированным требованиям на этапе проектирования.

4. ТЕСТИРОВАНИЕ

Для тестирования системы применялось функциональное тестирование, то есть тестирование программного обеспечения в целях проверки реализуемости функциональных требований. Тестирование проводилось на виртуальном устройстве с операционной системой Android версии 6.0 и на смартфоне с операционной системой Android версии 9.0. По результатам функционального тестирования построена таблица 2.

Таблица 2 – Набор тестов для функционального тестирования

№	Название теста	Действия	Ожидаемый результат	Тест пройден?
1	Отображение загрузочного экрана	Запуск приложения в первый раз	При запуске приложения появился загрузочный экран	Да
2	Добавление лекарства	1. Нажать на кнопку добавления 2. Ввести название, описание, выбрать цвет, фотографию, настроить уведомления и график приема 3. Нажать на кнопку сохранения	Лекарство добавилось в базу данных и появилось в списке на экране «Лекарства»	Да
3	Добавление фотографии	1. Нажать на кнопку добавления фотографии 2. Дать разрешение на доступ к галерее и к камере 3. Выбрать фотографию из галереи или сделать с камеры 4. Отредактировать фотографию 5. Нажать на кнопку сохранения	Фотография добавилась к лекарству	Да
4	Просмотр лекарства	Нажать на лекарство	Отображение названия, описания, фотографии, напоминания и времени последнего уведомления	Да
5	Изменение лекарства	1. Нажать на кнопку «Изменить» 2. Изменить параметры 3. Нажать на кнопку «Сохранить»	Параметры лекарства изменились и переписались в базе данных.	Да

Окончание таблицы 2

№	Название теста	Действия	Ожидаемый результат	Тест пройден?
6	Удаление лекарства	1. Нажать на кнопку «Удалить» 2. Нажать на кнопку «Удалить только лекарство»	Лекарство удалилось из списка лекарств, но осталась его история.	Да
7	Удаление лекарства и его истории	1. Нажать на кнопку «Удалить» 2. Нажать на кнопку «Удалить лекарство и историю»	Лекарство и его история удалилась из приложения.	Да
8	Просмотр истории из окна просмотра лекарства	Нажать на кнопку «История»	Появилось всплывающее окно с историей приема лекарства	Да
9	Редактирование окна истории	1. Нажать на кнопку меню действий 2. Отредактировать параметры	История изменилась и переписалась в базе данных	Да
10	Удаление одной записи в истории	1. Нажать на кнопку меню действий 2. Удалить запись в истории	Запись в истории удалена	Да
11	Отображение диаграмм	1. Перейти на экран «Статистика» 2. Перейти на вкладку «Диаграммы»	Диаграммы отобразились с процентной информацией	Да
12	Проверка отправки уведомления	Дождаться времени уведомления	Уведомление пришло на устройство в нужное время	Да
13	Проверка повторной отправки уведомления	Нажать на «Задержать на 5 минут»	Повторное уведомление пришло через 5 минут.	Да
14	Проверка изменения времени задержки в настройках	1. Перейти на экран «Настройки» 2. Изменить время задержки повторного напоминания	Повторное напоминание приходит через время, выбранное в настройках	Да
15	Изменение темы приложения	1. Перейти на экран «Настройки» 2. Изменить тему приложения	Тема изменилась на светлую или темную	Да

Вывод по четвертой главе

В четвертой главе разработанное мобильное приложение было протестировано набором функциональных тестов. Все тесты были выполнены успешно, поэтому можно сделать вывод, что приложение соответствует поставленным функциональным требованиям.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано Android-приложение для напоминаний о приеме медицинских препаратов. В процессе разработки были выполнены следующие задачи:

1) произведен анализ предметной области;
2) выполнено проектирование мобильного приложения;
3) выполнена реализация мобильного приложение;
4) проведено тестирование мобильного приложение.

В ходе выполнения выпускной квалификационной работы были изучены особенности разработки мобильных приложений для платформы Android, получен опыт работы с средой разработки Android Studio и языком программирования Kotlin.

В дальнейшем планируется продолжать разработку и улучшение мобильного приложения, внедряя новые функции и оптимизируя существующие. В первую очередь, будет добавлена аутентификация пользователей, что позволит каждому пользователю иметь собственный аккаунт. Также рассматривается возможность добавлять в аккаунт доступ для родственников или лечащих врачей. Кроме того, будет разработаны отслеживание запасов лекарств и напоминания о покупке лекарств, которые заканчиваются. В долгосрочной перспективе планируется подключить единую базу данных лекарств для просмотра информации о добавленном лекарстве.

ЛИТЕРАТУРА

1. Среднее экранное время и его использование по странам | ElectronicsHub. [Электронный ресурс] URL: https://www.electronicshub.org/the-average-screen-time-and-usage-by-country/ (дата обращения: 02.02.2024 г.).

2. Google Play vs Apple App Store | 42 Matters. [Электронный ресурс] URL: https://42matters.com/stats#available-apps-count (дата обращения: 02.02.2024 г.).

3. Соблюдение режима лечения | Справочник MSD. [Электронный ресурс] URL: https://www.msdmanuals.com/ (дата обращения: 04.02.2024 г.).

4. Официальный сайт приложения «MyTherapy». [Электронный ресурс] URL: https://www.mytherapyapp.com/ (дата обращения: 04.02.2024 г.).

5. Официальный сайт приложения «Medisafe». [Электронный ресурс] URL: https://www.medisafe.com/ (дата обращения: 04.02.2024 г.).

6. Pills Med Trecker | Google Play Market. [Электронный ресурс] URL: https://play.google.com/store/apps/details?id=mobilecreatures.pills-time&hl=en_US (дата обращения: 25.02.2024 г.).

7. Якобсон А., Ромбо Д., Буч Г. Язык UML. Руководство пользователя. // ДМК, 2006. – 432 с.

8. Model-View-VieModel | Geeksforgeeks. [Электронный ресурс] URL: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/ (дата обращения: 11.04.2024 г.).

9. Документация SQLite. [Электронный ресурс] URL: https://www.sqlite.org/docs.html (дата обращения: 11.04.2024 г.).

10. Save data in a local database using Room | Android Developers. [Электронный ресурс] URL: https://developer.android.com/training/data-storage/room (дата обращения: 11.04.2024 г.).

11. LiveData overview | Android Developers. [Электронный ресурс] URL: https://developer.android.com/topic/libraries/architecture/livedata (дата обращения: 20.04.2024 г.).

12. RecyclerView | Android Developers. [Электронный ресурс] URL: https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView (дата обращения: 20.04.2024 г.).

13. TabLayoutMediator | Android Developers. [Электронный ресурс] URL: https://developer.android.com/reference/com/google/android/material/tabs/TabLayoutMediator (дата обращения: 20.04.2024 г.).

14. Motion | Material Design. [Электронный ресурс] URL: https://m2.material.io/develop/android/theming/motion#fade (дата обращения: 20.04.2024 г.).

15. PopupMenu | Android Developers. [Электронный ресурс] URL: https://developer.android.com/reference/android/widget/PopupMenu (дата обращения: 20.04.2024 г.).

16. MPAndroidChart | GitHub. [Электронный ресурс] URL: https://github.com/PhilJay/MPAndroidChart (дата обращения: 10.05.2024 г.).

17. Preference | Android Developers. [Электронный ресурс] URL: https://developer.android.com/reference/android/preference/Preference (дата обращения: 10.05.2024 г.).

18. BroadcastReceiver | Android Developers. [Электронный ресурс] URL: https://developer.android.com/reference/android/content/BroadcastReceiver (дата обращения: 10.05.2024 г.).

19. AlarmManager | Android Developers. [Электронный ресурс] URL: https://developer.android.com/reference/android/app/AlarmManager (дата обращения: 10.05.2024 г.).

20. Notifications overview | Android Developers. [Электронный ресурс] URL: https://developer.android.com/develop/ui/views/notifications (дата обращения: 10.05.2024 г.).