

## СОДЕРЖАНИЕ

|   |    |
|---|----|
| Введение.....   | 6  |
| 1 Анализ исходных данных и постановка задач<br>на дипломное проектирование.....   | 7  |
| 1.1 Анализ и описание функциональных возможностей<br>программного средства .....  | 7  |
| 1.2 Обзор существующих программных средств по теме<br>дипломного проекта.....   | 9  |
| 1.3 Обоснование и описание выбора языка программирования, средств<br>разработки, используемых технологий и сторонних библиотек .....  | 14 |
| 1.4 Постановка задач на дипломное проектирование .....  | 16 |
| 2 Проектирование и разработка программного средства «SkinCare»<br>для контроля и советов по косметическому уходу.....   | 18 |
| 2.1 Проектирование архитектуры и описание состояний<br>программного средства .....  | 18 |
| 2.2 Разработка объектной модели программного средства .....   | 23 |
| 2.3 Проектирование и реализация способа хранения данных<br>программного средства .....  | 27 |
| 2.4 Проектирование и разработка графического интерфейса .....   | 31 |
| 2.5 Описание и реализация используемых в программном средстве<br>алгоритмов .....   | 38 |
| 3 Оценка количественных показателей функционирования<br>программного средства.....  | 41 |
| 3.1 Оценка временных показателей программного средства .....  | 41 |
| 3.2 Оценка ресурсных показателей программного средства.....   | 43 |
| 3.3 Оценка показателей надежности программного средства.....  | 50 |
| 4 Эксплуатация программного средства.....   | 54 |
| 4.1 Ввод в эксплуатацию и обоснование минимальных технических<br>требований к оборудованию .....  | 54 |
| 4.2 Руководство по эксплуатации программным средством .....   | 60 |
| 5 Техничко-экономическое обоснование разработки программного<br>средства «SkinCare» под операционную систему Android<br>для контроля и советов по косметическому уходу..... | 73 |
| 5.1 Характеристика программного средства .....  | 73 |
| 5.2 Расчет инвестиций (затрат) в разработку (модернизацию,<br>совершенствование) программного средства.....   | 73 |
| 5.3 Расчет результата от разработки и использования<br>программного средства .....  | 77 |
| 5.4 Расчет показателей экономической эффективности разработки<br>и использования программного средства на рынке .....   | 77 |
| Заключение .....  | 78 |
| Список использованных источников .....  | 79 |
| Приложение А (обязательное) Отчет по анализу заимствования<br>материала пояснительной записки .....   | 82 |
| Приложение Б (обязательное) Листинги программного кода.....   | 83 |
| Приложение В (обязательное) Ведомость дипломного проекта.....   | 96 |

## ВВЕДЕНИЕ

В современном мире человек не может обходиться без смартфона. Гаджеты заполнили нашу жизнь. Мобильные программные средства в смартфоне могут выполнять различные функции, облегчая рутинные дела. Например, вызвать такси или посмотреть расписание транспорта, послушать музыку или посмотреть новый фильм, для потребности любой сложности и разновидности есть программные средства в компактном мобильном устройстве. С каждым днем с ростом потребностей человека растет и количество программных средств. Из этого можно сделать вывод, что разработка программных средств для телефонов очень стремительно развивающаяся сфера. Темой дипломного проектирования выбрана разработка программного средства «SkinCare» под операционную систему Android для контроля и советов по косметическому уходу.

Кожа представляет собой самую большую часть организма, которая постоянно обновляется в течение всей жизни. Состояние кожи во многом зависит от возраста, питания и образа жизни. Лицо – самая открытая часть кожных покровов, и поэтому оно нуждается в постоянном уходе [1].

Наша кожа – это:

- общая площадь 1,5 кв.метра;
- 60% влаги;
- 100 пор на каждый кв.см.;
- 200 рецепторов на каждый кв.см.;

Таким образом, исходя из всего вышеперечисленного можно сделать вывод, что тема данного дипломного проектирования достаточно актуальна.

Задачей дипломного проектирования является разработка мобильного программного средства под операционную систему Android для контроля и советов по косметическому уходу.

Для решения данной задачи необходимо:

- определить технологии, используемые при разработке программного средства;
- спроектировать графический пользовательский интерфейс;
- спроектировать архитектуру мобильного программного средства;
- описать архитектуру мобильного программного средства;
- реализовать возможность определять свой тип кожи;
- реализовать возможность получать рекомендации по уходу;
- реализовать возможность получения информации по этапам ухода;

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности составляет 97,47%. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

Отчёт по анализу заимствования материала приведён в приложении А.

# **1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ НА ДИПЛОМНОЕ ПРОЕКТИРОВАНИЕ**

## **1.1 Анализ и описание функциональных возможностей программного средства**

На сегодняшний день огромное количество людей все чаще обращаются за помощью к косметологам за помощью избавиться от различных признаков старения кожи лица и тела. Однако большинства походов к косметологам и пластическим хирургам можно избежать, с помощью незамысловатого ухода за кожей в течение всей жизни. В прошлые десятилетия не было такого большого разнообразия «уходовой» косметики и знаний в этой области было критически мало. С появлением интернета все больше людей стало обращаться за советами в первую очередь к нему. Однако для того, чтобы найти хорошо структурированную информацию нужно перечитать очень много статей, разобраться в свойствах и компонентах средств, используемых в уходе. Таким образом, если человек действительно заинтересован в нахождении нужной информации, то он ее найдет, но как правило не многие доходят до конца и, как следствие, оставляют идею поиска рекомендаций и советов, а к врачу обращаться не хотят.

По личным наблюдениям, на данный момент декоративная косметика пользуется гораздо большим спросом, нежели «уходовая». Также на данный момент большинство подростков начинает использование декоративной косметики с 10-11 лет, совершенно не зная о том, как правильно ею пользоваться и как это сказывается на коже в дальнейшем. И как итог, по статистике, акне (воспалительные высыпания на лице и теле) страдает до 80% населения в возрасте от 12 до 25 лет, и примерно 30-40% лиц старше 25 лет. Из-за такой статистики подростки в школе все чаще сталкиваются с травлей в школе, что может привести к самым непредсказуемым последствиям. Однако этого можно избежать, просто введя в привычку и ежедневную рутину уход за лицом.

В целом уход за кожей лица достаточно индивидуален, по причине того, что у каждого человека разный тип кожи, разный подход к уходу и взгляд на косметологию. Это касается даже пола человека. Не секрет, что на данный момент женский пол значительно преобладает в уходе за кожей. Однако в последние все больше молодых людей интересуются и заботятся о своей внешности и качеством своей кожи лица, в частности.

Уход за кожей лица и тела с малых лет также важен, по причине того, что его отсутствие может привести к такой болезни как рак кожи. Например, в СНГ на 2017 год было около 617 тысяч заболевших [2]. Не все эти случаи связаны с тем, что у пациентов отсутствовал уход за кожей, однако даже эта цифра может быть куда меньше, если люди начнут использовать защиту от

солнца на ежедневной основе, откажутся от соляриев и идеи загорать летом под палящим солнцем до ожогов.

На рисунке 1.1, где изображена статистика количества программных средств в магазине приложений – PlayMarket, можно увидеть, что самой популярной категорией программных средств является «Games» – игры, а вот категория «Beauty» – красота, является самой непопулярной, наравне с такими категориями как «Comics» – комиксы, «Events» – мероприятия, «Medical» – медицина, «Parenting» – воспитание [3].

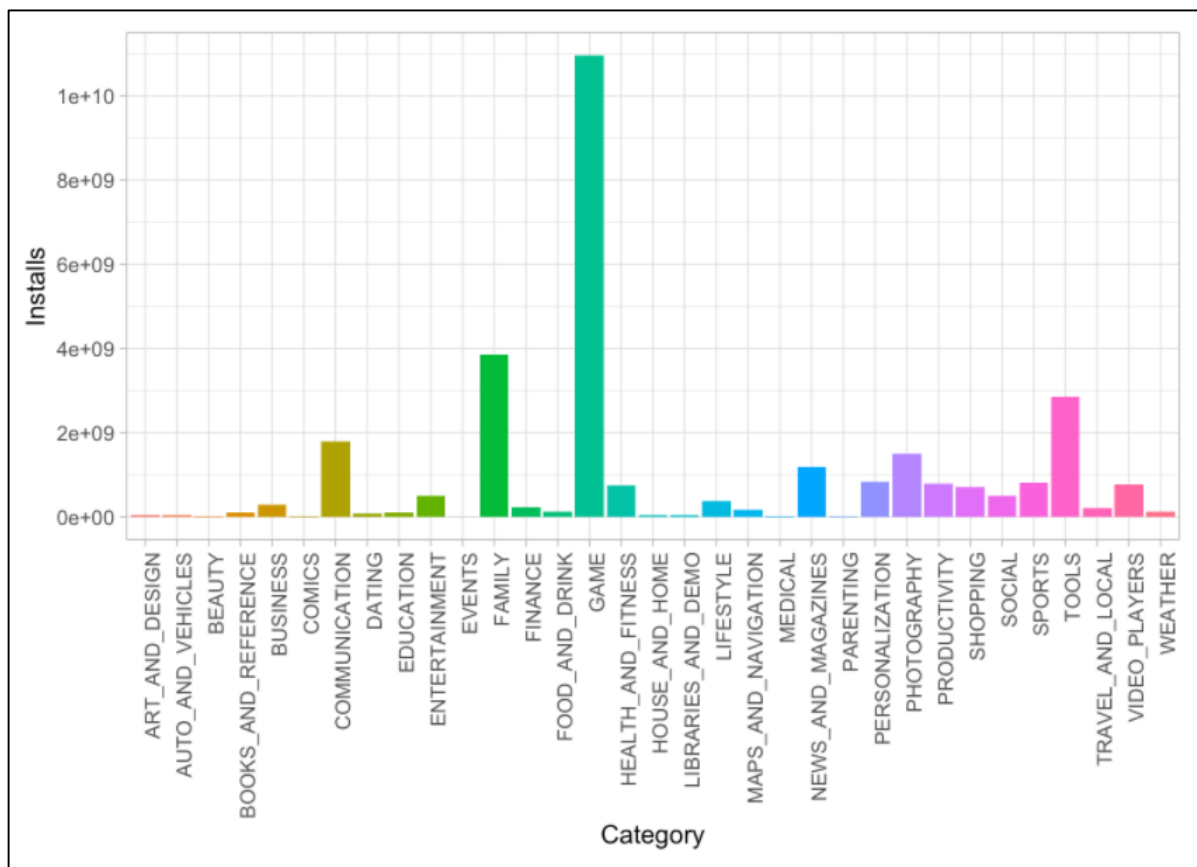


Рисунок 1.1 – Соотношение категорий программных средств в PlayMarket

Если категория комиксов, мероприятий и воспитания может быть актуальна не всем, но забота о собственной внешности и здоровье должна волновать каждого человека.

Проведя анализ рынка мобильных программных средств, предназначенных для контроля и советов по косметическому уходу, а также проведя опрос среди людей интересующихся данной темой, можно сделать вывод, что наиболее востребованными функциями в разрабатываемом программном средстве являются такие функции как:

- возможность определения своего типа кожи;
- возможность контроля ежедневного ухода;
- возможность получения индивидуальных рекомендаций по уходу;

- возможность подбора средств по типу кожи;
- возможность получения информации об этапах ухода;
- возможность контроля еженедельного ухода;
- возможность просматривать средства по шагам ухода;

При анализе рынка программных средств по данной теме были также и другие возможности программных средств, однако они не пользуются спросом. Приведенные выше функциональные возможности – это базис, которого достаточно для получения результата от использования программного средства.

Поскольку программное средство ориентировано в первую очередь на взаимодействие с пользователем, оно должно быть интуитивно понятным в использовании и максимально информативным.

## **1.2 Обзор существующих программных средств по теме дипломного проекта**

Более 74% всех мобильных устройств в мире работают на Android (данные StatCounter на июль 2021 года) [4]. Поэтому для данной операционной системы существует большое разнообразие программных средств для косметического ухода. В данном подразделе будут рассмотрены самые популярные из них.

Популярность программных средств в Play Market складывается из:

1 **Общего уровня установок.** Чем больше программное средство получает трафика из всех источников, тем выше оно будет ранжироваться по поисковым запросам.

2 **Дохода.** Чем больше прибыли приносит программное средство, тем выше ранжирование.

3 **Количества и частоты открытий программного средства.** Чтобы ранжирование было выше, необходимо именно запуск программного средства после его установки и соответственно тем чаще программное средство запускается, тем лучше.

4 **Рейтинга и количества отзывов.** При появлении программного средства в магазине приложений, оно находится на низких позициях, что не позволяет пользователям его находить, среди сотен других программных средств. Поэтому необходимо собрать как можно больше отзывов.

5 **Регулярность обновлений.** Чем чаще появляются необходимые пользователю функции, тем больше пользователь ценит программное средство, что не позволяет ему низко его оценивать или вовсе удалять.

6 **Количества удалений программного средства.** Чем раньше пользователь удалит программное средство, тем быстрее оно опускается ниже по списку в магазине приложений.

Одним из самых популярных программных средств является «TroveSkin» (рисунок 1.2). В магазине приложений Play Market данное программное средство насчитывает более 1 миллиона скачиваний и среднюю оценку 4,5 [5].

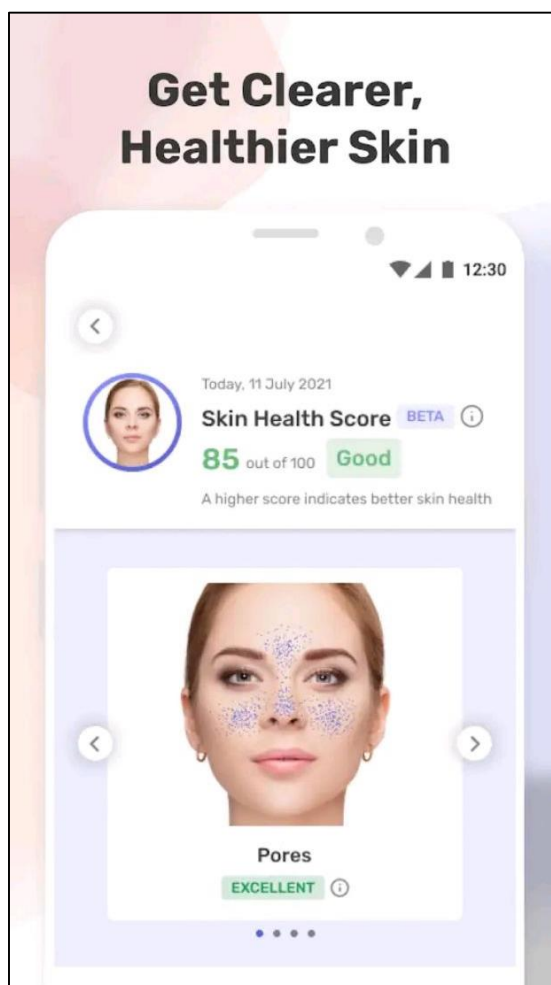


Рисунок 1.2 – Программное средство «TroveSkin»

Данное программное средство обладает следующими функциями:

- определение типа кожи по фотографии;
- подбор рекомендаций по уходу;
- возможность общения с другими пользователями;
- возможность ведения фото-ежедневника;
- возможность добавления продукты в избранное;
- возможность контроля ежедневного ухода;

На втором месте по популярности стоит программное средство «Picky – Skincare Community & Rewards» (рисунок 1.3). Данное программное средство было скачано более 100 тысяч раз, средняя оценка пользователей 5 [6].

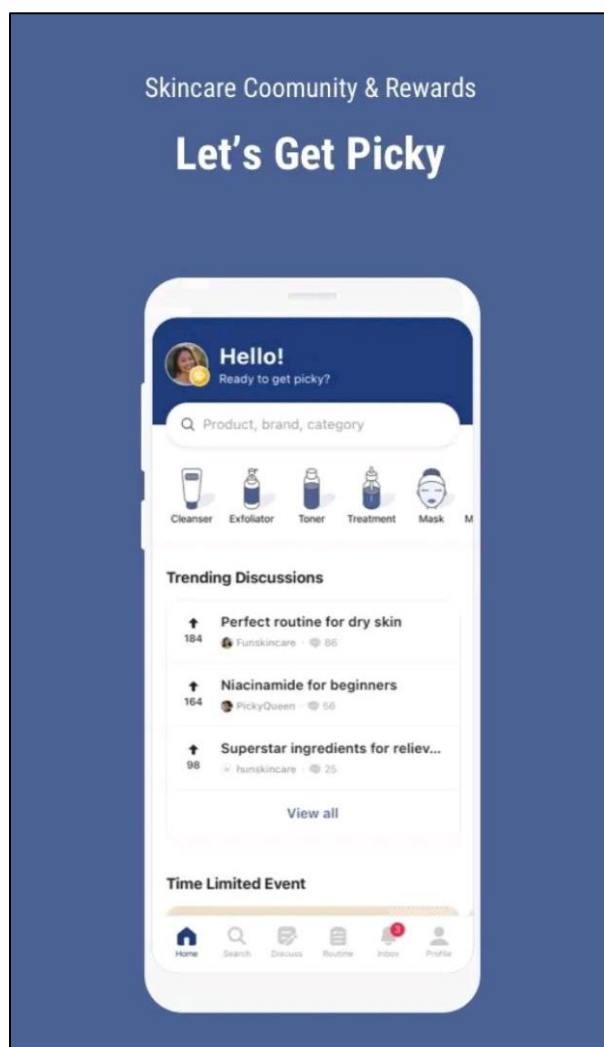


Рисунок 1.3 – Программное средство «Picky – Skincare Community & Rewards»

Данное программное средство обладает следующими функциями:

- подбор рекомендаций по уходу;
- поиск продуктов и производителей;
- возможность обсуждения продуктов с другими пользователями;
- возможность участия в розыгрышах продуктов;
- возможность читать отзывы на продукты;
- возможность оценивать продукты;
- возможность оставлять комментарии;
- возможность подбора продуктов по ингредиентам.

Следующим по популярности стоит программное средство «Skin Bliss: Cosmetics & Beauty» (рисунок 1.4). Программное средство скачано более 100 тысяч раз, средняя оценка программного средства 3,6 [7].



Рисунок 1.4 – Программное средство «Skin Bliss: Cosmetics & Beauty»

Данное программное средство обладает следующими функциями:

- подбор рекомендаций по уходу;
- подбор продуктов для разных типов кожи;
- возможность сканировать ингредиенты на упаковках;
- возможность получения информации о каждом ингредиенте;
- возможность покупки продуктов;
- возможность общения с другими пользователями.

Для удобства сравнения всех вышеперечисленных программных средств, все функции были собраны в единую таблицу, для выявления базовых и необходимых пользователю функций в разрабатываемом программном средстве. Сравнение возможностей приведенных программных средств представлено в таблице 1.1. Также, в таблице 1.1 указаны возможности разрабатываемого программного средства.



Таблица 1.1 – Таблица сравнения программных средств

| Доступные возможности                           | Название программного средства |                                       |                                |   |
|---|--------------------------------|---------------------------------------|--------------------------------|---|
|   | TroveSkin                      | Picky – Skin-care Community & Rewards | Skin Bliss: Cosmetics & Beauty | Разрабатываемое программное средство «SkinCare» |
| Подбор рекомендаций                             | Присутствует                   | Присутствует                          | Присутствует                   | Присутствует                                    |
| Определение типа кожи                           | Присутствует                   | Присутствует                          | Присутствует                   | Присутствует                                    |
| Возможность читать отзывы пользователей         | Присутствует                   | Присутствует                          | Отсутствует                    | Отсутствует                                     |
| Возможность общаться с другими пользователями   | Присутствует                   | Присутствует                          | Отсутствует                    | Отсутствует                                     |
| Возможность добавления продуктов в избранное    | Присутствует                   | Присутствует                          | Отсутствует                    | Присутствует                                    |
| Возможность оставлять комментарии               | Отсутствует                    | Присутствует                          | Отсутствует                    | Отсутствует                                     |
| Возможность контроля ежедневного ухода          | Присутствует                   | Отсутствует                           | Отсутствует                    | Присутствует                                    |
| Возможность получения информации по шагам ухода | Присутствует                   | Присутствует                          | Присутствует                   | Присутствует                                    |

Как видно из таблицы 1.1, возможности существующих мобильных программных средств контроля косметического ухода, в большинстве случаев, совпадают. Также, основываясь на полученной информации, можно сделать вывод о том, что самыми востребованными возможностями являются контроль ежедневного ухода, получение информации по шагам ухода, определение типа кожи и персональный подбор рекомендаций.

### **1.3 Обоснование и описание выбора языка программирования, средств разработки, используемых технологий и сторонних библиотек**

Тема дипломного проектирования подразумевает разработку программного средства для операционной системы (ОС) Android. Для разработки программных средств для данной ОС, существует специализированное средство разработки как Android Studio. Android Studio – это интегрированная среда разработки (IDE) для работы с платформой Android, выпущенная компанией Google. Данная среда разработки была выбрана по ряду нескольких преимуществ:

- 1 Удобный дизайн.
- 2 Удобный конструктор интерфейсов, позволяющий просматривать отображение экрана на любом устройстве, вплоть до телевизоров и часов. Элементы интерфейса отображаются прямо так, как они будут выглядеть на определенной версии операционной системы.
- 3 Встроенный комплекс средств разработки SDK (software development kit), выдает уведомление с установкой необходимого API для запуска старого проекта.
- 4 Удобная структура проекта.
- 5 Наличие логов для отслеживания ошибок, процессов и потоков.
- 6 Наличие достаточно большого количества литературы на русском языке.

Затем после выбора среды разработки необходимо выбрать язык программирования. В самом начале работы с Android Studio платформа предлагает два языка: Java и Kotlin. На старте разработки программного средства было решено использовать язык Java.

Java – самый популярный язык, используемый для разработки программных средств под Android. Android-устройства используют собственные оптимизированные форматы скомпилированного кода. Это означает, что нельзя воспользоваться обычной средой разработки на языке Java – также понадобятся специальные инструменты для преобразования откомпилированного кода в формат Android, установки его на Android-устройствах и отладки программного средства, когда оно заработает. Все необходимое содержится в Android SDK. Android SDK содержит библиотеки и инструменты, необходимые для разработки Android-приложений:

- 1 SDK Platform. Отдельная платформа для каждой версии Android.
- 2 SDK Tools. Инструменты отладки и тестирования, а также другие полезные служебные программы. Также включает набор платформеннозависимых инструментов.

3 Примеры программных средств. Если вы захотите просмотреть реальные примеры кода, чтобы лучше понять, как пользоваться API, примеры программных средств вам в этом помогут.

4 Документация. Предоставляет автономный доступ к новейшей документации API.

5 Вспомогательный инструментарий Android. Дополнительные API, отсутствующие в стандартной платформе.

6 Google Play Billing. Интеграция сервиса биллинга в программные средства.

Разработчики программного обеспечения выбирают Java потому что технология Java протестирована, усовершенствована, расширена и проверена участниками сообщества разработчиков Java, архитекторов и энтузиастов. Java позволяет разрабатывать высокопроизводительные портативные программные средства практически на всех компьютерных платформах. Доступность программных средств в разнородных средах позволяет компаниям предоставлять более широкий спектр услуг, способствует повышению производительности, уровня взаимодействия и совместной работы конечных пользователей и существенному снижению стоимости совместного владения корпоративными и потребительскими программными средствами. Java стала незаменимым инструментом для разработчиков и открыла возможность создания многофункциональных и эффективных программных средств для мобильных телефонов и практически любых других категорий электронных устройств [8].

Однако в процессе разработки было принято решение перейти на язык разработки Kotlin. По причине того, что сам по себе данный язык занимает гораздо меньше времени при написании кода и является чуть более функциональным. Оба языка могут работать совмещенно, поэтому код, который был написан на Java нет необходимости менять.

Kotlin – это язык программирования с открытым исходным кодом, который может работать на виртуальной машине Java (JVM). Это язык, который сочетает в себе объектно-ориентированное программирование (ООП) и функциональное программирование на неограниченной, самодостаточной и самобытной платформе. Также Kotlin имеет ряд таких преимуществ как:

1 Повышенная производительность. Создатели Kotlin достигли улучшенной производительности за счет интуитивно понятного синтаксиса. Чтобы написать программу уже нужно меньше строк, а значит и не так много времени. Таким образом, конечный результат достигается гораздо быстрее.

2 Полная совместимость с Java. Это значит, что можно функции Kotlin вызывать из Java, а методы Java – из Kotlin. Прекрасная новость и для программистов, и для крупных компаний, работающих с объемными базами данных на Java.

3 Простая поддержка. Kotlin встроен в большинство популярных IDE (например, в Android studio и некоторые другие SDK). Поэтому у

Android-разработчиков не возникает никаких трудностей при поддержке кода. Тем более, что набор инструментов остается привычным.

4 Официальная поддержка Android Studio. Kotlin поддерживается Android Studio, а за счет инструментов адаптации можно одновременно применять два языка: и Java, и Kotlin [9].

Таким образом использование данных языков программирования позволило реализовать сложный функционал с минимальным количеством кода.

При разработке программных средств существует 2 типа технологии:

- 1 Нативная разработка.
- 2 Кроссплатформенная разработка.

Основное отличие данных технологий состоит в том, что нативная разработка применяется, когда разработка ведется под конкретную платформу на конкретных языках поддерживаемых данной платформой, а кроссплатформенная разработка ведется сразу на несколько платформ. Например, разработка приложений под IOS происходит в среде разработки XCode на языке Swift (а раньше – на Objective-C). При использовании технологии разработки мобильных приложений на платформе андроид используется среда Android Studio и язык Kotlin (до 2018 года основным языком был Java). Поэтому исходя из задания дипломного проектирования, была выбрана именно нативная технология разработки.

Также при разработке программного средства были использованы сторонние библиотеки, некоторые из них приведены в списке:

1 Material Design – это адаптируемая система руководств, компонентов и инструментов, поддерживающая лучшие практики проектирования пользовательского интерфейса.

2 Glide – это популярная библиотека Android с открытым исходным кодом для загрузки изображений, видео и анимированных GIF. С Glide можно загружать и отображать медиа из разных источников, например, удаленных серверов или из локальной файловой системы.

3 Android KTX – набор расширений, предназначенных для того, чтобы сделать написание кода Kotlin для Android более кратким. Android KTX предоставляет хороший слой API поверх фреймворка Android и библиотеки поддержки, чтобы сделать написание кода Kotlin более естественным.

Таким образом при использовании данных библиотек получим визуально красивое программное средство с подгружаемыми с удаленного сервера изображениями, при этом не тратя время на написание большого количества кода.

#### **1.4 Постановка задач на дипломное проектирование**

Разработка любого программного средства начинается с постановки задач, которые должны быть решены в ходе работы, для того чтобы получить

конечный продукт. В ходе дипломного проектирования необходимо разработать программное средство «SkinCare» под операционную систему Android для контроля и советов по косметическому уходу. Для этого необходимо решить следующие задачи:

1 Выбрать язык программирования и среду разработки. Привести описание выбранного языка программирования и среды разработки. Обосновать выбор языка программирования и среды разработки.

2 Произвести анализ рынка мобильных программных средств контроля и советов по косметическому уходу под операционную систему Android. Привести описание рассматриваемых программных средств для контроля и советов по косметическому уходу. Составить сравнительную таблицу возможностей каждого из рассматриваемых программных средств для контроля и советов по косметическому уходу.

3 Выделить наиболее востребованные возможности существующих программных средств ухода за кожей.

4 Разработать объектную модель программного средства для контроля и советов по косметическому уходу. Привести описание основных классов и их методов.

5 Разработать алгоритмы функционирования программного средства для контроля и советов по косметическому уходу.

6 Разработать и обосновать пользовательский интерфейс программного средства для контроля и советов по косметическому уходу.

7 Произвести оценку количественных показателей функционирования программного средства.

8 Произвести оценку временных показателей программного средства.

9 Произвести оценку ресурсных показателей программного средства.

10 Произвести оценку показателей надежности программного средства.

11 Разработать руководство к пользованию программным средством для контроля и советов по косметическому уходу. В руководстве к пользованию предоставить изображения экранов программного средства, а также описание к каждому из них.

12 Произвести технико-экономическое обоснование разработки программного средства для контроля и советов по косметическому уходу. Описать функции, назначение и потенциальных пользователей программного средства. Оценить экономический эффект от разработки программного средства для реализации на рынке информационных технологий.

Таким образом, если при разработке будут выполнены вышестоящие задачи, то по итогу будет получено программное средство для контроля и советов по уходу за кожей, работающее без сбоев в работе, транслирующий правильную информацию в любой момент времени, а также отвечающий всем требованиям безопасности.

## 2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА «SKINCARE» ДЛЯ КОНТРОЛЯ И СОВЕТОВ ПО КОСМЕТИЧЕСКОМУ УХОДУ

### 2.1 Проектирование архитектуры и описание состояний программного средства

Программное средство разрабатывалось с использованием шаблона MVVM (Model-View-ViewModel). Такой шаблон позволяет четко разделить бизнес-логику и представление представления программного средства от пользовательского интерфейса. Поддержание чистого разделения логики программного средства и пользовательского интерфейса помогает устранить многочисленные проблемы разработки и упростить тестирование, обслуживание и развитие программного средства. Она также может значительно улучшить возможности повторного использования кода и позволяет разработчикам и дизайнерам пользовательского интерфейса упростить совместную работу при разработке соответствующих частей программного средства.

В шаблоне MVVM есть три основных компонента: модель, представление и модель представления. Каждый из них обслуживает отдельную цель. На рисунке 2.1 показаны связи между тремя компонентами.

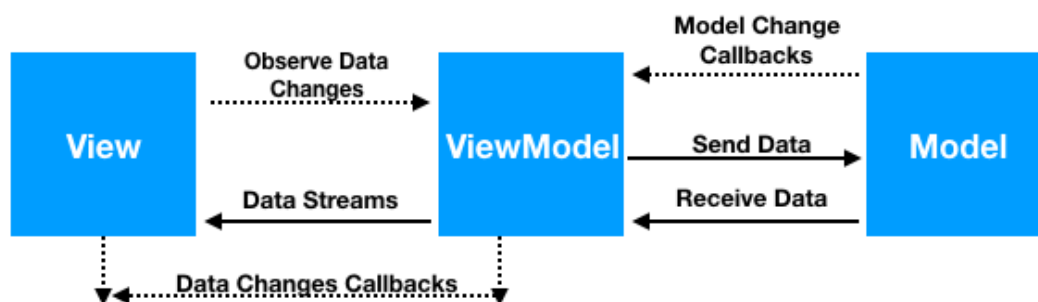


Рисунок 2.1 – Шаблон MVVM

Кроме понимания обязанностей каждого компонента, важно понимать, как они взаимодействуют друг с другом. На высоком уровне в представлении «известно о модели представления и модели представления» известно о модели, но модель не знает модель представления, а модель представления не знает об этом представлении. Таким образом, модель представления изолирует представление от модели и позволяет модели развиваться независимо от представления.

Ниже приведены преимущества использования шаблона MVVM.

1 Если реализована существующая реализация модели, которая инкапсулирует существующую бизнес-логику, она может быть сложной или рискованной для ее изменения. В этом сценарии модель представления выступает в

качестве адаптера для классов модели и позволяет избежать внесения значительных изменений в код модели.

2 Разработчики могут создавать модульные тесты для модели представления и модели без использования представления. Модульные тесты для модели представления могут выполнять точно те же функциональные возможности, которые используются в представлении.

3 Пользовательский интерфейс программного средства можно переконструировать, не затрагивая код, при условии, что представление полностью реализовано в XML. Поэтому Новая версия представления должна работать с существующей моделью представления.

4 Разработчики могут одновременно работать с компонентами в процессе разработки. Дизайнеры могут сосредоточиться на представлении, тогда как разработчики могут работать над моделью представления и компонентами модели.

Ключ к использованию MVVM эффективно полагается на понимание того, как разделить код программного средства на правильные классы и как понять, как взаимодействуют классы. В следующих разделах рассматриваются обязанности каждого класса в шаблоне MVVM.

Представление отвечает за определение структуры, макета и внешнего вида того, что видит пользователь на экране. В идеале каждое представление определяется в XML-файлах с ограниченным кодом программной части, который не содержит бизнес-логику. Однако в некоторых случаях код программной части может содержать логику пользовательского интерфейса, которая реализует визуальное поведение, которое сложно выразить в XML, например, анимации.

Модель представления реализует свойства и команды, к которым может быть привязано представление данных, и уведомляет представление о любых изменениях состояния с помощью событий уведомления об изменениях. Свойства и команды, предоставляемые моделью представления, определяют функциональные возможности, предоставляемые пользовательским интерфейсом, но представление определяет, как эти функции должны отображаться.

Модель представления также отвечает за координацию взаимодействия представления с любыми необходимыми классами модели. Как правило, существует связь «один ко многим» между моделью представления и классами модели. Модель представления может предоставить доступ к классам модели непосредственно в представлении, чтобы элементы управления в представлении могли привязывать данные к ним напрямую. В этом случае классы модели должны быть спроектированы для поддержки привязки данных и событий уведомления об изменении.

Каждая модель представления предоставляет данные из модели в форме, которую представление может легко использовать. Для этого модель пред-

ставления иногда выполняет преобразование данных. Размещение этого преобразования данных в модели представления является хорошей идеей, поскольку она предоставляет свойства, к которым может быть привязано представление. Например, модель представления может сочетать значения двух свойств, чтобы упростить отображение представления.

Классы моделей – это классы, не являющиеся визуальными, которые инкапсулируют данные программного средства. Таким образом, модель можно рассматривать как представление модели предметной области программного средства, которая обычно включает в себя модель данных, а также логику бизнес-процессов и проверки [10].

В разрабатываемом программном средстве присутствует каждый компонент из данной архитектуры. Примером компонента Model является класс Product.

Листинг исходного кода класса Product, используемый в разрабатываемом программном средстве:

```
package com.example.myapplication.model

import android.os.Parcelable
import com.google.firebase.database.DataSnapshot
import kotlinx.parcelize.Parcelize

@Parcelize
data class Product(
    var id: String = "",
    val name: String = "",
    val description: String = "",
    val type: Int = 0,
    val price: String = "",
    val image: String = ""
) : Parcelable {
    companion object Types {
        const val Spf = 0
        const val Cleaner = 1
        const val Serum = 2
    }
}

fun DataSnapshot.mapToProducts(): List<Product> {
    return children.mapNotNull { dataSnapshot ->
        dataSnapshot.getValue(Product::class.java)?.apply {
            id = dataSnapshot.key.orEmpty()
        }
    }
}
```

Примером компонента View является класс ProductsFragment. Листинг класса ProductsFragment:

```
package com.example.myapplication.ui.product

class ProductsFragment : Fragment(R.layout.fragment_products) {
    private val binding by viewBinding<FragmentProductsBinding>::bind
    private val spfAdapter = ProductsAdapter(::handleProductClick)
    private val cleanerAdapter = ProductsAdapter(::handleProductClick)
    private val serumAdapter = ProductsAdapter(::handleProductClick)
```



```

        override fun onCreateView(view: View, savedInstanceState: Bundle?) {
            super.onCreateView(view, savedInstanceState)
            setupView()
            loadProducts()
        }

        private fun setupView() {
            with(binding) {
                spfList.adapter = spfAdapter
                cleansersList.adapter = cleanerAdapter
                serumsList.adapter = serumAdapter
            }
        }

        private fun loadProducts() {
            viewLifecycleOwner.lifecycleScope.launch(Dispatchers.IO) {
                Firebase.database.getReference("Products")
                    .get()
                    .addOnCompleteListener { binding.progress.isVisible = false }
                    .addOnSuccessListener { dataSnapshot -> showProducts(dataSnapshot.mapToProducts()) }
                    .addOnFailureListener { showSnackbar(it.localizedMessage.orEmpty()) }
            }
        }

        private fun showProducts(list: List<Product>) {
            viewLifecycleOwner.lifecycleScope.launch(Dispatchers.Default) {
                val spfList = list.filter { it.type == Product.Spf }
                val cleanerList = list.filter { it.type == Product.Cleaner }
                val serumList = list.filter { it.type == Product.Serum }

                withContext(Dispatchers.Main) {
                    spfAdapter.submitList(spfList) {
                        binding.cleansersTitle.isVisible = true
                        binding.cleansersList.isVisible = true
                    }
                    cleanerAdapter.submitList(cleanerList) {
                        binding.spfTitle.isVisible = true
                        binding.spfList.isVisible = true
                    }
                    serumAdapter.submitList(serumList) {
                        binding.serumsTitle.isVisible = true
                        binding.serumsList.isVisible = true
                    }
                }
            }
        }

        private fun handleProductClick(product: Product) {
            val directions = ProductsFragmentDirections.actionFragmentProductsToFragmentProductDetails(product)
            findNavController().navigate(directions)
        }
    }
}

```

И последним компонентом в архитектуре MVVM является ViewModel. Примером данного компонента является класс ProductsAdapter. Листинг класса ProductsAdapter:

```

package com.example.myapplication.adapter

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.example.myapplication.databinding.ItemProductBinding
import com.example.myapplication.model.Product

class ProductsAdapter(
    private val clickListener: (product: Product) -> Unit
) : ListAdapter<Product, ProductsAdapter.ProductViewHolder>(Comparator) {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
    ProductViewHolder {
        val binding = ItemProductBinding.inflate(LayoutInflater.from(par-
ent.context), parent, false)
        return ProductViewHolder(binding)
    }

    override fun onBindViewHolder(holder: ProductViewHolder, position: Int) {
        holder.bind(currentList[position])
    }

    inner class ProductViewHolder(
        private val binding: ItemProductBinding
    ) : RecyclerView.ViewHolder(binding.root) {
        fun bind(item: Product) {
            with(binding) {
                name.text = item.name
                price.text = item.price
                Glide.with(binding.root.context)
                    .load(item.image)
                    .into(binding.image)
                root.setOnClickListener { clickListener(item) }
            }
        }
    }

    companion object Comparator : DiffUtil.ItemCallback<Product>() {
        override fun areItemsTheSame(oldItem: Product, newItem: Product):
        Boolean {
            return oldItem == newItem
        }

        override fun areContentsTheSame(oldItem: Product, newItem: Product):
        Boolean {
            return oldItem == newItem
        }
    }
}

```

По итогу данного раздела мы рассмотрели, как наглядно выглядит работа архитектуры Model-View-ViewModel в разрабатываемом программном средстве.

## 2.2 Разработка объектной модели программного средства

В начале работы программного средства пользователь попадает на первый экран авторизации. Если пользователь не авторизован, для него есть возможность зарегистрироваться. Оба эти действия выполняют классы Login и SignUp соответственно (рисунок 2.2)

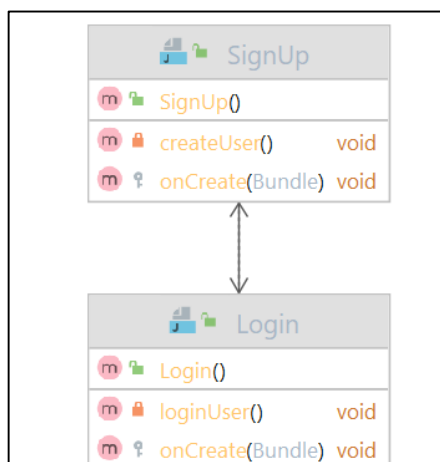


Рисунок 2.2 – Классы Login и SignUp

После прохождения авторизации пользователь попадает на главный экран программного средства. На данном экране пользователь может реализовать ежедневный и еженедельный контроль ухода. Данные функции реализованы с помощью главного класса HomeFragment (рисунок 2.3).

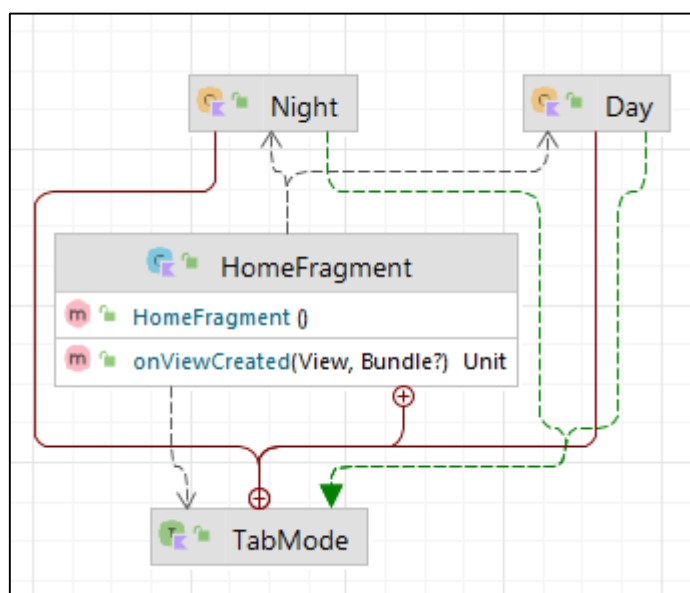


Рисунок 2.3 – Класс HomeFragment

Затем у пользователя есть возможность просмотреть продукты для качественного ухода. Продукты разделены на категории и реализованы в программном средстве с помощью таких классов как: Product, ProductsAdapter, ProductsFragment, ProductDetailsFragment (рисунок 2.4).

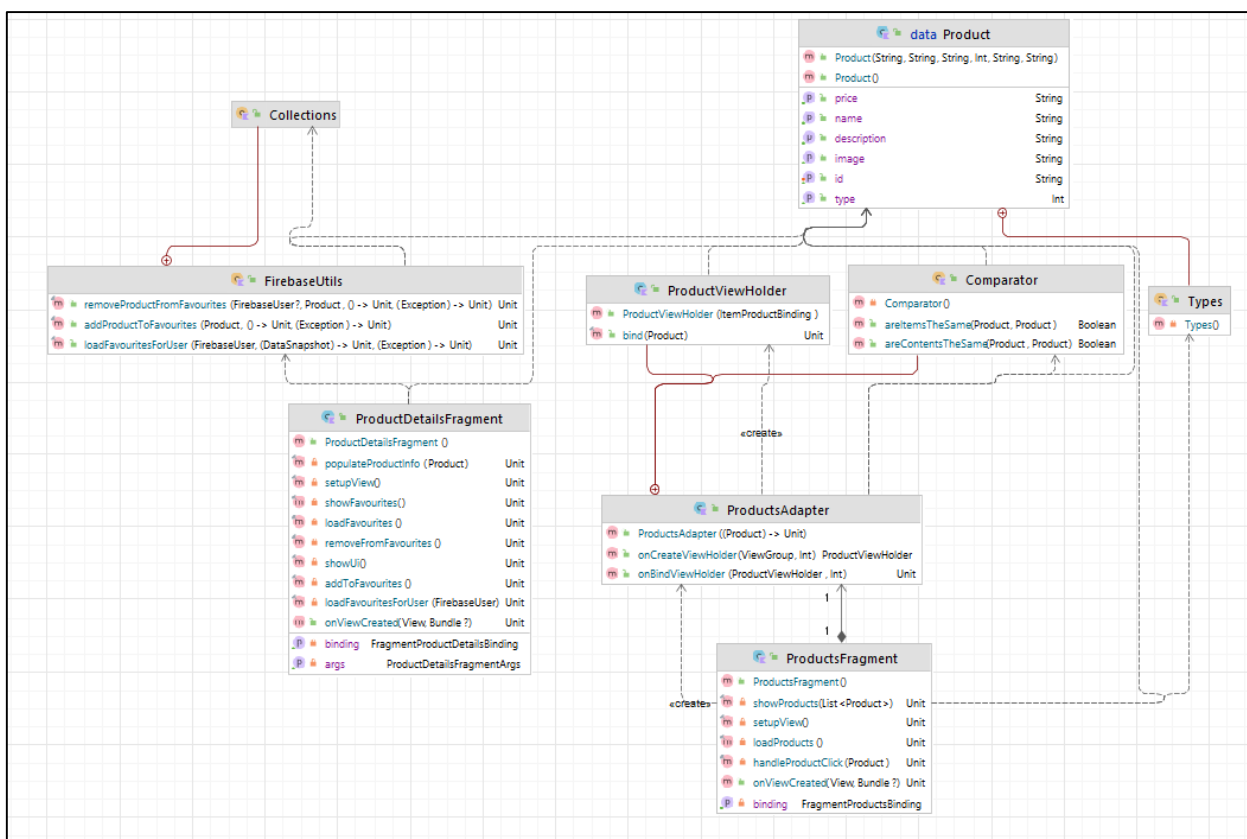


Рисунок 2.4 – Классы Product, ProductsAdapter, ProductsFragment, ProductDetailsFragment и их взаимосвязи

Также у пользователя есть возможность добавлять, удалять и просматривать избарные продукты. Для этого используется лишь две кнопки на экране ProductDetailsFragment, при нажатии кнопки «В избранное», она меняется на «Из избранного» и наоборот, что дает возможность сократить количество элементов на экране, чтобы пользователю было удобнее и комфортнее использовать программное средство. Также на экране ProductDetailsFragment возможность просмотра списка избранных продуктов реализована с помощью одной кнопки «Избранное», при нажатии на которую, соответственно, происходит переход на экран с избранным. После чего пользователь, нажав на кнопку «Back» своего устройства, может вернуться обратно к деталям выбранного продукта. Данная возможность реализована с помощью класса Favourite, FavouritesFragment и методов addToFavourites(), removeFromFavourites() и showFavourites() (рисунки 2.5 – 2.6).

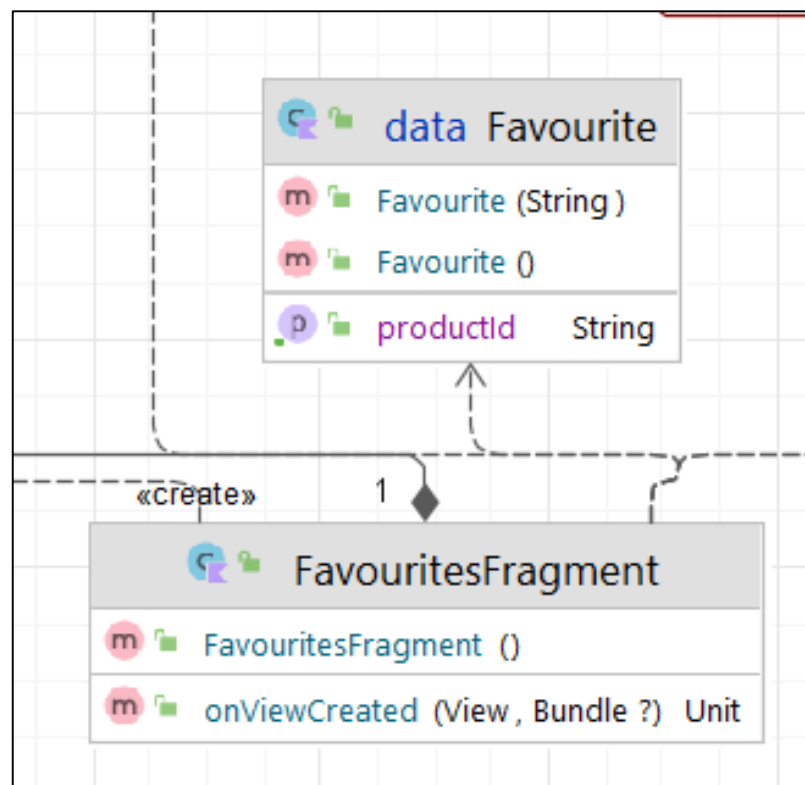


Рисунок 2.5 – Класс Favourite

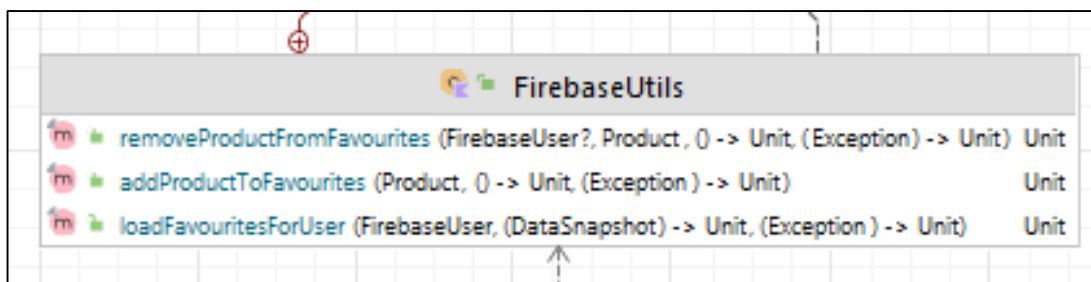


Рисунок 2.6 – Методы addToFavourites(), removeFromFavourites() и showFavourites()

При разработке программного средства была реализована возможность прохождения тестирования на определение типа кожи и получение персональных рекомендаций, на основе ответов, полученных от пользователя. Тестирование было реализовано с помощью классов Question, TestViewHolder, TestAdapter, TestFragment (рисунок 2.7).

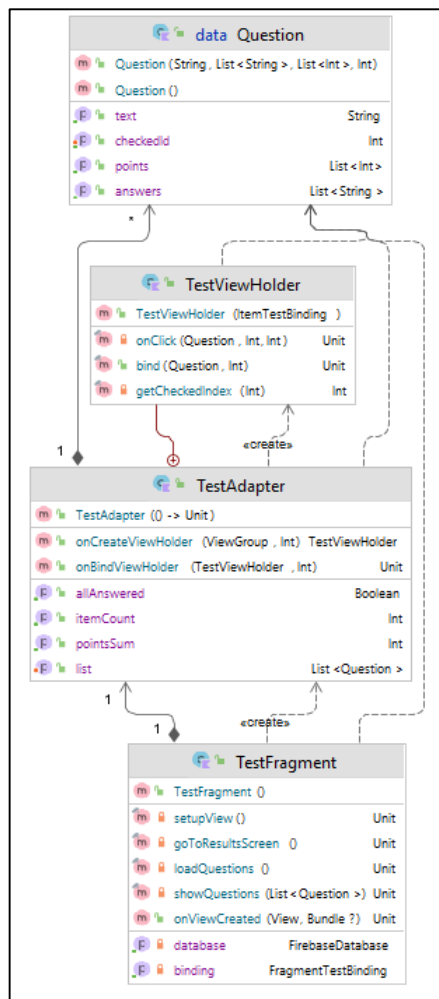


Рисунок 2.7 – Реализация теста на определения типа кожи

Для вывода результатов тестирования были реализованы классы **TestResultProvider**, **TestResultFragment** (рисунок 2.7).

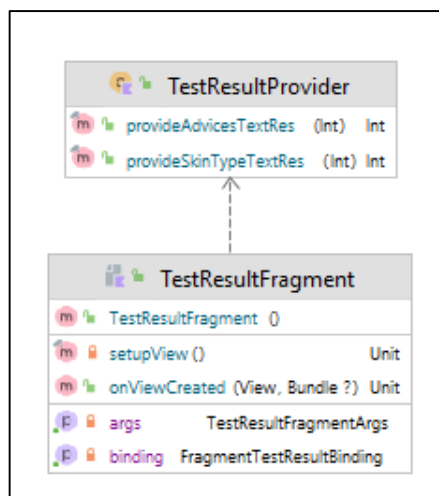


Рисунок 2.8 – Реализация вывода результатов тестирования

Навигация в программном средстве реализована с помощью класса MainActivity (рисунок 2.9).

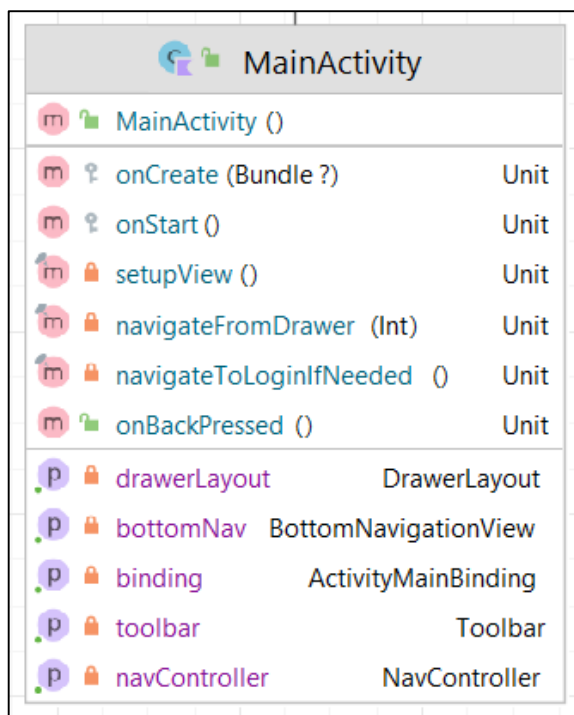


Рисунок 2.9 – Класс Main

В данном подразделе была показана объектная модель программного средства.

### 2.3 Проектирование и реализация способа хранения данных программного средства

База данных (БД) – объективная форма представления совокупности данных, систематизированных так, чтобы эти данные были найдены и обработаны с помощью ЭВМ. Для управления, изменения и администрирования базы данных используется система управления базами данных (СУБД). Каждая СУБД поддерживает различные модели и структуры БД.

Одной из самых популярных моделей базы данных является реляционная модель. Реляционная модель предоставляет способы четкого структурирования данных. В ней используются таблицы с колоннами и рядами. У каждого ряда таблицы (записи) есть уникальный номер (первичный ключ), а колонны служат полями данных [12].

Вторая модель базы данных – NoSQL. В данной модели структура данных не регламентирована – в отдельной строке или документе можно добавить произвольное поле без предварительного декларативного изменения структуры всей таблицы. К данной модели относится облачная база данных Firebase,

которая и была выбрана в качестве базы данных для разрабатываемого программного средства.

Firebase – это платформа для разработки веб-приложений и мобильных программных средств. Она была создана Firebase, Inc в 2011 году и приобретена в 2014 году Google.

SDK Firebase позволяет разработчикам сосредоточиться на разработке клиентской части программного средства, что сказывается на качестве разрабатываемого ПО и впечатлениях пользователей [13].

Возможности Firebase делятся на несколько категорий:

1 Сборка: базы данных (Firestore + RTDB); машинное обучение; облачные функции; аутентификация; облачные сообщения; хостинг; хранилище.

2 Выпуск и мониторинг: crashlytics; аналитика; мониторинг производительности; песочница; дистрибутивы программных средств.

3 Вовлечение: удаленная настройка; прогнозирование; A/B тестирование; динамические ссылки; обмен сообщениями.

Все вышеперечисленные функции необязательно использовать в одном разрабатываемом программном средстве. Можно использовать только то, что необходимо для реализации определенных задач или для хранения конкретной информации. Для того, чтобы использовать все необходимые функции в разрабатываемом программном средстве в среде разработки Android Studio необходимо подключить саму базу данных Firebase.

Для начала необходимо создать проект в консоли БД (рисунок 2.10 ).

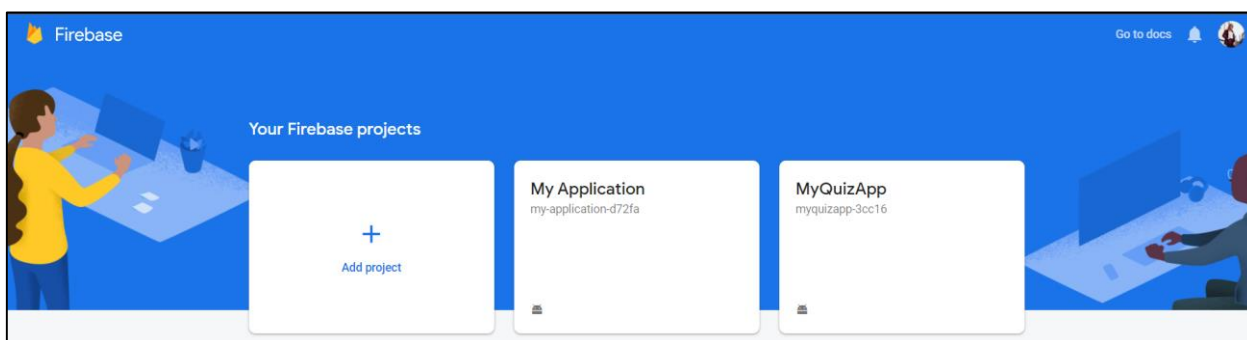


Рисунок 2.10 – Создание проекта в Firebase

Однако создания проекта в Android Studio и в Firebase недостаточно, необходимо их соединить. Сделать это можно с помощью внутренних SDK инструкций. Основное что необходимо сделать, это поместить уникальный, для каждого созданного программного средства Firebase, сгенерированный json-файл. И в gradle-файлы Android Studio поместить инструкции для выполнения.



В файл проекта build.gradle:

```
dependencies {classpath 'com.google.gms:google-services:4.3.10'}
```

И в файл программного средства build.gradle:

```
implementation platform('com.google.firebase:firebase-bom:29.0.0')
```

После настройки можно приступать к реализации необходимых для программного средства функции.

По заданию дипломного проектирования с помощью Firebase можно реализовать:

- авторизацию и регистрацию пользователя;
- ввод пользователем информации по дневному и вечернему уходу;
- функцию контроля (уведомлений) еженедельного ухода;
- функцию определения типа кожи, посредством тестирования.

Регистрацию и авторизацию пользователя можно реализовать с помощью функции Firebase Authentication. Знание личности пользователя позволяет программному средству безопасно сохранять пользовательские данные в облаке и обеспечивать одинаковую персонализированную работу на всех устройствах пользователя.

Для того, чтобы реализовать регистрацию в своем программном средстве необходимо также подключить ее и в Firebase, и в Android Studio.

В файле app/build.gradle добавляем:

```
dependencies {  
    implementation 'com.google.firebase:firebase-auth'  
}
```

Далее необходимо создать класс и в нем инициализировать метод создания пользователя:

```
private void createUser(){  
    String email = etRegEmail.getText().toString();  
    String password = etRegPassword.getText().toString();  
  
    if (TextUtils.isEmpty(email)){  
        etRegEmail.setError("Email cannot be empty");  
        etRegEmail.requestFocus();  
    }else if (TextUtils.isEmpty(password)){  
        etRegPassword.setError("Password cannot be empty");  
        etRegPassword.requestFocus();  
    }else{  
        mAuth.createUserWithEmailAndPassword(email,password).addOnCompleteListener(new OnCompleteListener<AuthResult>() {  
            @Override  
            public void onComplete(@NonNull Task<AuthResult> task) {  
                if (task.isSuccessful()){  
                    Toast.makeText(SignUp.this, "User registered successfully", Toast.LENGTH_SHORT).show();  
                    startActivity(new Intent(SignUp.this, Login.class));  
                }else{
```

```

        Toast.makeText(SignUp.this, "Registration Error: " +
task.getException().getMessage(), Toast.LENGTH_SHORT).show();
    }
}
});
}
}

```

По такому же принципу можно реализовать и авторизацию пользователя.

Функцию ввода пользователем информации по дневному и вечернему уходу, а также функцию определения типа кожи, посредством тестирования можно реализовать с помощью функции Realtime Database.

Realtime Database – является облачной базой данных. Данные хранятся как JSON и синхронизируются в реальном времени с каждым подключенным клиентом. При создании кроссплатформенных программных средств с iOS, Android и JavaScript SDK, все клиенты делятся одним экземпляром базы данных Realtime и автоматически получают обновления с новейшими данными [14].

В разрабатываемом программном средстве данная функция Realtime Database используется в первую очередь для того, чтобы всю информацию, картинки, вопросы и так далее не хранились в самом программном средстве, а подгружались с сервера удаленной базы данных, чтобы программное средство в Play Market не хранило большое количество данных, что позволит уменьшить размер файла-установщика. К тому же информация хранится очень структурированно, что не позволяет запутаться при разработке. На рисунке 2.11 представлена реализованная база данных программного средства.

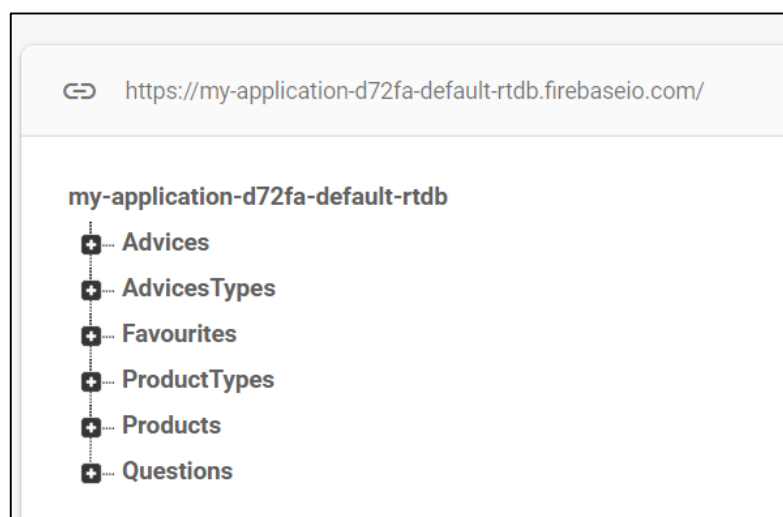


Рисунок 2.11 – Realtime Database разрабатываемого программного средства

Firebase Cloud Messaging (FCM) – это кроссплатформенное решение для обмена сообщениями, с помощью которого можно отправлять сообщения устройствам [15].

В разрабатываемом программном средстве была задача реализовать уведомления для контроля ежедневного ухода. На рисунке 2.12 показана реализация данной задачи.

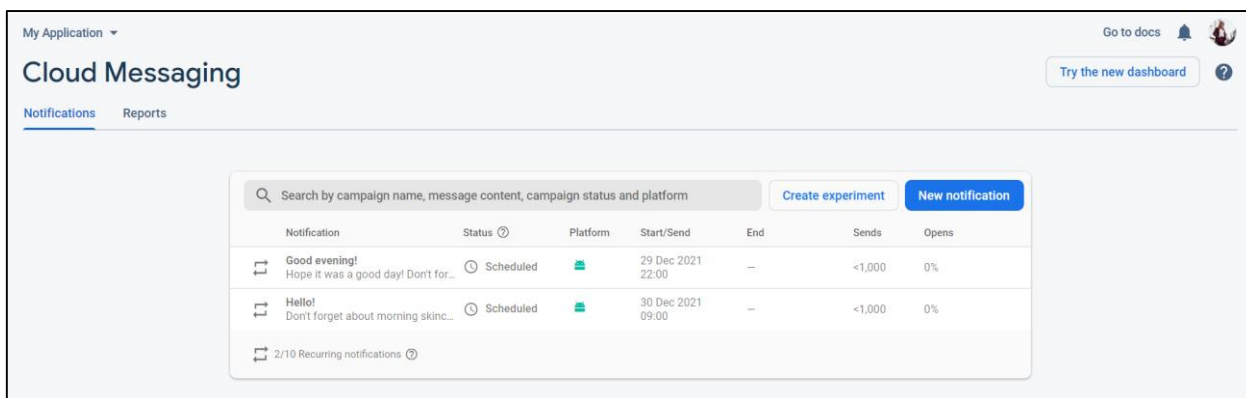


Рисунок 2.12 – Реализованные уведомления с помощью FCM

Таким образом, в данном подразделе были рассмотрены способы хранения информации различного типа.

## 2.4 Проектирование и разработка графического интерфейса

В настоящее время дизайн программных средств является очень важным фактором, так как пользователь в первую очередь оценивает именно интерфейс программного средства. Поэтому вес красивого и функционального дизайна очень существенен. Поэтому этап дизайна и визуального оформления нельзя пропускать при разработке программных средств.

Главная цель UI (user interface) – упростить взаимодействие со сложными техническими объектами. Любая программа выглядит как бесконечное количество 0 и 1: набирать их вручную, чтобы написать простое сообщение – долгий бесполезный процесс. Интерфейсы помогли ускорить эти действия, сделав их доступными для всех. Они транслируют информацию, отправляют команды, помогают обмениваться данными и выполняют другие полезные функции.

С развитием программных средств UI стали также выполнять задачу помощника. Графические элементы выступают индикаторами, направляя пользователей. Если убрать их, то перед вами окажется набор символов [16].

В сфере разработки дизайна существует 2 глобальных направления. Это UX-дизайн и UI-дизайн. Начнем с понятия попроще. UI-User Interface, в пере-

воде с английского – это пользовательский интерфейс. То есть это направление отвечает за визуальную составляющую программных средств. Данное направление очень важное по одной простой причине, психология человека такова, что человеку необходимо 7 секунд для того, чтобы визуально оценить сайт, программное средство или любое другое программное средство, если человеку не нравится, он просто выходит или удаляет программное средство. Поэтому одной из основных задач при разработке программного средства является разработка визуально привлекающего дизайна.

Однако, как мы помним, что кроме UI-дизайна есть еще и UX-дизайн. И данное направление не менее важное. UX-user interface, в переводе с английского – пользовательский опыт, это то как пользователь взаимодействует с интерфейсом. При разработке программного средства, необходимо учитывать также и прозрачность понимания использования программного средства. UX-дизайн отвечает за функции, адаптивность продукта и то, какие эмоции он вызывает у пользователей. Чем понятнее интерфейс, тем легче пользователю получить результат и совершить целевое действие. Это значит, что пользователь должен интуитивно понимать, где что находится и что для чего нужно.

Основными требованиями к качественному UX/UI-дизайну являются:

1 Ясность. В интерфейсе нет двусмысленности, а текст и структура направляют пользователя к цели.

2 Лаконичность. Интерфейс не перегружен подсказками, всплывающими окнами и анимацией. Задавайте себе вопросы: «А нужно ли это здесь? Для чего?». Это поможет сфокусировать внимание пользователя на конкретном элементе.

3 Узнаваемость. Элементы дизайна легко распознать, даже если пользователь видит ваш сайт впервые. Делайте интерфейс интуитивно понятным. Например, не красьте кнопку подтверждения в оранжевый, если на большинстве сайтов она зеленая.

4 Отзывчивость. Хороший интерфейс реагирует на действия пользователя мгновенно. Он должен понимать, что происходит на экране прямо сейчас: прошла ли оплата, получил ли менеджер заявку, отправилось ли сообщение.

5 Постоянство. Соблюдайте постоянство для всех разделов сайта и программного средства. Элементы интерфейса – меню и слайдеры – должны вести себя одинаково на любой странице.

6 Эстетика. Создавайте интерфейс визуально привлекательным, чтобы пользователю было приятно работать, ничто его не раздражало и не отвлекало от решения задач.

7 Эффективность. Помимо внешней привлекательности хороший интерфейс экономит время пользователя и доставляет его в нужную точку с минимальными усилиями.

8 Снисходительность. Даже при самом продуманном интерфейсе ни один пользователь не застрахован от ошибки. Продумайте заботливые сообщения на случай, если что-то пошло не так. Это поможет сохранить деньги, время и лояльность клиентов в случае сбоя.

Очень важно в процессе разработки программного средства не путать два направления дизайна, для того чтобы по итогу средство было и красивое, и практичное. Разница между UX и UI – в том, что UX-дизайнер моделирует, как пользователь взаимодействует с интерфейсом, какие шаги будет предпринимать, чтобы достичь цели. А UI-дизайнер прорабатывает все визуальные детали, все эти шаги и путь к цели [17].

В процессе разработки программного средства для дипломного проектирования все вышеперечисленные рекомендации и требования были учтены. И прежде чем работать с кодом, необходимо было разработать примерный дизайн будущего программного средства для определения объема работы и технологий, которые могут быть использованы в процессе написания кода.

Для того чтобы не рисовать от руки на листочке дизайн, практикующие дизайнеры и разработчики объединившись создали онлайн-сервис для разработки интерфейсов и прототипирования с возможностью организации совместной работы в режиме реального времени. Данный сервис называется Figma [18]. Так как сервис ориентирован не только на работу одного пользователя, но и для больших проектов, где задействовано большое количество людей, в сервисе могут работать дизайнеры, маркетологи, менеджеры и разработчики одновременно.

В Figma можно отрисовать элементы интерфейса, создать интерактивный прототип сайта и программного средства, иллюстрации, векторную графику. Элементы интерфейса – это внешний вид продукта. Также можно создать кнопки, иконки, формы обратной связи и настроить эффекты: сделать кликабельные кнопки, раскрыть списки, создать анимацию для блоков и многое другое.

Поэтому предварительное визуальное представление будущего программного средства было создано в Figma (рисунки 2.4 – 2.5).

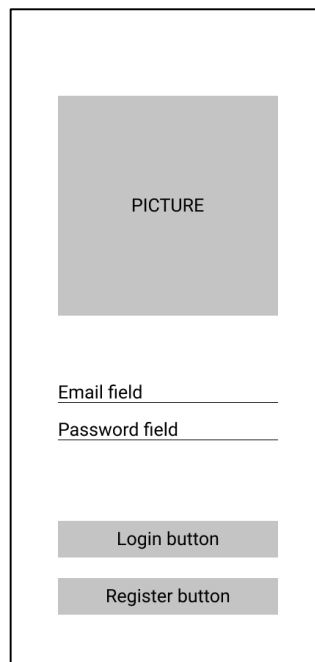


Рисунок 2.4 – Прототип экрана авторизации

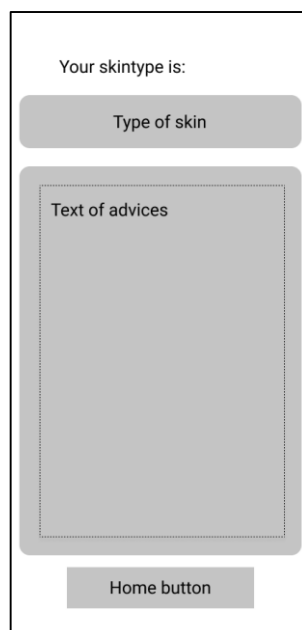


Рисунок 2.5 – Прототип экрана результат тестирования

Далее необходимо перейти в Android Studio для создания программного средства и его дизайна уже там. Все экраны в андроид-разработке представлены в файлах .xml. Например, экран аутентификации разрабатываемого программного средства представлен на рисунке 2.6.

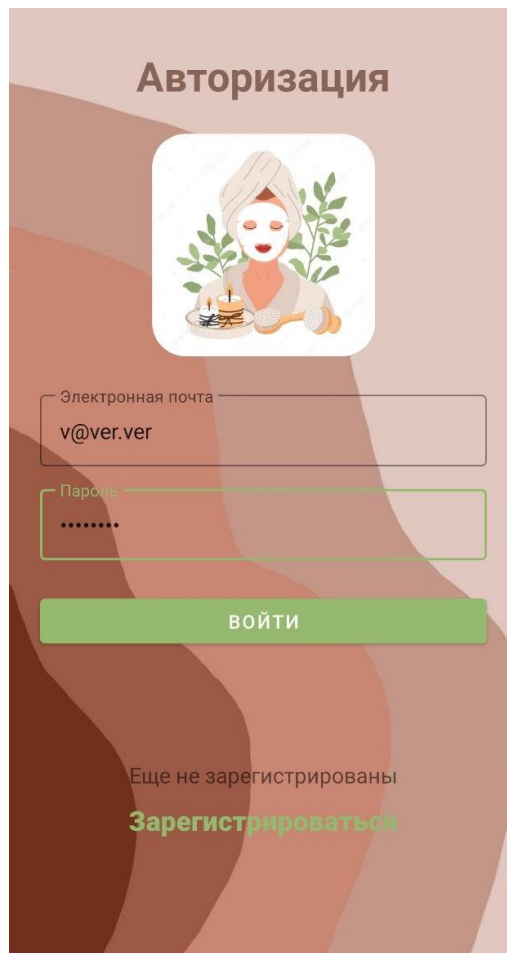


Рисунок 2.6 – Визуальное представление файла activity\_login.xml

Листинг файла activity\_login.xml:

```
<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    style="@style/Widget.MaterialComponents.TextInputLayout.Out-
linedBox"
    android:layout_height="wrap_content"
    app:boxStrokeColor="@color/green"
    app:hintTextColor="@color/black">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/etLoginEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="25dp"
        android:hint="Email"
        android:inputType="textEmailAddress"
        android:textColor="@color/black"
        android:textColorHighlight="@color/green"
        android:textColorLink="@color/green" />
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        style="@style/Widget.MaterialComponents.TextInputLayout.Out-
linedBox">
```

```

        android:layout_height="wrap_content"
        android:layout_marginTop="12dp"
        app:boxStrokeColor="@color/green"
        app:hintTextColor="@color/black">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/etLoginPass"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Password"
            android:inputType="textPassword"
            android:layout_margin="25dp"/>
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="25dp"
        android:text="Login"
        app:backgroundTint="@color/green" />

```

Для визуального представления еще также используют Material Design – стиль графического дизайна интерфейсов программного обеспечения и программных средств, разработанный компанией Google. Благодаря Material Design анимация сегодня – не просто эффектное дополнение дизайна, а полноценная его часть.

В основе Material Design лежат четыре принципа:

1 **Тактильные поверхности.** Все элементы интерфейса – это слои цифровой бумаги. Они располагаются на разной высоте и отбрасывают тени. Это помогает пользователям отличить главные элементы от второстепенных и делает интерфейс интуитивно понятным.

2 **Полиграфический дизайн.** Логично, что на цифровой бумаге нужно писать цифровыми чернилами. Все, что изображено и написано на слоях-элементах, подчиняется законам печатного дизайна. Так можно акцентировать внимание пользователя на нужном элементе и обозначить иерархию интерфейса.

3 **Осознанная анимация.** Все элементы, которые есть на экране, не могут просто так появляться и исчезать, – ведь в реальной жизни так не бывает. Объекты плавно переходят один в другой и подсказывают пользователю, как работает интерфейс.

4 **Адаптивный дизайн.** Все вышеперечисленное должно работать на любых устройствах [19].

С помощью такого стиля в разрабатываемом программном средстве реализована возможность просматривать продукты «уходовой» косметики (рисунк 2.6), а также советы по уходу за кожей лица.



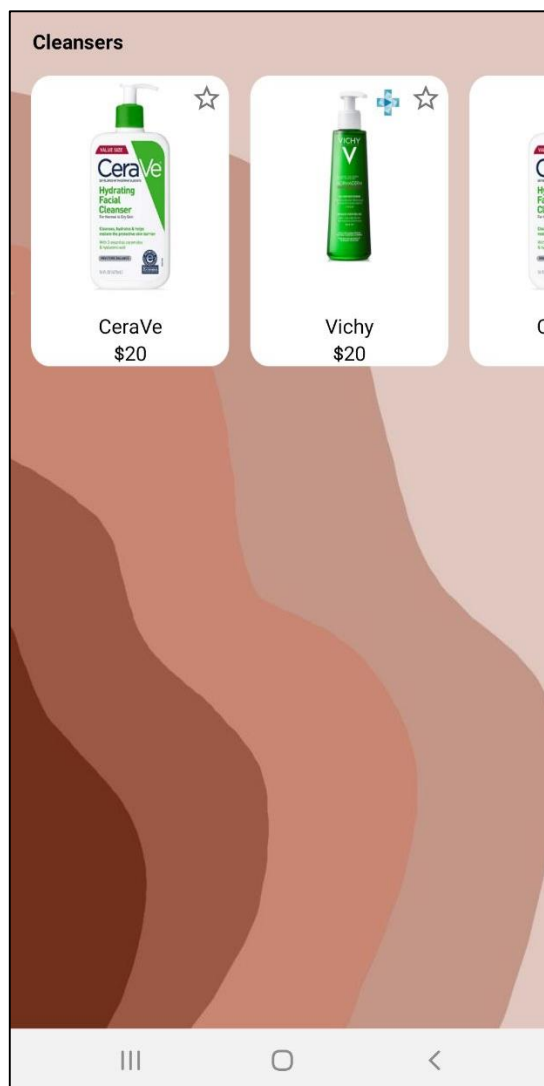


Рисунок 2.6 – Реализация просмотра продуктов «уходовой» косметики с помощью стиля Material Design

На практике в коде данная возможность реализуется с помощью такого компонента как RecyclerView.

Листинг компонента RecyclerView в activity\_products.xml:

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/cleansers_item"
    android:layout_width="409dp"
    android:layout_height="623dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView2" />
```

По такому же принципу разработки дизайна было реализовано и все программное средство.

## **2.5 Описание и реализация используемых в программном средстве алгоритмов**

Алгоритм – это любая корректно определенная вычислительная процедура, на вход (input) которой подается некоторая величина или набор величин и результатом выполнения которой является выходная (output) величина или набор значений. Таким образом, алгоритм представляет собой последовательность вычислительных шагов, преобразующих входные величины в выходные.

Алгоритм также можно рассматривать как инструмент, предназначенный для решения корректно поставленной вычислительной задачи (computational). В постановке задачи в общих чертах задаются отношения между входом и выходом. В алгоритме описывается конкретная вычислительная процедура, с помощью которой удастся добиться выполнения указанных отношений. При написании любого программного средства необходимо реализовать как минимум простые алгоритмы, чем сложнее программное средство, тем сложнее будут алгоритмы [20].

Интерфейс программы является «связью с внешним миром». Он ожидает в цикле прихода какой-либо команды, периодически проверяя ее наличие. Когда приходит команда, происходит ее разбор, а именно определяется тип и, соответственно, действие, которое нужно выполнить или определить какому модулю передать управление.

Перед запуском поиска оптимального сочетания критериев обязательно осуществляется проверка корректности модели. Данная проверка заключается в определении циклов в зависимостях компонентов одной подсистемы, в проверке правильности выражений. Одновременно вычлняются неиспользуемые компоненты и подсистемы, и их ветви «отрезаются» от общего дерева.

Специфика применяемых методов проектирования алгоритмов и используемых при этом инструментальных средств разработки программ может повлиять на форму представления и содержание алгоритма обработки данных. Алгоритм работы программного средства можно представить в виде схемы алгоритма.

Схема алгоритма – это схематичное представление процесса, системы или компьютерного алгоритма. Блок-схемы часто применяются в разных сферах деятельности, чтобы документировать, изучать, планировать, совершенствовать и объяснять сложные процессы с помощью простых логических диаграмм. Для построения блок-схем применяются прямоугольники, овалы, ромбы и некоторые другие фигуры (для обозначения конкретных операций), а также соединительные стрелки, которые указывают последовательность шагов или направление процесса. Блок-схемы варьируются от незамысловатых, нарисованных вручную до подробных, составленных на компьютере диаграмм со множеством шагов и процессов. Если учесть все возможные вариации,

блок-схемы можно признать одним из самых распространенных видов схем во всем мире. Они широко используются в разных сферах как технической, так и нетехнической направленности. Данные блок-схемы создаются с помощью специальных обозначений.

Схемы алгоритмов и программ отображают основные операции процесса обработки данных [21]. Схемы включают:

- условные графические обозначения (символы), имеющие заданные значения;
- краткий пояснительный текст внутри символов, разъясняющий функцию данных символов, и более подробный текст в символе комментария;
- соединяющие символы линии, отображающие поток данных.

В процессе разработки программного средства был разработан алгоритм его работы (рисунок 2.3). На данном рисунке можно увидеть, как последовательно работает программное средство, что позволяет лучше ориентироваться в его функциональности.

В начале работы программного средства пользователь попадает на экран авторизации и, если пользователь не зарегистрирован ему следует сначала пройти регистрацию на соответствующем экране. После регистрации пользователь возвращается на экран авторизации, где вводит все данные, уже сохраненные в базе данных и входит в программное средство. Далее после прохождения авторизации пользователь попадает на домашнюю страницу программного средства, где уже может отметить выполненные шаги ухода, для контроля. Также у пользователя есть выбор навигационного меню: боковое и нижнее. Если пользователь выбирает боковое меню, у него есть возможность выбрать прохождение тестирования на определение типа кожи, просмотреть список продуктов, добавленных им в избранное, просмотреть статистику, какие средства наиболее популярны среди пользователей. Если пользователь выбирает прохождение тестирования, он переходит на экран с тестом. После ответов на все вопросы, в данном тестировании, пользователь получает персональные рекомендации, после которых может вернуться на главный экран программного средства с помощью кнопки «Вернуться домой». Если пользователь, в боковом меню, выбирает просмотр избранного или просмотр статистики, то переходит на соответствующий экран. Также в нижнем меню у пользователя есть 3 опции, первая – это возможность ежедневного и еженедельного контроля ухода с чек-боксов, которые разделены на «День», «Ночь», «Недельный уход». Вторая возможность – находить и читать советы по уходу, на экране «Советы». Третья возможность – находить и просматривать составы продуктов по уходу, которые также можно добавить в избранное, с помощью экрана «Продукты».

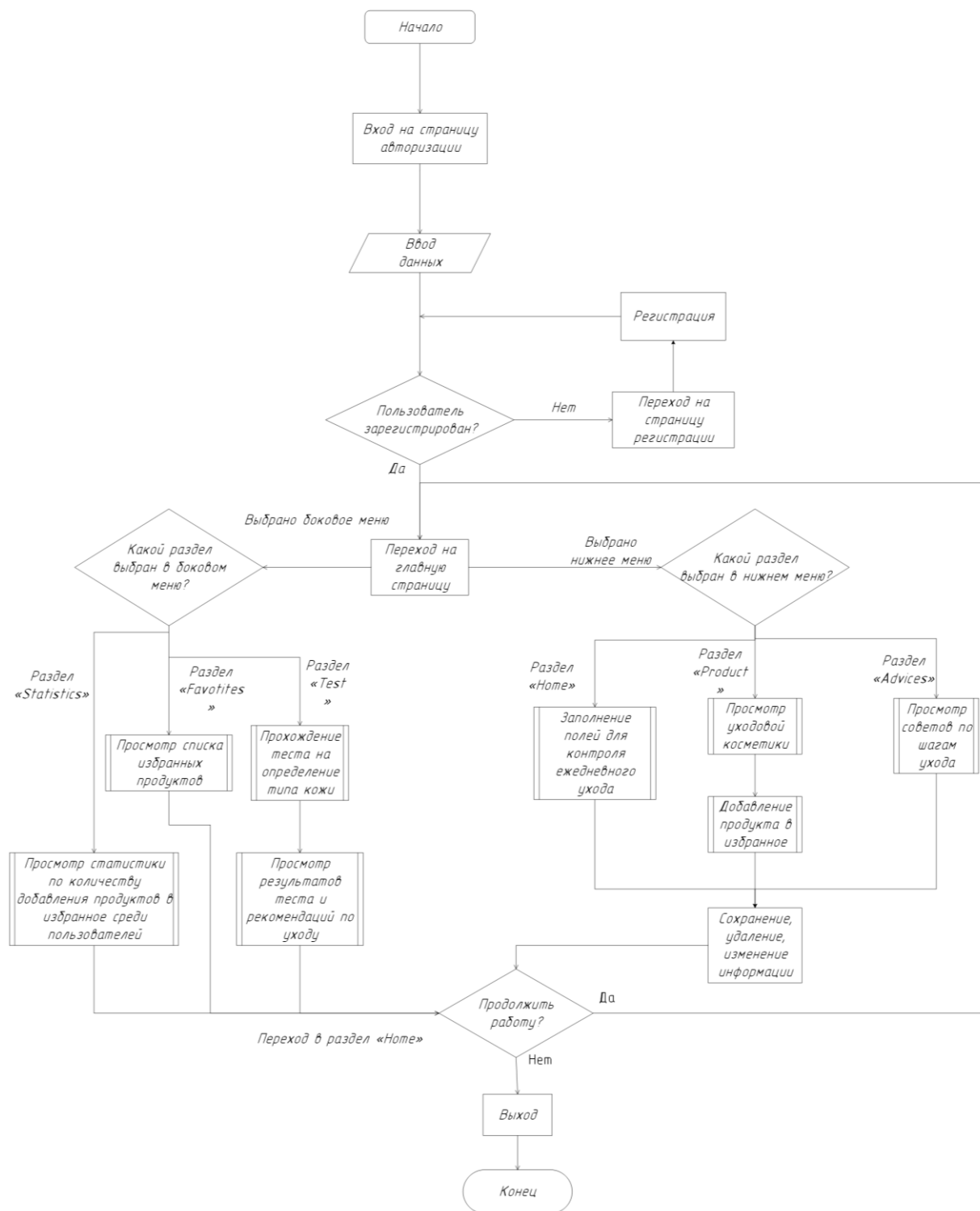


Рисунок 2.3 – Алгоритм работы программного средства

Таким образом, используя составленный алгоритм работы разрабатываемого программного средства можно перейти к оценке показателей его функционирования.

### 3 ОЦЕНКА КОЛИЧЕСТВЕННЫХ ПОКАЗАТЕЛЕЙ ФУНКЦИОНИРОВАНИЯ ПРОГРАММНОГО СРЕДСТВА

#### 3.1 Оценка временных показателей программного средства

Для оценки временных показателей было использовано расширение Firebase – Google Analytics. Google Analytics – это бесплатное решение для измерения программных средств, которое дает представление об использовании программного средства и взаимодействии с пользователем. В основе Firebase лежит Google Analytics, бесплатное и неограниченное аналитическое решение. Аналитика объединяет функции Firebase и предоставляет неограниченные отчеты по 500 отдельным событиям, которые можно определить с помощью Firebase SDK. Отчеты аналитики помогают четко понимать, как ведут себя пользователи, что позволяет принимать обоснованные решения в отношении маркетинга программных средств и оптимизации производительности.

Google Analytics помогает понять, как люди используют Интернет и разрабатываемое программное средство Android. SDK (software development kit – «комплект для разработки программного обеспечения») автоматически фиксирует ряд событий и свойств пользователей, а также позволяет определять свои собственные события для измерения вещей, которые имеют уникальное значение для разработки. После сбора данных они доступны на панели управления через консоль Firebase. Эта панель управления предоставляет подробную информацию о ваших данных – от сводных данных, таких как активные пользователи и демографические данные, до более подробных данных, таких как идентификация наиболее востребованных товаров (рисунок 3.1).

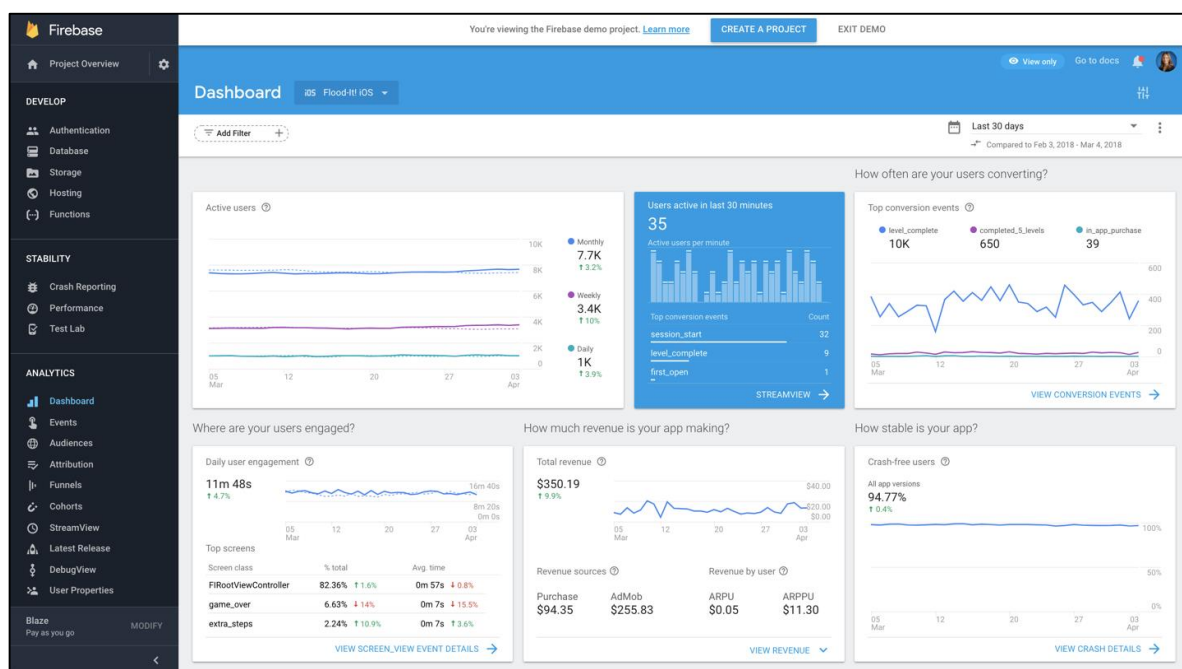


Рисунок 3.1 – Пример Firebase Analytics Dashboard

Аналитика также интегрируется с рядом других функций Firebase. Например, он автоматически регистрирует события, которые соответствуют сообщениям уведомлений, отправленным через композитор уведомлений, и предоставляет отчеты о влиянии каждой кампании.

Аналитика помогает понять, как ведут себя пользователи, для принятия обоснованного решения о том, как продвигать свое программное средство на рынке [23].

Основным временным показателем при разработке программного средства, является время запуска программного средства. Пользователи ожидают, что программные средства будут быстро реагировать и загружаться. Программное средство с медленным запуском не соответствует этим ожиданиям и может разочаровать пользователей. Подобный плохой опыт может привести к тому, что пользователь плохо оценит программное средство в Play Market или даже откажется от него вообще.

Запуск программного средства может происходить в одном из трех состояний, каждое из которых влияет на то, сколько времени требуется, чтобы программное средство стало видимым для пользователя: холодный запуск, теплый запуск или горячий запуск. При холодном запуске программное средство запускается с нуля. В других состояниях системе необходимо вывести работающее программное средство из фона на передний план. Рекомендуется всегда выполнять оптимизацию в состоянии холодного запуска. Это также может улучшить характеристики горячего и горячего запуска [24].

После подключения Firebase Google Analytics к программному средству можно с легкостью установить время запуска программного средства, на всех устройствах из разных состояний. На рисунке 3.2 можно увидеть показатели времени запуска программного средства.

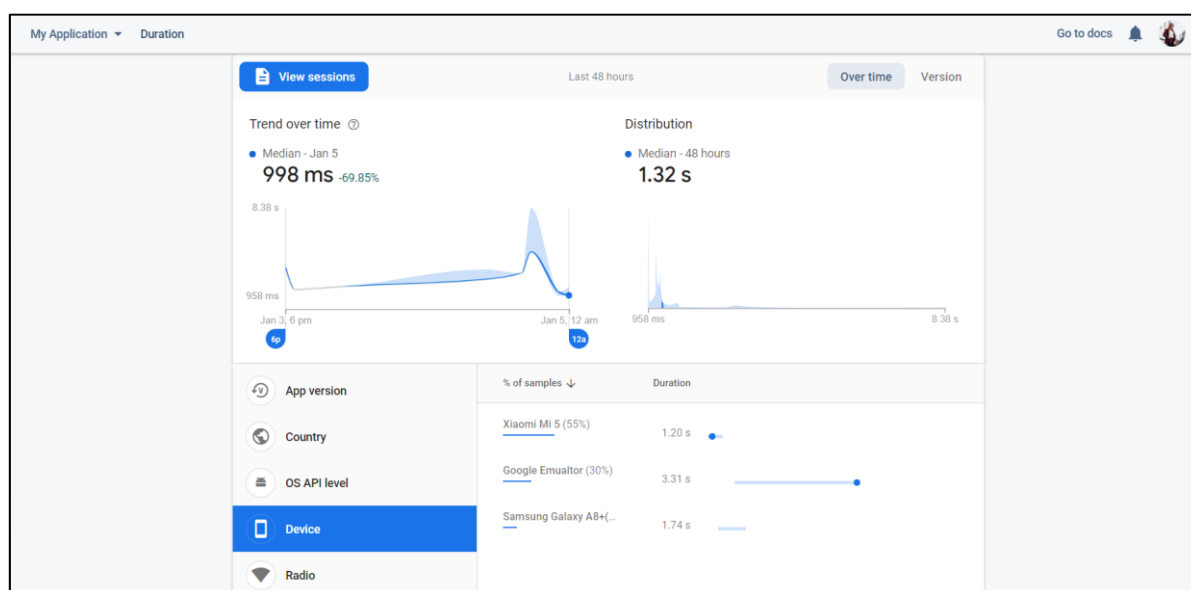


Рисунок 3.2 – Показатели времени запуска

На рисунке 3.2 можно увидеть, что аналитика основывается на разных показателях. На то из какой страны запущено программное средство, какой девайс использует пользователь. Все факторы влияют на время запуска.

Для удобства все значения из данных графиков занесены в таблицу 3.1.

Таблица 3.1 – Время запуска программного средства на разных устройствах

| Устройство               | Время запуска в разных состояниях, сек |
|--------------------------|--|
| Xiaomi Mi 5              | 2,56                                   |
|                          | 1,74                                   |
|                          | 1,49                                   |
| Google Emulator Pixel XL | 8,38                                   |
|                          | 3,76                                   |
|                          | 3,18                                   |
| Samsung Galaxy A8+       | 1,89                                   |
|                          | 1,32                                   |
|                          | 0,99                                   |

Исходя из данных приведенных в таблице 3.1 можно посчитать среднее время запуска по формуле 3.1:

$$B_{cp} = \frac{\sum B}{N}, \quad (3.1)$$

где  $B_{cp}$  – среднее время, затраченное на запуск программного средства, с;  $N$  – количество временных отрезков;  $B$  – показатели времени, с;

Исходя из данных в таблице 3.1, рассчитаем среднее время, затраченное на запуск программного средства с помощью формулы 3.1:

$$B_{cp} = (2,56 + 1,74 + 1,49 + 8,38 + 3,76 + 3,18 + 1,89 + 1,32 + 0,99)/9 = 2,81 \text{ с}$$

Таким образом среднее время, затраченное на запуск программного средства, составляет 2,81 секунд.

### 3.2 Оценка ресурсных показателей программного средства

Для оценки ресурсных показателей используется встроенный в Android Studio профилировщик Profiler. Android Profiler в Android Studio 3.0 и более поздних версиях заменяет инструменты Android Monitor. Инструменты Android Profiler предоставляют данные в реальном времени, чтобы помочь понять, как программное средство использует ресурсы ЦП, памяти, сети и аккумулятора [25].

Используя встроенный в Android Studio профилировщик CPU Profiler, получим данные о нагрузке на процессор в момент запуска программного средства, выполнения трудоемких процессов и во время простоя. На рисунках 3.3 – 3.5 показано, как отображаются данные.

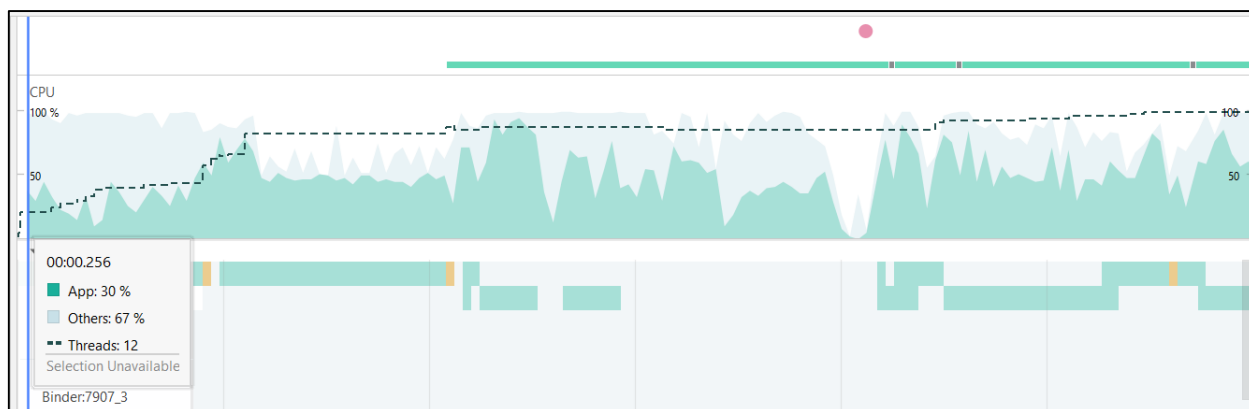


Рисунок 3.3 – Нагрузка на процессор в момент запуска программного средства

Как видно из рисунка 3.6, максимальное значение использования процессора в момент запуска программного средства составляет 30%.

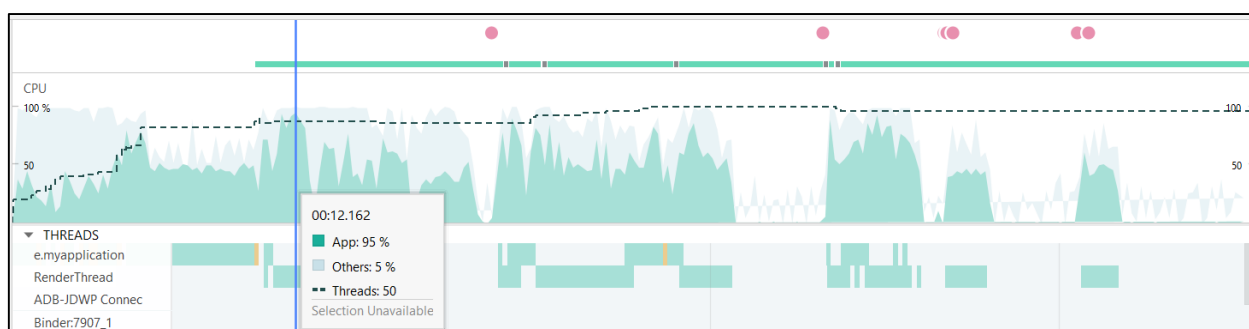


Рисунок 3.4 – Нагрузка на процессор в момент выполнения трудоемких процессов

Как видно из рисунка 3.4, максимальное значение использования процессора во время выполнения трудоемких процессов составляет 95%.



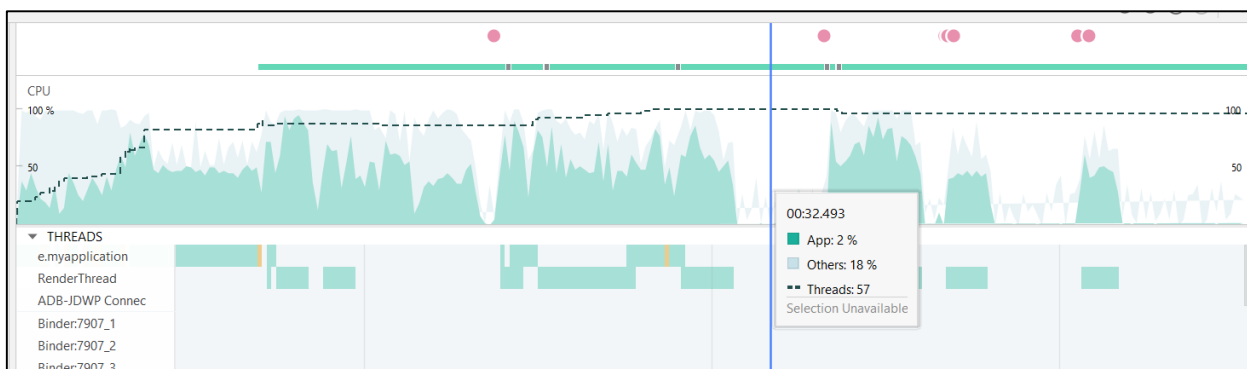


Рисунок 3.5 – Нагрузка на процессор во время простоя

Как видно из рисунка 3.5, максимальное значение использования процессора во время простоя составляет 2%.

Расчёт среднего значения нагрузки на процессор создаваемой программным средством осуществляется по формуле 3.2:

$$НП_{ср} (НП_{зап} + НП_{труд} + НП_{пр}) / 3, \quad (3.2)$$

где  $НП_{ср}$  – средняя нагрузка на процессор, %;  $НП_{зап}$  – нагрузка на процессор во время запуска программного средства, %;  $НП_{труд}$  – нагрузка на процессор во время выполнения трудоемких процессов, %;  $НП_{пр}$  – нагрузка на процессор во время простоя.

Исходя из полученных ранее результатов, рассчитаем среднее значение нагрузки на процессор, создаваемой программным средством, с помощью формулы 3.2:

$$НП_{ср} = (30 + 95 + 2) / 3 = 42,3\%$$

Таким образом, средняя нагрузка на процессор, создаваемая программным средством, составляет 42,3 %.

Профилировщик памяти – это компонент профилировщика Android, который помогает выявлять утечки памяти и отток памяти, которые могут привести к прерыванию работы, зависаниям и даже сбоям программных средств. Он показывает график использования памяти вашего программного средства в реальном времени для отслеживания распределение памяти [26].

На рисунках 3.6 представлено количество используемой памяти на старте запуска программного средства.

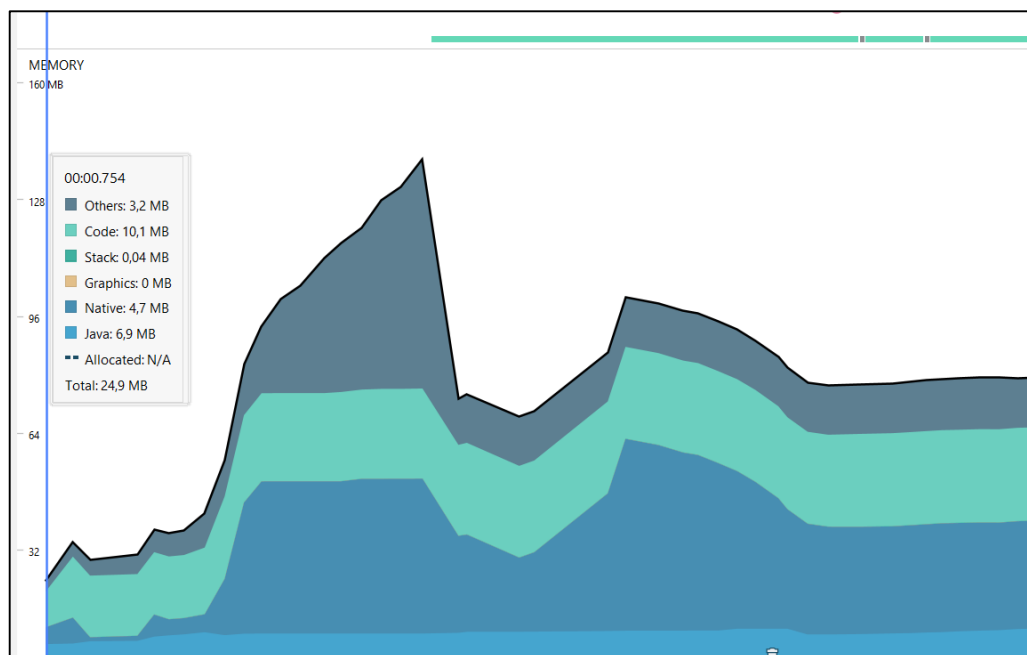


Рисунок 3.6 – Распределение памяти во время запуска программного средства

Как можно видеть из рисунка 3.6, потребление оперативной памяти на старте запуска программного средства составляет 24,9 МБ. Также, из данного рисунка видно, что наибольшее количество оперативной памяти отведено под код.

На рисунке 3.7 представлено максимальное количество потребляемой памяти во время использования программного средства.

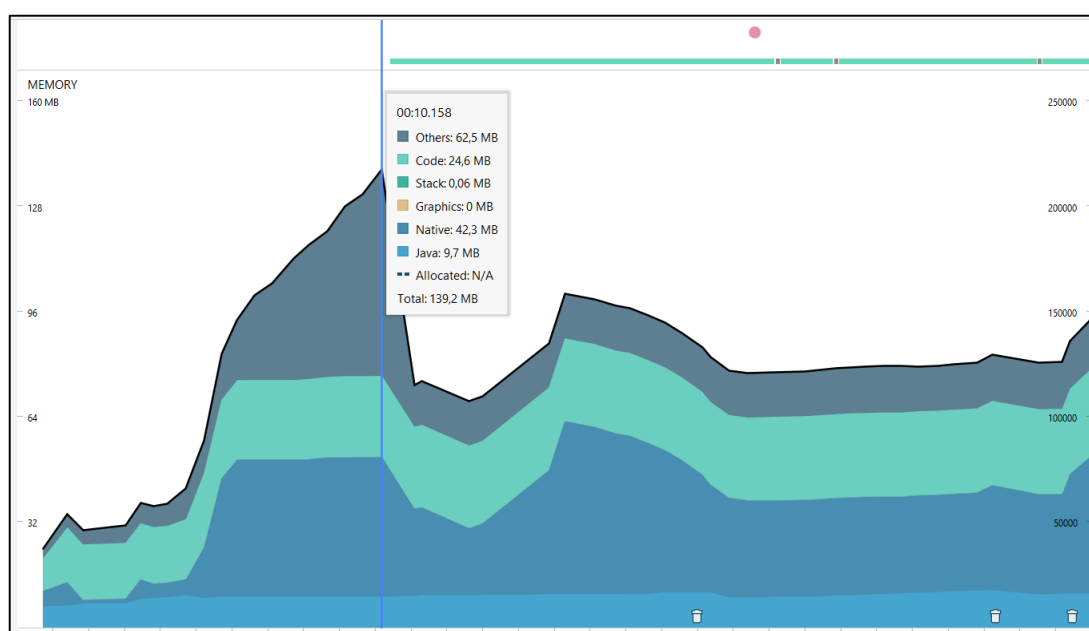


Рисунок 3.7 – Максимальное потребление оперативной памяти во время использования программного средства

Максимальное значение потребления оперативной памяти было зафиксировано в момент запуска и отображения программного средства на устройстве. Исходя из представленной на рисунке 3.7 информации, максимальное потребление оперативной памяти во время использования программного средства составило 139,2 МБ.

Количество используемой программным средством оперативной памяти во время простоя представлено на рисунке 3.8.

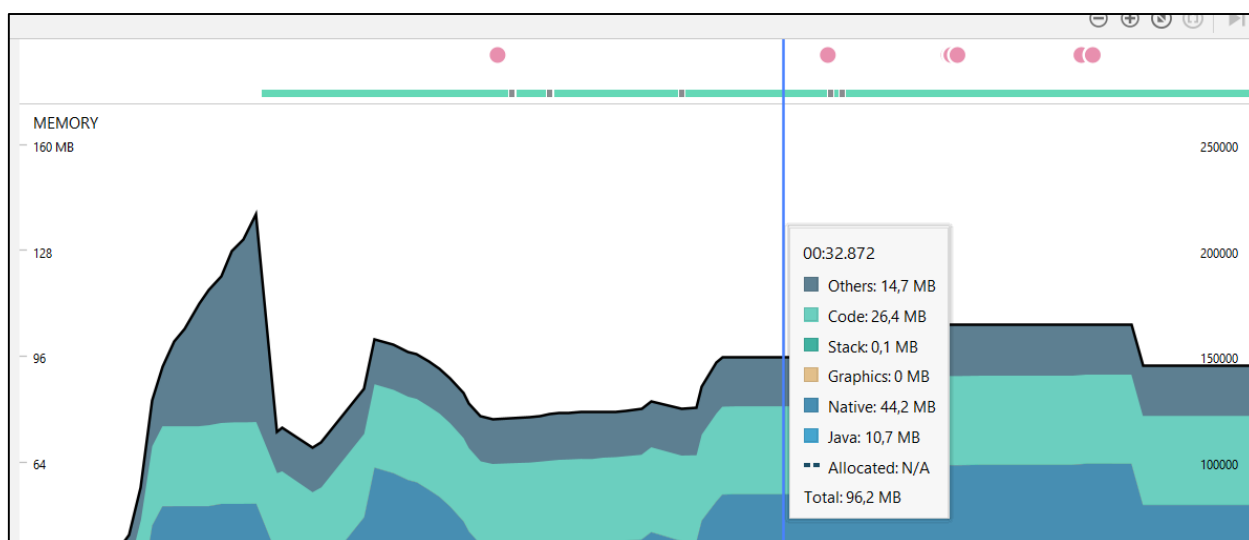


Рисунок 3.8 – Распределение памяти во время простоя программного средства

Исходя из значений, представленных на рисунке 3.8, общее потребление оперативной памяти во время простоя программного средства составило 96,2 МБ.

Суммарная информация о потреблении оперативной памяти программным средством представлена в таблице 3.2

Таблица 3.2 – Потребление оперативной памяти программным средством

| Название категории счетчика | Количество потребляемой оперативной памяти, МБ |               |                  |
|-----------------------------|--|---------------|------------------|
|                             | На страте                                      | Под нагрузкой | Во время простоя |
| Code                        | 10,1   | 24,6          | 26,4             |
| Java                        | 6,9  | 9,7           | 10,7             |
| Native                      | 4,7  | 42,3          | 44,2             |
| Graphics                    | 0  | 0             | 0                |
| Stack                       | 0,04   | 0,06          | 0,1              |
| Others                      | 3,2  | 62,5          | 14,7             |
| Total                       | 24,9   | 139,2         | 96,2             |

Категории в счетчике памяти обозначают следующее:

- java: память из объектов, выделенных из кода Java или Kotlin;

- native: память из объектов, выделенных из кода C или C++;
- graphics: память, используемая для очередей графического буфера для отображения пикселей на экране, включая поверхности GL, текстуры GL и так далее;
- stack: память, используемая в программном средстве как нативным кодом так и кодом Java, предназначенная для хранения вызовов методов и локальных переменных;
- code: память, которую ваше программное средство использует для кода и ресурсов, таких как dex байт-код, оптимизированный или скомпилированный код dex, .so библиотеки и шрифты;
- others: память, используемая вашим программным средством, которую система не знает, как классифицировать;
- total: суммарное количество оперативной памяти, используемое программным средством [27].

Расчёт среднего значения потребления оперативной памяти программным средством осуществляется по формуле 3.1:

$$P_{\text{ср}} (P_{\text{ст}} + P_{\text{н}} + P_{\text{пр}}) / 3, \quad (3.3)$$

где  $P_{\text{ср}}$  – среднее потребление оперативной памяти, МБ;  $P_{\text{ст}}$  – потребление оперативной памяти на старте, МБ;  $P_{\text{н}}$  – потребление оперативной памяти под нагрузкой, МБ;  $P_{\text{пр}}$  – потребление оперативной памяти во время простоя, МБ. Исходя из полученных ранее результатов, рассчитаем среднее значение потребления оперативной памяти программным средством используя формулу 3.1:

$$P_{\text{ср}} (P_{\text{ст}} + P_{\text{н}} + P_{\text{пр}}) / 3 = 24,9 + 139,2 + 96,2 = 86,8 \text{ МБ}$$

Таким образом, среднее потребление оперативной памяти программным средством составляет 86,8 МБ. Следует отметить, что полученные данные справедливы для отладочной версии программного средства и могут отличаться от выпускаемой на рынок.

Network Profiler отображает сетевую активность в реальном времени на временной шкале, показывая отправленные и полученные данные, а также текущее количество подключений. Это позволяет вам изучить, как и когда программное средство передает данные, и соответствующим образом оптимизировать базовый код [28].

На рисунке 3.9 показано количество полученных и отправленных данных в момент запуска программного средства.



Рисунок 3.9 – Количество полученных и отправленных данных в момент запуска программного средства

При переходе на фрагменты программного средства, где необходимо подгружать данные с сервера, количество полученных и отправленных данных справедливо растет. На рисунке 3.10 показано количество полученных и отправленных данных в момент перехода на фрагменты.

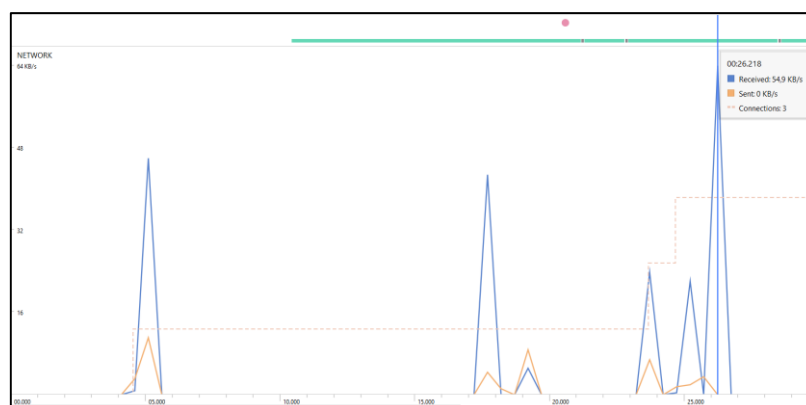


Рисунок 3.10 – Количество полученных и отправленных данных в момент перехода на фрагменты

Для удобства расчетов среднего количества полученных и отправленных данных во время работы программного средства, все данные были помещены в таблицу 3.3

Таблица 3.3 – Количество полученного и отправленного трафика

| Состояние                               | Трафик, Кб/с |            |
|---|--------------|------------|
|   | Получено     | Отправлено |
| На старте запуска программного средства | 39,5         | 9,5        |
| После запуска программного средства     | 36,6         | 7          |

Продолжение таблицы 3.3

| Состояние                           | Трафик, Кб/с |            |
|-------------------------------------|--------------|------------|
|                                     | Получено     | Отправлено |
| При переходе на фрагмент «Advices»  | 54,9         | 5,6        |
| При переходе на фрагмент «Products» | 26           | 3,6        |
| При переходе на фрагмент «Test»     | 35,2         | 4,3        |

Исходя из данных в таблице 3.3, можно найти средние значения полученного трафика по формуле:

$$ТП_{cp} = (ТП_{сз} + ТП_{пз} + ТП_{ad} + ТП_{pr} + ТП_t) / 5, \quad (3.3)$$

где  $ТП_{cp}$  – среднее количество полученных данных, Кб/с;  $ТП_{сз}$  – количество полученных данных на старте запуска программного средства, Кб/с;  $ТП_{пз}$  – количество полученных данных после запуска программного средства, Кб/с;  $ТП_{ad}$  – количество полученных данных при переходе на фрагмент «Advices», Кб/с;  $ТП_{pr}$  – количество полученных данных при переходе на фрагмент «Products», Кб/с;  $ТП_t$  – количество полученных данных при переходе на фрагмент «Test».

$$ТП_{cp} = (39,5 + 36,6 + 54,9 + 26 + 35,2) / 5 = 38,44 \text{ Кб/с}$$

По такому же принципу можно найти среднее значение отправленного трафика:

$$ОП_{cp} = (ОП_{сз} + ОП_{пз} + ОП_{ad} + ОП_{pr} + ОП_t) / 5, \quad (3.4)$$

где  $ОП_{cp}$  – среднее количество полученных данных, Кб/с;  $ОП_{сз}$  – количество полученных данных на старте запуска программного средства, Кб/с;  $ОП_{ad}$  – количество полученных данных при переходе на фрагмент «Advices», Кб/с;  $ОП_{pr}$  – количество полученных данных при переходе на фрагмент «Products», Кб/с;  $ОП_t$  – количество полученных данных при переходе на фрагмент «Test».

$$ОП_{cp} = (9,5 + 7 + 5,6 + 3,6 + 4,3) / 5 = 6, \text{ Кб/с}$$

Таким образом можно увидеть, что при первом запуске программного средства в среднем полученные данные составляют 38,44 Кб/с, а отправленные 6 Кб/с.

### 3.3 Оценка показателей надежности программного средства

Показатель надежности – способность ПС поддерживать заданный уровень качества функционирования при его использовании в заданных условиях. Ограничения надежности в процессе эксплуатации вызваны ошибками в требованиях, проектировании и разработке [29].

В основном ошибки происходят в двух случаях. Первый это несоответствие настроек устройства с требованиями программного средства и второй

это изменения в коде, которые глобально влияют на работу программного средства на всех устройствах. К первому случаю можно отнести, например, отсутствие интернет-подключения на устройстве, когда программному средству необходимо загружать информацию с сервера. Когда такое происходит, разрабатываемое программное средство оповещает пользователя о том, что устройство находится в режиме оффлайн и соответственно информация с сервера не отображается (рисунок 3.11).

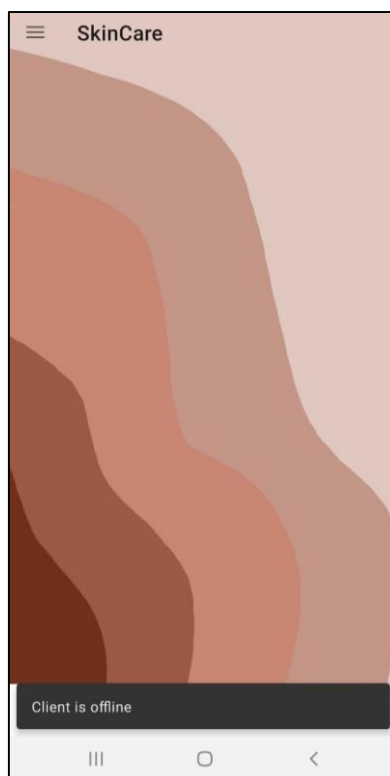


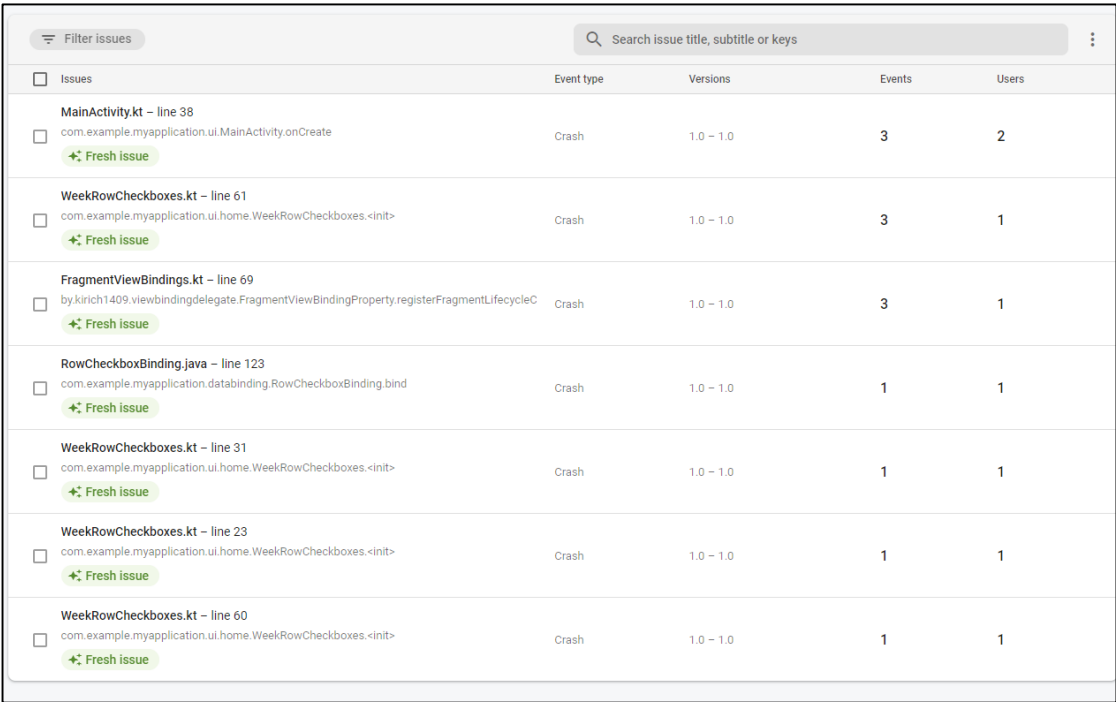
Рисунок 3.11 – Оповещение об ошибке подключения к сети Интернет

Ко второму типу ошибок относятся такие ситуации, когда программное средство модернизируется или, добавляются новые функции. Благодаря Firebase Crashlytics можно оперативно получать информацию об ошибках и также оперативно их устранять.

Crashlytics – бесплатный инструмент отслеживания и анализа багов в мобильных программных средствах на операционных системах iOS и Android. Сервис рассчитан на неограниченное количество программ, пользователей и ошибок. Отчёты о сбоях можно получать в режиме реального времени. Платформа предназначена для разработчиков программных средств, так как позволяет отслеживать баги и жалобы пользователей на падение программы и анализировать, что именно привело к сбою. Crashlytics собирает и отображает данные о багах для каждого в отдельности проекта с указанием номера строки с ошибкой. Полученный массив можно фильтровать по версиям программных средств, типам устройств, операционным системам, наличию джейлбрейку на

телефоне. Платформа высчитывает значения и строит графики с показателями среднего количества багов в день, приблизительного процента смартфонов с авариями, общего количества пользователей, столкнувшихся с проблемой. В режиме реального времени доступна статистика о числе устройств с запущенной программой и корректности работы. В перечне сбоев можно пометить о решении проблемы [30].

На рисунке 3.12 можно увидеть, как выглядит список ошибок и путь к этим ошибкам.



| <input type="checkbox"/> Filter issues  | Search issue title, subtitle or keys |            |           |        |       |
|---|--------------------------------------|------------|-----------|--------|-------|
| <input type="checkbox"/> Issues   |                                      | Event type | Versions  | Events | Users |
| <input type="checkbox"/> MainActivity.kt – line 38<br>com.example.myapplication.ui.MainActivity.onCreate<br><a href="#">Fresh issue</a>   |                                      | Crash      | 1.0 – 1.0 | 3      | 2     |
| <input type="checkbox"/> WeekRowCheckboxes.kt – line 61<br>com.example.myapplication.ui.home.WeekRowCheckboxes.<init><br><a href="#">Fresh issue</a>                                  |                                      | Crash      | 1.0 – 1.0 | 3      | 1     |
| <input type="checkbox"/> FragmentViewBindings.kt – line 69<br>by kirich1409.viewbindingdelegate.FragmentViewBindingProperty.registerFragmentLifecycleC<br><a href="#">Fresh issue</a> |                                      | Crash      | 1.0 – 1.0 | 3      | 1     |
| <input type="checkbox"/> RowCheckboxBinding.java – line 123<br>com.example.myapplication.databinding.RowCheckboxBinding.bind<br><a href="#">Fresh issue</a>                           |                                      | Crash      | 1.0 – 1.0 | 1      | 1     |
| <input type="checkbox"/> WeekRowCheckboxes.kt – line 31<br>com.example.myapplication.ui.home.WeekRowCheckboxes.<init><br><a href="#">Fresh issue</a>                                  |                                      | Crash      | 1.0 – 1.0 | 1      | 1     |
| <input type="checkbox"/> WeekRowCheckboxes.kt – line 23<br>com.example.myapplication.ui.home.WeekRowCheckboxes.<init><br><a href="#">Fresh issue</a>                                  |                                      | Crash      | 1.0 – 1.0 | 1      | 1     |
| <input type="checkbox"/> WeekRowCheckboxes.kt – line 60<br>com.example.myapplication.ui.home.WeekRowCheckboxes.<init><br><a href="#">Fresh issue</a>                                  |                                      | Crash      | 1.0 – 1.0 | 1      | 1     |

Рисунок 3.12 – Список ошибок в Firebase Crashlytics

Исходя из данных на рисунке 3.13 программное средство выходило из строя 13 раз.

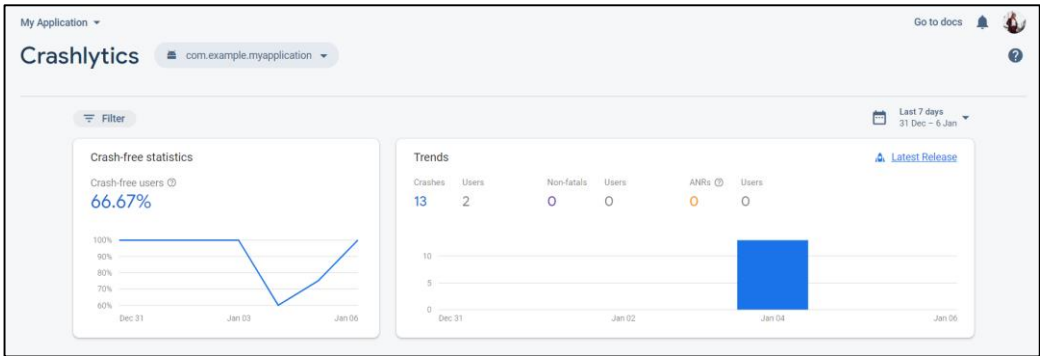


Рисунок 3.13 – Статистика в Firebase Crashlytics



Согласно функции Firebase Analytics программное средство было запущено 99 раз (рисунок 3.14).

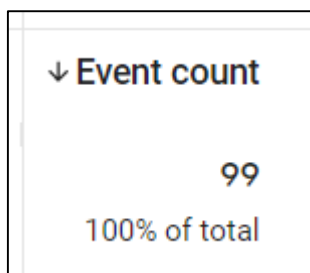


Рисунок 3.14 – Количество запусков программного средства

Соответственно исходя из приведенных данных можно найти процент неуспешных сценариев использования программного средства по формуле:

$$\%C_{\text{ну}} = \frac{C_{\text{ну}}}{C_y} \cdot 100, \% \quad (3.5)$$

где  $C_{\text{ну}}$  – количество неуспешных сценариев,  $C_y$  – количество успешных сценариев.

Итого процент неуспешных сценариев составляет:

$$\frac{13}{99} \cdot 100 = 13,13 \%$$

Соответственно по такому же принципу найти количество успешных сценариев можно найти по формуле:

$$100 - C_{\text{ну}}, \% \quad (3.6)$$

где  $\%C_{\text{ну}}$  – процент неуспешных сценариев.

Итого процент успешных сценариев составит:

$$100 - 13,13\% = 86,87 \%$$

Итого в период разработки программного средства процент неуспешных сценариев составлял 13,13%, а успешных 86,87%.

## **4 ЭКСПЛУАТАЦИЯ ПРОГРАММНОГО СРЕДСТВА**

### **4.1 Ввод в эксплуатацию и обоснование минимальных технических требований к оборудованию**

Для ввода программного средства в эксплуатацию необходимо его опубликовать в магазине программных средств. Так как разрабатываемое программное средство было разработано для платформы Android, следовательно, опубликовать данное программное средство необходимо в Google Play Store – магазин программных средств, а также игр, книг, музыки и фильмов от компании Google, позволяющий сторонним компаниям предлагать владельцам устройств с операционной системой Android устанавливать и приобретать различные программные средства. Публикация – это обычный процесс, который делает программные средства под Android доступными для пользователей. При публикации программных средств для Android выполняется 2 основные задачи:

1 Подготовка программного средства к выпуску. На этапе подготовки создается выпускная версия программного средства, которую пользователи могут загрузить и установить на свои Android-устройства.

2 Выпуск программного средства для пользователей. На данном этапе публикуется, продается и распространяется конечная версия программного средства среди пользователей.

С августа 2021 года публиковать новые программные средства в Google Play можно будет только в виде Android App Bundle (AAB).

Android App Bundle – это новый официальный формат публикации Android, который предлагает более эффективный способ создания и выпуска вашего программного средства.

Android App Bundle включает в себя весь скомпилированный код и ресурсы вашего программного средства, а также перекладывает создание APK и подписки на Google Play. Конечные APK-файлы под конкретные устройства и архитектуры процессоров в этом случае магазин собирает сам. В случае необходимости их можно будет получить потом из консоли разработчика. Пакет Android App Bundle позволяет упростить работу по сборке программного средства меньшего размера, что может повысить успешность установки и сократить количество удалений. Пакет имеет расширение файла ".aab".

Google Play использует набор программных средств для создания и обслуживания оптимизированных APK-файлов для каждой конфигурации устройства, поэтому для запуска вашего программного средства загружаются только код и ресурсы, необходимые для конкретного устройства. Больше не нужно создавать, подписывать и управлять несколькими APK-файлами, чтобы оптимизировать поддержку различных устройств, а пользователи получают более мелкие и оптимизированные загрузки.

Большинство проектов программных средств не требуют особых усилий для создания наборов программных средств, поддерживающих обслуживание оптимизированных APK-файлов. Например, если ранее уже были организованы код и ресурсы программного средства в соответствии с установленными соглашениями, необходимо просто создать подписанные пакеты Android App Bundle с помощью Android Studio или командной строки и загрузить их в Google Play. Оптимизированное обслуживание APK становится автоматическим преимуществом.

Когда используется формат пакета программных средств для публикации программного средства, также можно воспользоваться функцией Play Feature Delivery, которая позволяет добавлять функциональные модули в проект вашего программного средства. Эти модули содержат функции и ресурсы, которые включены в программное средство только в соответствии с указанными условиями или доступными позже во время выполнения, для загрузки с помощью основной библиотеки Play.

Публикация с помощью наборов Android App Bundle помогает пользователям устанавливать программное средство с минимально возможным количеством загрузок и увеличивает предельный размер сжатой загрузки до 150 МБ. То есть, когда пользователь загружает программное средство, общий размер сжатых APK-файлов, необходимых для установки программного средства (например, базовый APK + APK-файлы конфигурации), не должен превышать 150 МБ. Любые последующие загрузки, такие как загрузка функционального модуля (и его конфигурационных APK-файлов) по запросу, также должны соответствовать этому ограничению размера сжатой загрузки. Пакеты активов не влияют на это ограничение по размеру, но имеют другие ограничения по размеру.

Если при загрузке пакета программных средств Play Console обнаружит, что размер любой из возможных загрузок программного средства или его функций по запросу превышает 150 МБ, будет получено сообщение об ошибке [31].

Наборы Android App Bundle не поддерживают файлы расширения APK (\*.obb). Итак, если появляется ошибкой при публикации пакета программных средств, необходимо использовать один из следующих советов, чтобы уменьшить размер загружаемых сжатых APK:

- 1 Убедиться, что включены все APK конфигурации, установлены `enableSplit = true` для каждого типа APK конфигурации. Это гарантирует, что пользователи загружают только код и ресурсы, необходимые для программного средства на своем устройстве.

- 2 Убедиться, что уменьшен размер программного средства, посредством удаления неиспользуемого кода и ресурсов.

- 3 Рассмотреть вариант преобразования функций, которые используются только некоторыми из ваших пользователей, в функциональные модули,

которые программное средство может загрузить позже по запросу. Необходимо иметь в виду, что для этого может потребоваться некоторый рефакторинг программного средства, поэтому для начала необходимо уменьшить размер программного средства.

Финальным шагом работы над проектом, перед публикацией программного средства, является создание иконки. Для этого воспользуемся бесплатным векторным графическим редактором Figma, который был описан в разделе 2. Данный графический онлайн-редактор поддерживает большое количество шаблонов. Для создания иконки, воспользуемся шаблоном Android Adaptive Icon [32] (рисунок 4.1).

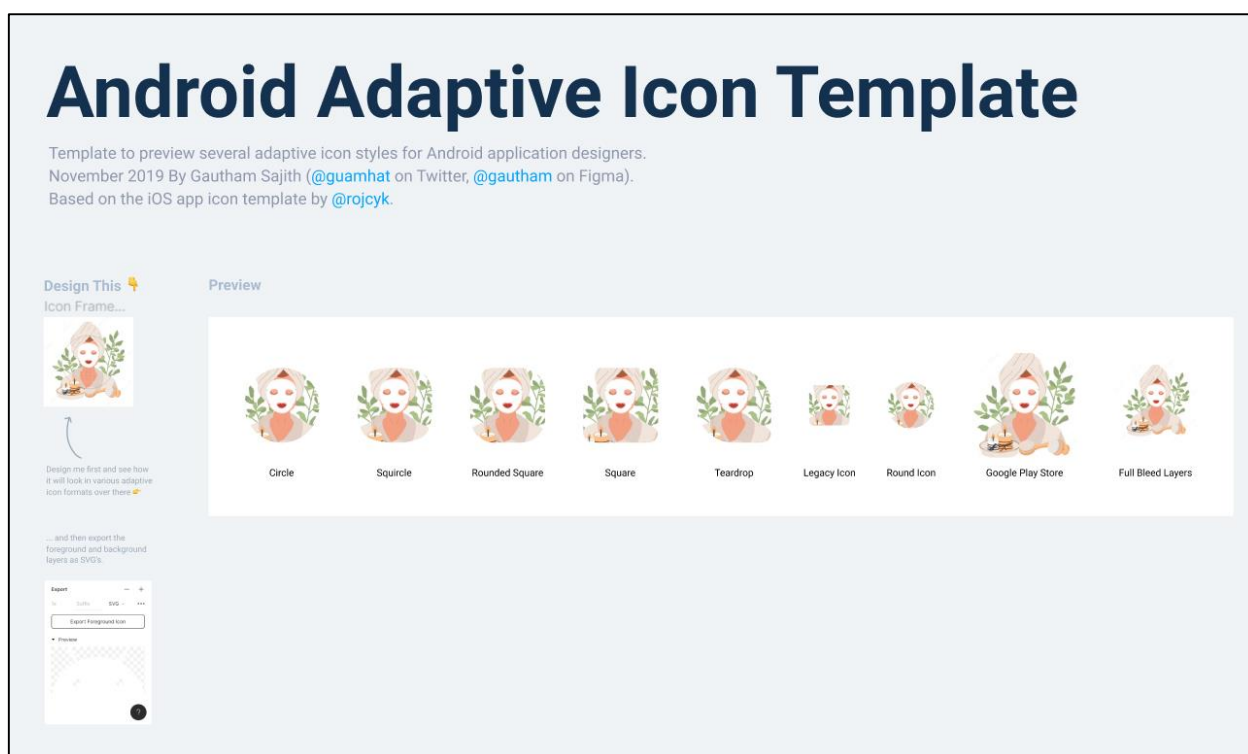


Рисунок 4.1 – Создание иконки программного средства с помощью шаблона Android Adaptive Icon

После создания иконки обновляем файл манифеста, добавив путь к созданной иконке (рисунок 4.2)



Рисунок 4.2 – Обновленный файл манифеста

Для публикации программного средства прежде всего необходимо создать аккаунт разработчика (рисунок 4.3)

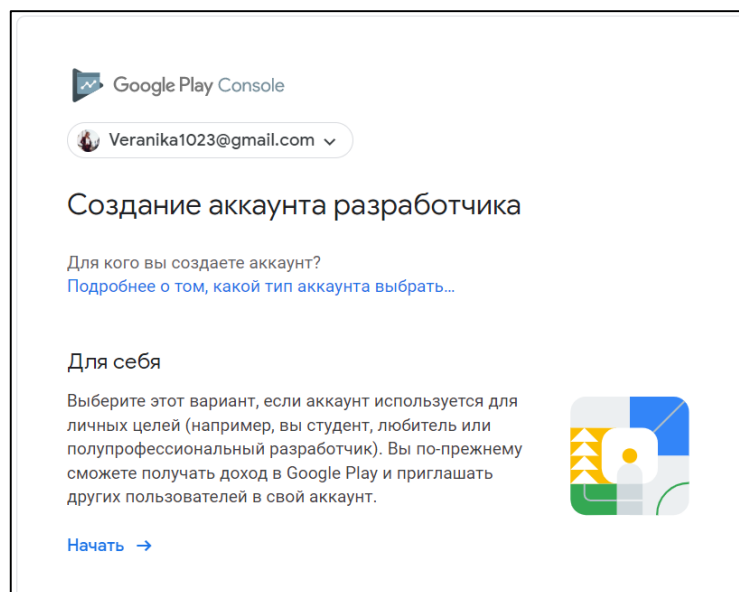


Рисунок 4.3 – Создание аккаунта разработчика

Далее необходимо создать уникальный ключ подписания программных средств с помощью «Программы подписания приложений». После создания и экспорта данного ключа можно приступить к сборке программного средства в формат .aab.

При создании Android App Bundle были выполнены следующие шаги:

- 1 Открыт проект в Android Studio (рисунок 4.4)

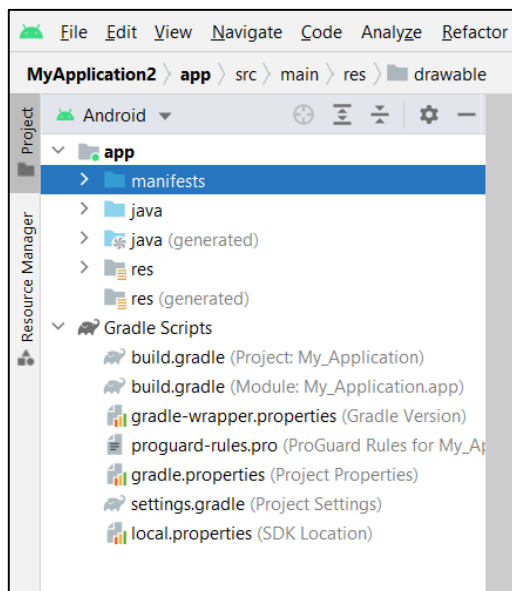


Рисунок 4.4 – Открытый проект в Android Studio

- 2 Во вкладке «Build» выбрана функция «Generate Singned Bundle/APK» (рисунок 4.5)

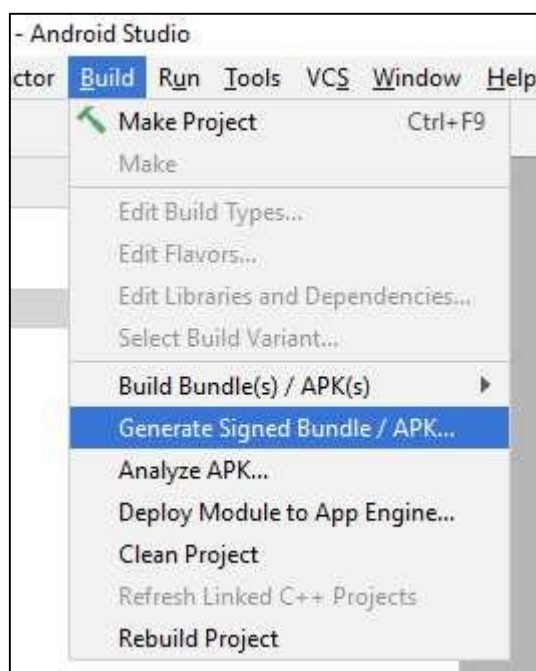


Рисунок 4.5 – Выбранная функция «Generate Singned Bundle/APK»

3 В появившемся окне генерации выбрать пункт «Android App Bundle» (рисунок 4.6)

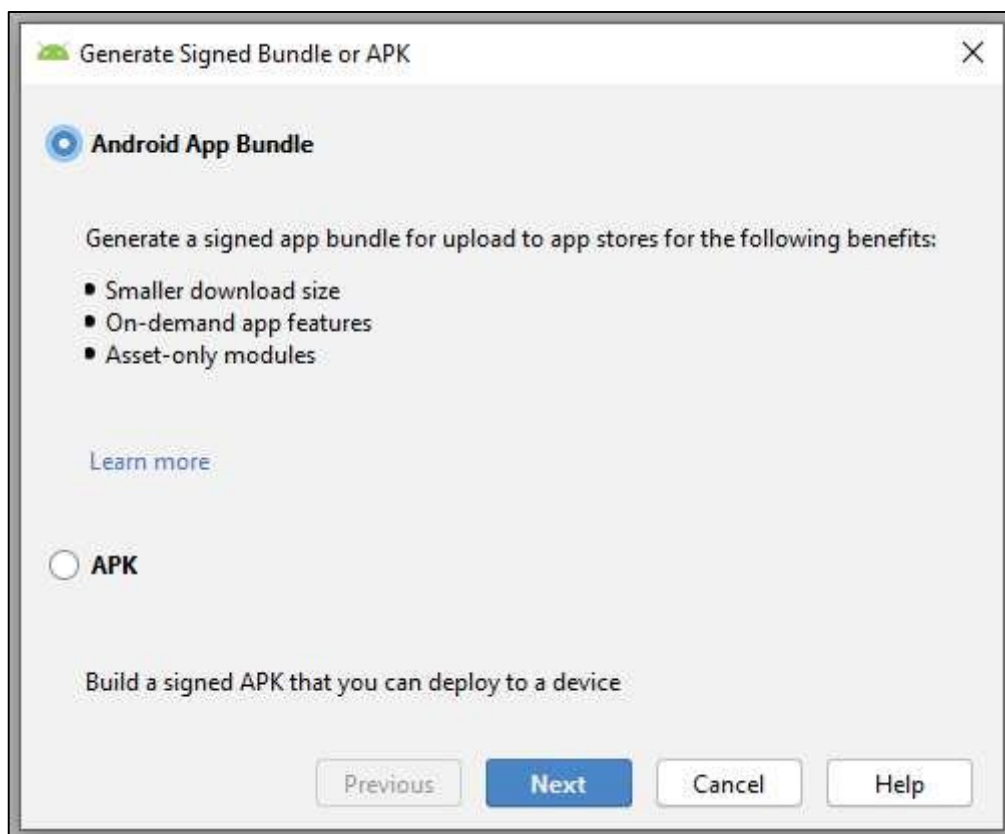


Рисунок 4.6 – «Generate Singned Bundle/APK»

4 Далее необходимо ввести ключ для подписания программного средства в расширении .keystore (рисунок 4.7).

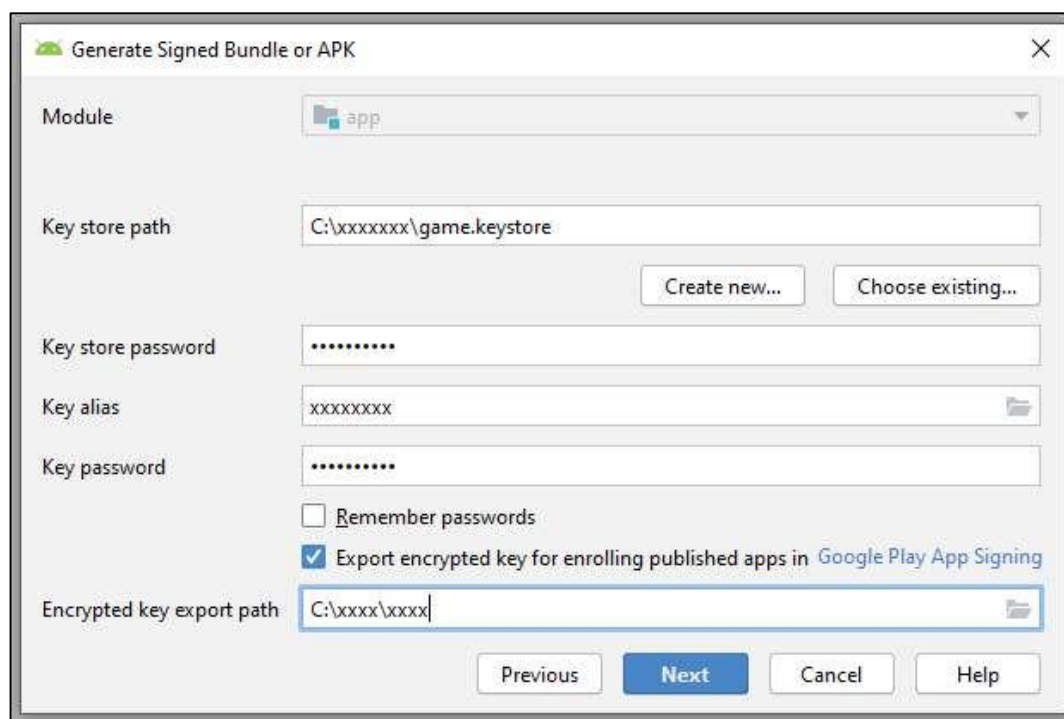


Рисунок 4.7 – Создание файла .aab

После выполнения данных шагов был получен сам файл программного средства в формат .aab.

Теперь программное средство готово для распространения в Google Play или на других площадках для размещения программных средств под операционную систему Android.

Для того чтобы программное средство появилось в Google Play, необходимо загрузить его в Play Console. После загрузки программного средства в Play Console, следует предоставить описание, снимки экрана, возрастное ограничение и прочую информацию, позволяющую определить назначение и возможности программного средства.

Минимальные технические требования к мобильным устройствам для работы с приложением: 1. Операционная система: Android 6.0 и выше 2. Размер ОЗУ: от 2 ГБ 3. Поддержка 3G, 4G 4. Поддержка WiFi 5. Размер встроенной памяти: от 8 ГБ.

## 4.2 Руководство по эксплуатации программным средством

Настоящее руководство по эксплуатации программным средством предназначено для ознакомления пользователя с техническими характеристиками и функциональными возможностями программного средства «SkinCare».

На рисунке 4.8 представлен экран авторизации – первое окно которое видит пользователь при входе в программное средство.



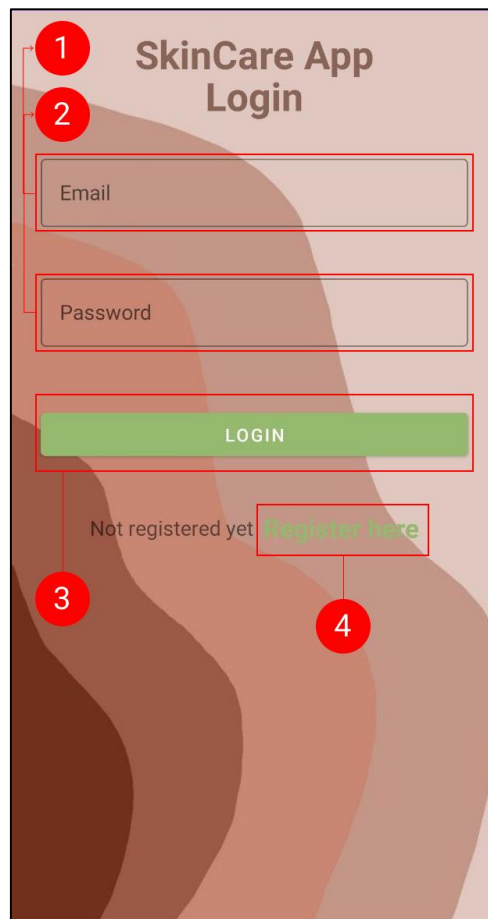


Рисунок 4.8 – Экран авторизации

Экран авторизации содержит следующие элементы:

- 1 Окно ввода почты(логина) пользователя
- 2 Окно ввода пароля пользователя
- 3 Кнопка «Login», для авторизации пользователя
- 4 Ссылка «Register here» для пользователей, которые еще не зарегистрированы

Когда пользователь заполняет поле «Email», введенные данные обязательно должны содержать символ «@» и «.», например, test@test.test. Также в поле «Password» пользователь должен ввести минимум 6 символов, например, 123456. Если хотя бы одно из этих условий не соблюдено, программное средство выдает ошибку. Если пользователь успешно авторизован программное средство также оповещает о том, что пользователь авторизован успешно.

Если пользователь не зарегистрирован, у него есть возможность кликнуть на кликабельную ссылку «Register here» и перейти на экран регистрации. На рисунке 4.9 показан экран регистрации пользователя. Он реализован на подобии экрана авторизации.

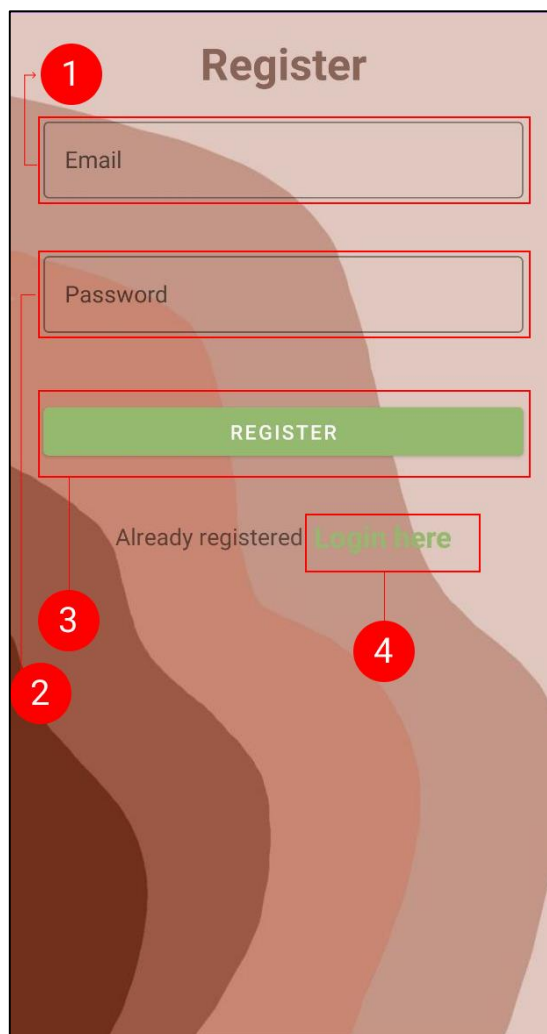


Рисунок 4.9 – Экран регистрации

- 1 Поле ввода почты пользователя
- 2 Поле ввода пароля пользователя
- 3 Кнопка «Register», при нажатии которой пользователь попадает на экран авторизации
- 4 Ссылка «Login Here» для пользователей, которые уже зарегистрированы

Логика заполнения данных подобна экрану авторизации. Здесь присутствуют те же правила при заполнении полей «Email» и «Password». Однако при нажатии на кнопку «Register» не происходит входа в программное средство, а происходит переход на страницу авторизации, где пользователю необходимо ввести данные, которые уже вписаны в базу данных.

Процедуры регистрации происходят лишь один раз, повторно авторизоваться в программном средстве, если пользователь вышел из него, необходимости нет. Но если пользователь переустановил программное средство, авторизацию необходимо будет пройти.

На рисунке 4.10 представлен экран «Advices». На котором пользователь может найти советы и интересные статьи по уходу.

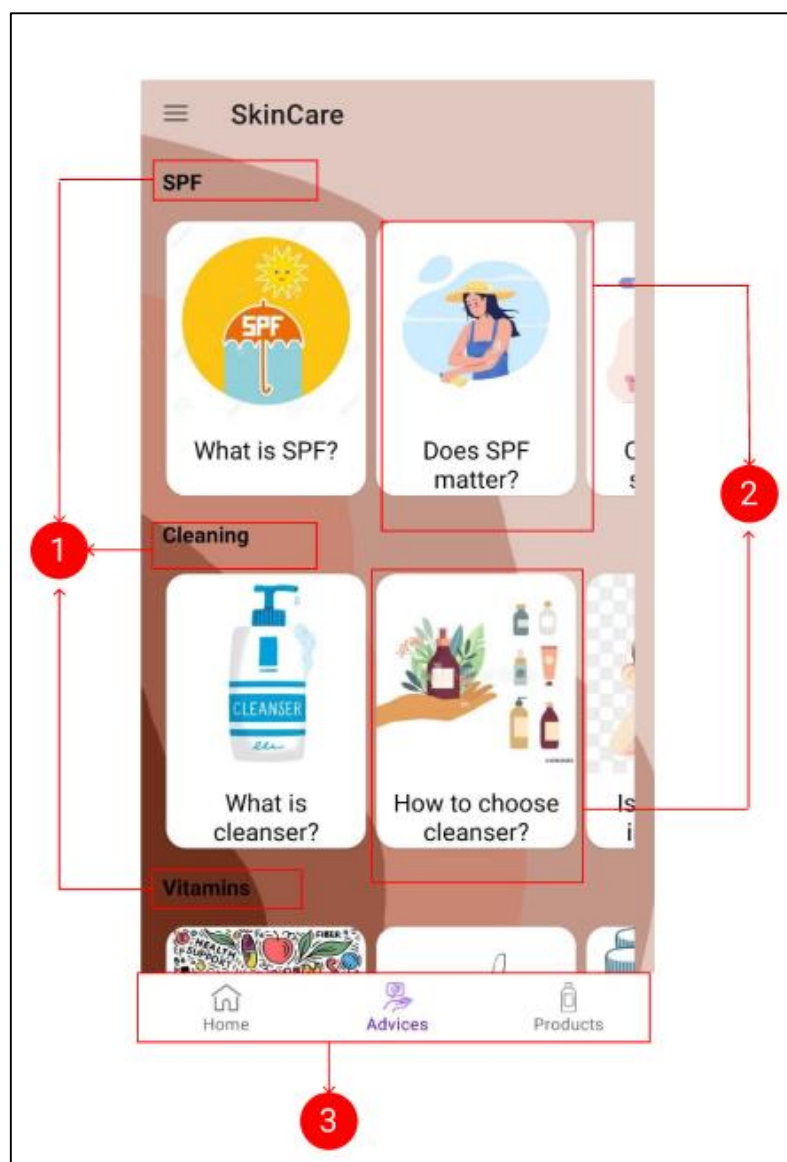


Рисунок 4.10 – Экран «Advices»

- 1 Разделы статей и советов по уходу
- 2 Кликабельные окна, для перехода на страницу интересующей информации
- 3 Навигация в программном средстве

Советы разделены на категории для удобства поиска совета по необходимой теме. В каждой категории есть прокручиваемый список советов. В данном списке каждая карточка кликабельна. Если на нее нажать, то произойдет переход на экран Advice Page (рисунок 4.11).

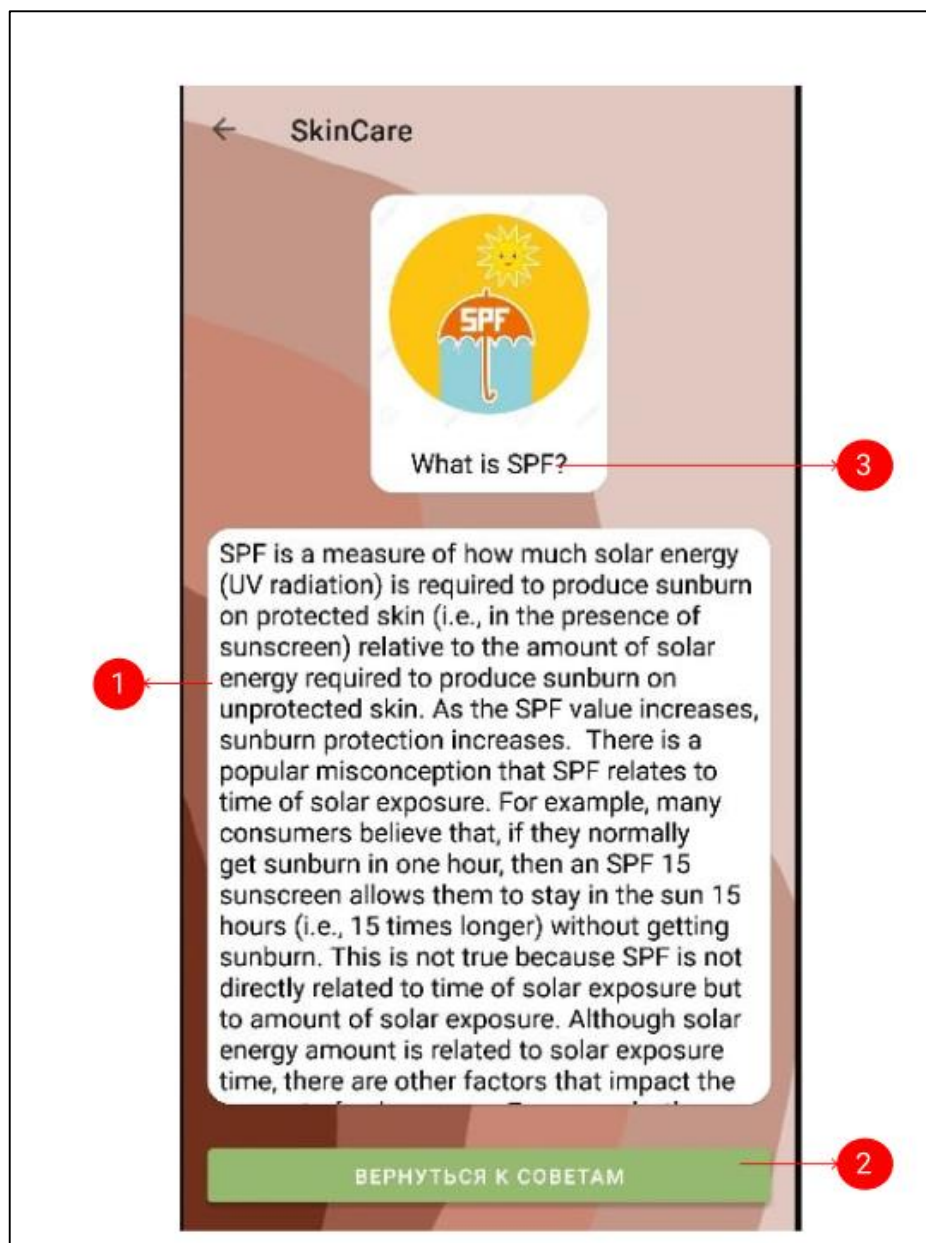


Рисунок 4.11– Экран «Advice Page»

- 1 Прокручиваемый текст совета
- 2 Кнопка выхода из экрана «Advice Page» обратно к экрану «Advices»
- 3 Название совета, совпадающее с названием на экране «Advices»

На данном экране у пользователя есть возможность детальнее ознакомиться с советом по уходу. Для того, чтобы не ограничивать или не сокращать текст, у пользователя есть возможность листать текст вверх-вниз, что достаточно удобно при чтении. После прочтения пользователь может вернуться обратно на экран «Advices» с помощью кнопки «Вернуться к советам».

На подобии экрана «Advices» реализован экран «Products» (рисунок 4.12).

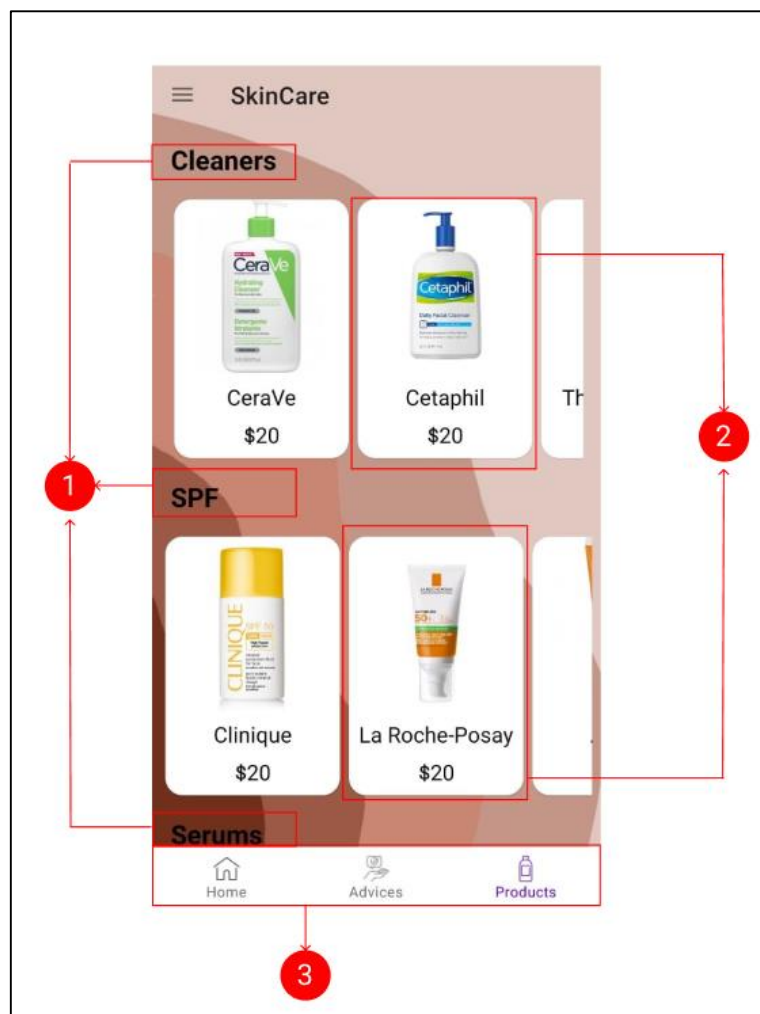


Рисунок 4.12 – Экран «Products»

- 1 Категории продуктов
- 2 Кликабельные окна, для перехода на страницу интересующего продукта
- 3 Навигация в программном средстве

На данном экране у пользователя есть возможность просматривать продукты по категориям, что удобно при поиске необходимого средства ухода. Также пользователь может увидеть, как выглядит продукт, благодаря картинке внутри карточки каждого продукта. И для удобства пользователя в карточке есть не только название продукта, но и примерная цена чтобы пользователь мог сразу видеть, что ему подходит по цене, а что нет.

При нажатии на любой продукт происходит переход на страницу с дополнительной информацией о продукте (рисунок 4.13).



Рисунок 4.13 – Экран с дополнительной информацией о продукте

- 1 Описание продукта
- 2 Кнопка добавления/удаления продукта из избранного
- 3 Кнопка перехода на страницу избранного
- 4 Навигация в программном средстве

На данном экране у пользователя есть возможность ознакомиться с описанием и составом продукта с помощью прокручиваемого вверх-вниз текста. Также чтобы не забыть какой продукт понравился по всем параметрам, пользователь может добавить продукт в избранное с помощью кнопки «Add to favorites», которая при добавлении меняется на «Remove from favorites». И соответственно пользователь может посмотреть список добавленных в избранное им продуктов нажав на кнопку «Favorites».

Возможность ежедневного и еженедельного контроля происходит с помощью главной страницы программного средства (рисунок 4.14).

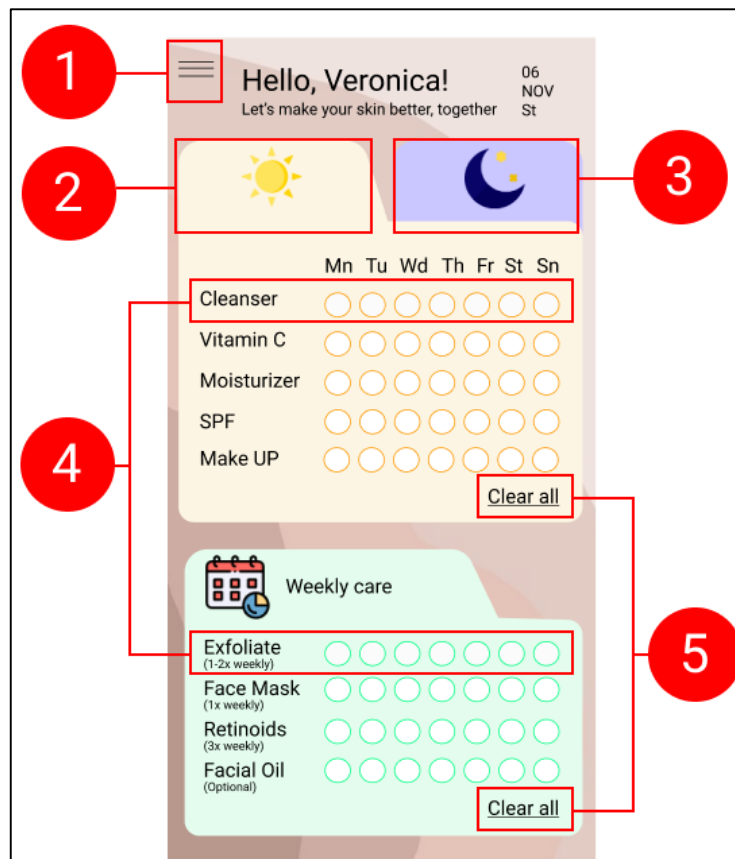


Рисунок 4.14 – Экран «Home»

- 1 Кнопка, при нажатии которой появляется боковое меню
- 2 Виджет, для контроля дневного ухода
- 3 Виджет, для контроля вечернего ухода
- 4 Радиобаттоны, которые помогают контролировать уход
- 5 Возможность очистить все отмеченные шаги ухода

На данном экране пользователь у пользователя есть возможность контролировать свой ежедневный и еженедельный уход с помощью специальных зон. В каждой зоне присутствует ряд радиобаттонов, которые пользователь должен заполнять, как только выполняет какой-то шаг ухода. Для удобства все шаги прописаны в столбик и также, чтобы пользователь помнил, когда какой уход был есть разделение по дням недели. Такая система позволяет сильно упростить уход за кожей, благодаря четкой системности.

При нажатии на кнопку 1 рисунка 4.14, у пользователя появится боковое меню, в котором есть такие разделы как: «Test», «Favorites», «Statistics» (рисунок 4.15).



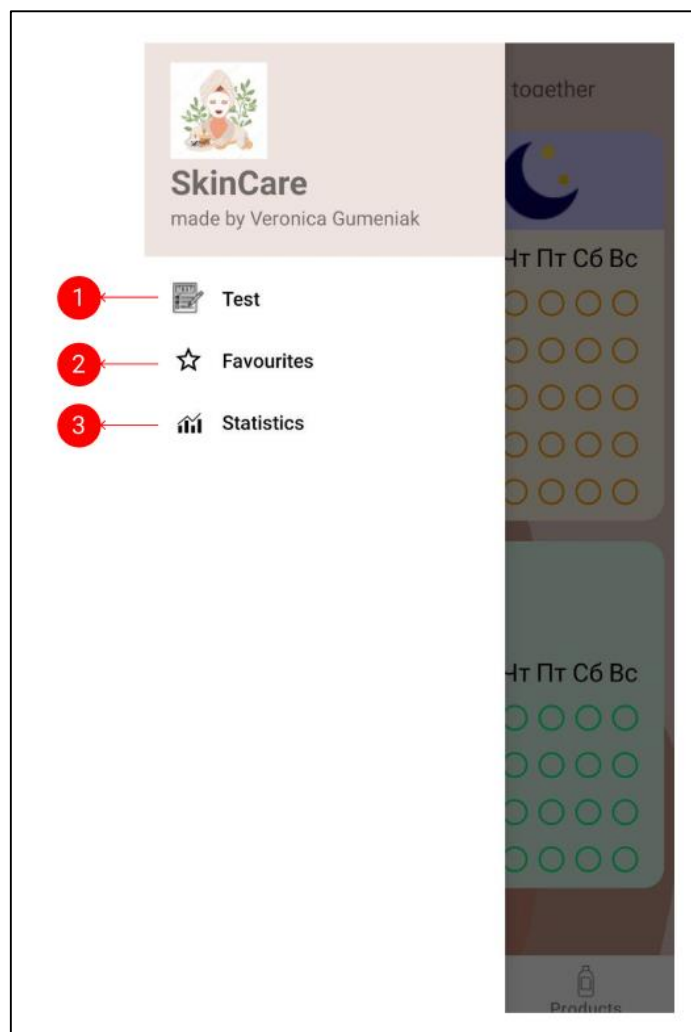


Рисунок 4.15 – Боковое меню программного средства

- 1 Кнопка перехода на экран тестирования
- 2 Кнопка перехода на экран избранных продуктов
- 3 Кнопка перехода на экран статистики

Основная задача данного экрана состоит в том, чтобы не самые используемые возможности скрыть с главного экрана и переместить их в легкодоступное боковое меню, которое пользователь может вызвать одним нажатием. В данном боковом меню присутствует 3 возможности. Переход на страницу тестирования, с помощью которого пользователь может узнать свой тип кожи. Вторая возможность, это просмотр списка избранных продуктов, которые он сам добавил. И третья это возможность просмотра статистики какие продукты пользуются спросом среди пользователей.

На рисунке 4.16 показан экран при переходе по кнопке «Test».



**Skin type test**

Let's determine your skin type. Answer a few questions below.

☐ (намерена) обращаться к косметологу

Моя кожа чиста, высыпаний

☐ не было с подросткового периода

**Вопрос 8**

Какие шаги ухода присутствуют в уходе на данный момент?

☒ Очищение, тонизирование, увлажнение

☐ Очищение, тонизирование, увлажнение, пилинг, увлажняющие маски

☐ Только увлажнение

☐ Не использую никаких продуктов

**Получить результат**

Рисунок 4.16 – Экран «Test»

1 Карточка вопроса

2 Кнопка «Получить результат» для перехода на экран с результатом

Данный экран и возможность прохождения теста на определения типа кожи очень важная возможность для пользователей, которые только начинают разбираться в уходе. В тесте всего 8 вопросов, направленные на выявление текущего состояния кожи и текущего ухода, для подбора персональных рекомендаций. Пользователю необходимо ответить на все вопросы, иначе кнопка «Получить результат» будет неактивна. При нажатии на кнопку «Получить результат» пользователь переходит на экран Result (4.17).

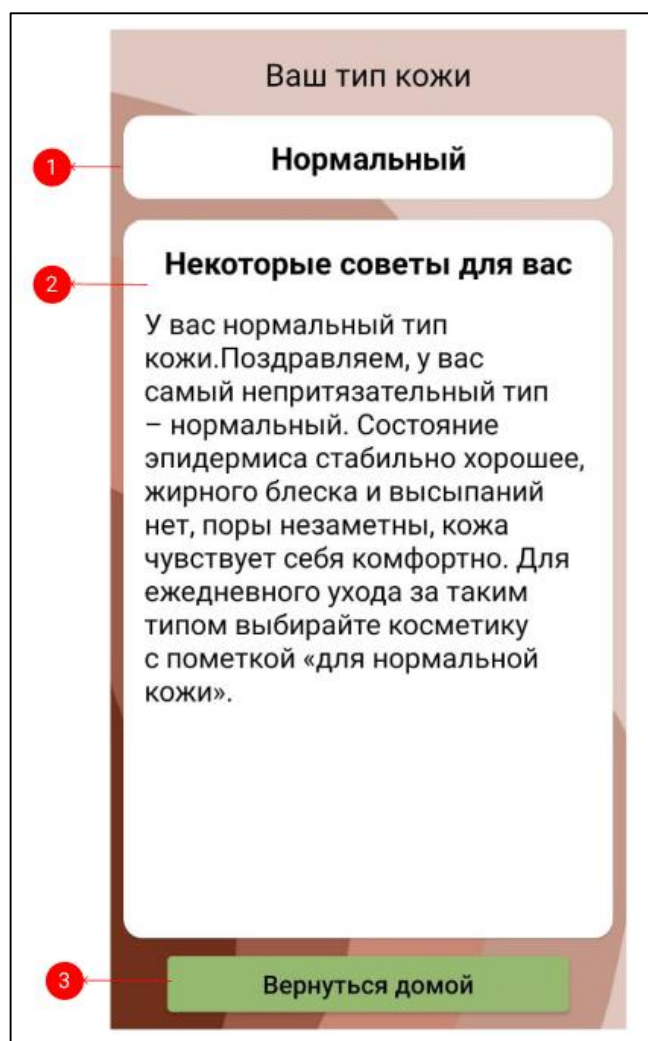


Рисунок 4.17 – Экран «Result»

- 1 Определенный тип кожи
- 2 Советы по уходу для конкретного типа кожи
- 3 Кнопка «Вернуться домой»

Экран «Result» также важен для пользователей, кто еще ничего не знает о своем типе кожи и как за ним ухаживать. С помощью данного экрана пользователь может узнать свой тип кожи, прочитать советы по уходу в прокручиваемом тексте и с помощью кнопки «Вернуться домой» вернуться на главный экран программного средства.

Второй опцией в боковом меню является возможность просмотра списка избранных продуктов (рисунок 4.18).

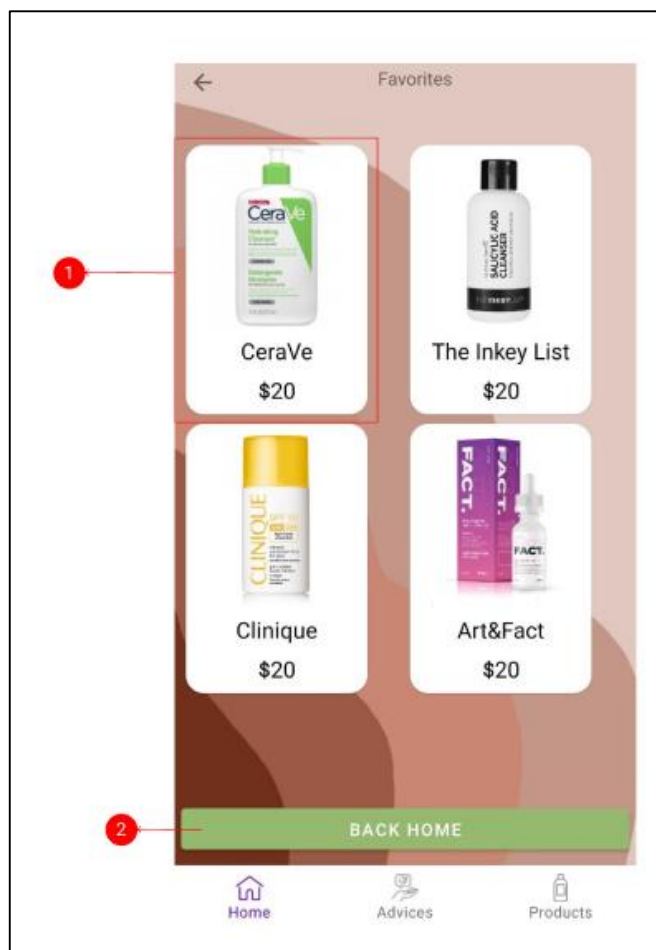


Рисунок 4.18 – Экран «Favorites»

- 1 Продукт, добавленный пользователем в избранное
- 2 Кнопка «Back home»

На данном экране у пользователя есть возможность просмотреть полный список продуктов, которые он добавил в избранное. А также есть возможность вернуться на домашнюю страницу программного средства с помощью кнопки «Back Home».

И третья возможность бокового меню – это переход на страницу «Statistics». Где пользователь может увидеть какие продукты пользуются спросом, среди других пользователей (рисунок 4.19).

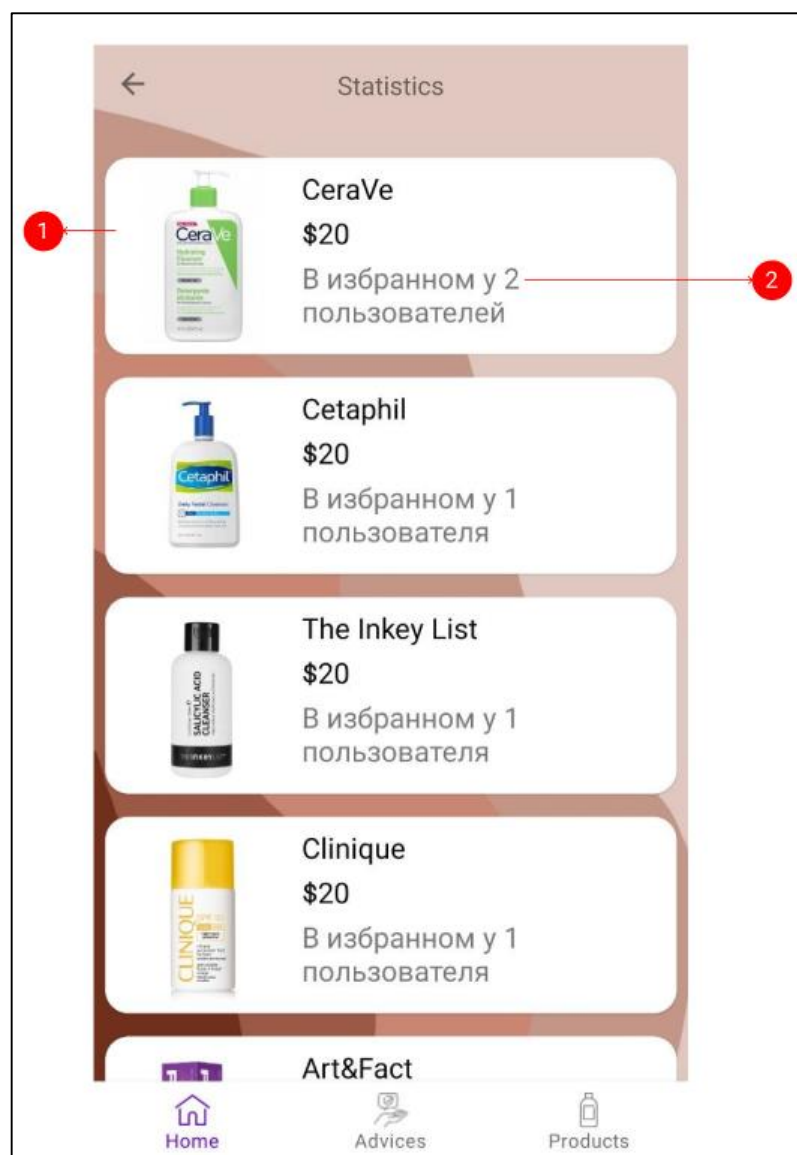


Рисунок 4.19 – Экран «Statistics»

1 Карточка продукта

2 Количество пользователей, добавивших этот продукт в избранное

У данного экрана главная цель – это отображение статистики по пользователям, т.е. пользователь может видеть какое количество раз конкретный продукт добавлен в избранное.

Используя данное руководство по эксплуатации можно с легкостью разобраться в работе программного средства и оценить все возможности.

## **5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА «SKINCARE» ПОД ОПЕРАЦИОННУЮ СИСТЕМУ ANDROID ДЛЯ КОНТРОЛЯ И СОВЕТОВ ПО КОСМЕТИЧЕСКОМУ УХОДУ**

### **5.1 Характеристика программного средства**

Разработанное программное средство предназначено для контроля и советов по косметическому уходу. Целью данного дипломного проекта является оптимизация процесса контроля ежедневного ухода пользователя. Данное программное средство выполняет задачи по контролю и подбору рекомендаций и «уходовых» средств для пользователя. Основными функциями, которые выполняет данное программное средство являются: ежедневный контроль ухода, хранение и отображение информации для советов по уходу, возможность определения типа кожи с помощью теста, возможность получения персональных рекомендаций.

Разработчиками программного средства являются менеджер, бизнес-аналитик, веб-дизайнер, тестировщик, Java-Android разработчик.

Потенциальными заказчиками могут быть, как и отдельно взятые пользователи по всему миру, так и косметологи, которые напрямую работают с клиентами, которым данный продукт необходим.

В результате внедрения данного программного средства планируется сократить время пользователя на поиск информации в сети Интернет, а также обеспечить ему помощь в формировании знаний о шагах ежедневного ухода для дальнейшего улучшения качества кожи.

### **5.2 Расчет инвестиций (затрат) в разработку (модернизацию, совершенствование) программного средства**

Инвестициями для организации-разработчика программного средства являются затраты на его разработку, которые рассчитываются в следующей последовательности, представленной в таблице 5.2.

Первоначально производится расчет затрат на основную заработную плату команды разработчиков.

Данный расчет осуществляется исходя из состава и численности команды, размера месячной заработной платы каждого участника команды, а также трудоемкости работ, выполняемых при разработке программного средства отдельными исполнителями по формуле:

$$З_o = K_{\text{пр}} \sum_{i=1}^n З_{\text{чи}} \cdot t_i, \quad (5.1)$$

где  $k_{np}$  – коэффициент премий;  $n$  – категории исполнителей, занятых разработкой программного средства;  $З_{чi}$  – часовая заработная плата исполнителя  $i$ -й категории, р.;  $t_i$  – трудоемкость работ, выполняемых исполнителем  $i$ -й категории, определяется исходя из сложности разработки программного обеспечения и объема выполняемых им функций, ч.

Часовая заработная плата каждого исполнителя определяется путем деления его месячной заработной платы (оклад плюс надбавки) на количество рабочих часов в месяце в данном проекте принято равным 168 ч.

Размер месячной заработной платы соответствует сложившемуся на рынке труда размеру заработной платы в Республике Беларусь на 2021 год для сотрудников различных категорий ИТ-отрасли. Данные взяты с сайта <https://salaries.dev.by/>.

В работе над проектом задействованы следующие разработчики:

- Project manager, с трудоемкостью в 80 часов;
- Бизнес-аналитик, с трудоемкостью в 160 часов;
- Software engineer, с трудоемкостью в 320 часов;
- UI/UX дизайнер, с трудоемкостью в 120 часов;
- QA-специалист, с трудоемкостью в 240 часов.

Расчет затрат на основную заработную плату представлен в таблице 5.1.

Таблица 5.1 – Расчет затрат на основную заработную плату команды разработчиков

| Категория исполнителя                                    | Месячная заработная плата, р. | Часовая заработная плата, р. | Трудоемкость работ, ч | Итого, р. |
|--|-------------------------------|------------------------------|-----------------------|-----------|
| 1  | 3                             | 4                            | 5                     | 6         |
| Project manager  | 6244                          | 37,17                        | 80                    | 2973,6    |
| Бизнес-аналитик  | 3838                          | 22,85                        | 160                   | 3655,24   |
| S  | 4414                          | 26,27                        | 320                   | 8407,62   |
| UI/UX дизайнер   | 2559                          | 15,23                        | 120                   | 1827,86   |
| QA-специалист  | 2589                          | 15,41                        | 240                   | 3698,57   |
| Всего затраты на основную заработную плату разработчиков |                               |                              |                       | 20562,89  |

Так как при расчете заработной платы используется среднемесячная заработная плата в Республике Беларусь для сотрудников различных категорий ИТ-отрасли, то премия не рассчитывалась.

Для организации-заказчика инвестициями в разработку (модернизацию) программного средства является цена программного средства), которая может определяться следующими альтернативными способами:

- в процессе переговоров между разработчиком и заказчиком ( $Ц_{дог}$ );
- на основе средних рыночных цен на программные средства  $Ц_p$ .

Таблица 5.2 – Методика формирования цены программного средства на

основе затрат

| Наименование статьи затрат                                | Формула (таблица) для расчета  |
|---|--|
| Основная заработная плата разработчиков                   | Формула (5.1), см. табл. 5.1   |
| Дополнительная заработная плата разработчиков             | $З_д = \frac{З_о \cdot Н_д}{100}, \quad (5.2)$ <p>где <math>Н_д</math> - норматив дополнительной заработной платы (принят равным 10%)</p>                    |
| Отчисления на социальные нужды                            | $Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (5.3)$   |
| Прочие расходы  | $Р_{пр} = \frac{З_о \cdot Н_{пр}}{100}, \quad (5.4)$   |
| Общая сумма затрат на разработку                          | $З_p = З_о + З_д + Р_{соц} + Р_{пр}, \quad (5.5)$  |
| Плановая прибыль, включаемая в цену программного средства | $\Pi_{пс} = \frac{З_p \cdot Р_{пс}}{100}, \quad (5.6)$ <p>где <math>Р_{пс}</math> – рентабельность затрат на разработку программного средства, (25–40 %)</p> |
| Отпускная цена программного средства                      | $\Pi_{пс} = З_p + \Pi_{пс}, \quad (5.7)$   |

Рассчитаем затраты на дополнительную заработную плату команды разработчиков. Дополнительная заработная плата включает выплаты, предусмотренные законодательством о труде, и определяется по формуле 5.2, где  $З_о$  – основная заработная плата исполнителей;

$Н_д$  – норматив дополнительной заработной платы (принят равным 10%).

Таким образом, размер дополнительной заработной платы составит:

$$З_д = \frac{20562,89 \cdot 10}{100} = 2056,28 \text{ р.}$$

После расчет отчислений на социальные нужды, в фонд социальной защиты населения и на обязательное страхование, определяется в соответствии с действующими законодательными актами по формуле 5.3, где  $Н_{соц}$  – ставка отчислений в ФСЗН и Белгосстрах (в соответствии с действующим законодательством по состоянию на 01.01.2021 г. – 34,6%).

Таким образом, размер отчислений на социальные нужды составляют:

$$Р_{соц} = \frac{(20562,89 + 2056,28) \cdot 34,6}{100} = 7826,23 \text{ р.}$$

Следующим этапом определяются прочие расходы, которые включаются в себестоимость программного средства в проценте от затрат на основную заработную плату команды разработчиков по формуле 5.4, где  $H_{пр}$  – норматив прочих расходов (принят равным 30%).

К прочим расходам относятся командировочные расходы, плата сторонним организациям за подготовку и переподготовку кадров.

Расчет расходов по статье «Прочие расходы»:

$$P_{пр} = \frac{20562,89 \cdot 30}{100} = 6168,87 \text{ р.}$$

Общая сумма инвестиций на разработку рассчитывается по формуле 5.5: При расчете всех статей затрат общая сумма составит:

$$З_p = 20562,89 + 2056,28 + 7826,23 + 6168,87 = 36614,27 \text{ р.}$$

Далее необходимо рассчитать плановую прибыль, включаемую в цену программного средства по формуле 5.6, где за рентабельность программного средства возьмем средние 33%. Расчет плановой прибыли:

$$\Pi_{пс} = \frac{36614,27 \cdot 33}{100} = 12082,7091 \text{ р.}$$

Затем необходимо рассчитать отпускную цену программного средства по формуле 5.7:

$$Ц_{пс} = 36614,27 + 12082,7091 = 48696,9791 \text{ р.}$$

Расчет формирования цены программного средства на основе затрат представлен в таблице 5.3.

Таблица 5.3 – Формирование цены программного средства на основе затрат

| Наименование статьи затрат                                | Расчет по формуле (в таблице) | Значение, р. |
|---|-------------------------------|--------------|
| Основная заработная плата разработчиков                   | См. табл. 5.1                 | 20562,89     |
| Дополнительная заработная плата разработчиков             | Формула (5.2)                 | 2056,28      |
| Отчисления на социальные нужды                            | Формула (5.3)                 | 7826,23      |
| Прочие расходы  | Формула (5.4)                 | 6168,87      |
| Общая сумма затрат на разработку                          | Формула (5.5)                 | 36614,27     |
| Плановая прибыль, включаемая в цену программного средства | Формула (5.6)                 | 12082,7091   |
| Отпускная цена программного средства                      | Формула (5.7)                 | 48696,9791   |



### 5.3 Расчет результата от разработки и использования программного средства

Для организации-разработчика экономическим эффектом является прирост чистой прибыли, полученной от разработки и реализации программного средства заказчику, который может быть рассчитан следующим образом: программное средство будет реализовываться организацией-разработчиком по отпускной цене, сформированной на основе затрат на разработку (см. табл. 5.2), то экономический эффект, полученный организацией-разработчиком, в виде прироста чистой прибыли от его разработки, определяется по формуле:

$$\Delta\Pi_{\text{ч}} = \Pi_{\text{пс}} \left( 1 - \frac{H_{\text{п}}}{100} \right), \quad (5.8)$$

где  $\Pi_{\text{пс}}$  – прибыль, включаемая в цену программного средства, р.

$$\Delta\Pi_{\text{ч}} = 12082,7091 \left( 1 - \frac{18}{100} \right) = 9907,8215, \text{ р.}$$

### 5.4 Расчет показателей экономической эффективности разработки и использования программного средства на рынке

Для организации-разработчика программного средства оценка экономической эффективности разработки осуществляется с помощью расчета простой нормы прибыли (рентабельности затрат на разработку программного средства)) по формуле:

$$P_{\text{и}} = \frac{\Delta\Pi_{\text{ч}}}{Z_{\text{р}}} \cdot 100\%, \quad (5.9)$$

где  $\Delta\Pi_{\text{ч}}$  – прирост чистой прибыли, полученной от разработки программного средства организацией-разработчиком по индивидуальному заказу, р.(см. п. 5.3);  $Z_{\text{р}}$  – затраты на разработку программного средства организацией-разработчиком, р.

$$P_{\text{и}} = \frac{9907,8215}{36614,27} \cdot 100\% = 27,06\%$$

Таким образом, полные затраты на разработку составили 36614,27 рублей, чистая прибыль от использования программного средства составила 9907,8215 рублей. Таким образом уровень рентабельности составит 27,06%.

## ЗАКЛЮЧЕНИЕ

Основной задачей данного дипломного проекта является разработка мобильного программного средства для контроля и советов по косметическому уходу. В ходе работы над проектом был спроектирован графический пользовательский интерфейс, разработана объектная модель программного средства, а также реализованы следующие возможности: регистрация/авторизация пользователей; контроль ежедневного/еженедельного ухода; отображение подбора советов по уходу за кожей; определение типа кожи посредством тестирования.

На первом этапе проектирования, были проанализированы предметная область темы дипломного проектирования, требования и возможности программного средства, обоснован выбор языка программирования и средств разработки, а также представлен обзор существующих программных средств для контроля и советов по косметическому уходу. В процессе постановки решаемой задачи была уяснена цель дипломного проектирования и кратко описаны шаги реализации.

На втором этапе была разработана объектная модель программного средства, спроектирована архитектура программного средства, описаны используемые средства, технологии и сторонние библиотеки, разработаны алгоритмы функционирования программного средства, а также разработан и обоснован пользовательский интерфейс программного средства.

На третьем этапе была произведена оценка использования оперативной памяти, процессора, памяти при взаимодействии с программным средством. Была произведена временная оценка запуска программного средства.

На четвёртом этапе был произведён ввод в эксплуатацию программного средства, а также приведено руководство к пользованию разработанным программным средством.

В заключительной части было приведено технико-экономическое обоснование разработки программного средства, описаны функции, назначение и потенциальные пользователи программного средства, а также произведён расчёт затрат и экономический эффект от разработки программного средства.

В ходе работы над дипломным проектом, были приобретены навыки работы с такими библиотеками как Material Design, ViewBinding, JetPack Navigation, Glide, Firebase auth, Firebase database, Firebase crashlytics, Firebase analytics и Glide. Помимо этого, были получены знания о базовых подходах и принципах дизайна при разработке пользовательских интерфейсов.

Итого полученные знания и навыки позволили разработать готовое программное средство, которое при постоянном использовании помогает упростить уход за кожей, и они также могут быть использованы в будущем.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Зачем нам нужен уход за кожей? [Электронный ресурс]. – Режим доступа: <https://nonicare.ru/beauty/beautyclub/1006.html>. – Дата доступа: 30.10.2021.
- [2] Статистика заболевших. [Электронный ресурс]. – Режим доступа: <https://www.rbc.ru/society/16/07/2018/5b487e189a794760d7be404a>. – Дата доступа: 30.10.2021.
- [3] Статистика мобильных приложений 2021: загрузки, тренды и доходность индустрии. [Электронный ресурс]. – Режим доступа: <https://vc.ru/marketing/245003-statistika-mobilnyh-prilozheniy-2021-zagruzki-trendy-i-dohodnost-industrii>. – Дата доступа: 31.10.2021
- [4] Статистика Android vs iOS в 2021 году.[Электронный ресурс]. – Режим доступа: [https://gdetrtraffic.com/Analitika/Android\\_vs\\_iOS](https://gdetrtraffic.com/Analitika/Android_vs_iOS). – Дата доступа: 31.10.2021.
- [5] Google Play. TroveSkin - Get Clearer Skin. [Электронный ресурс]. – Режим доступа: <https://play.google.com/store/apps/details?id=com.trove&hl=ru&gl=US>. – Дата доступа: 01.11.2021.
- [6] Google Play. Picky - Skincare Community & Rewards. [Электронный ресурс]. – Режим доступа: <https://play.google.com/store/apps/details?id=care.jivaka.picky&hl=ru&gl=US>. – Дата доступа: 01.11.2021.
- [7] Google Play. Skin Bliss: Cosmetics & Beauty. [Электронный ресурс]. – Режим доступа: <https://play.google.com/store/apps/details?id=com.getskinbliss.skinbliss&hl=ru&gl=US>. – Дата доступа: 01.11.2021.
- [8] Дон Гриффитс и Дэвид Гриффитс. Head First. Программирование для Android: учебник, 2016. – 704 с.
- [9] Что такое Котлин? [Электронный ресурс]. – Режим доступа: <https://itproger.com/news/java-vs-kotlin-kto-zhe-kruche>. – Дата доступа: 01.11.2021
- [10] Шаблон Model-View-ViewModel. [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/xamarin/xamarin-forms/enterprise-application-patterns/mvvm>. – Дата доступа: 03.11.2021.
- [11] Как работать с RecyclerView? [Электронный ресурс]. – Режим доступа: <https://devcolibri.com/как-работать-с-recyclerview/>. – Дата доступа: 03.11.2021.
- [12] База данных: определение и классификация. [Электронный ресурс]. – Режим доступа: [https://studopedia.ru/20\\_12244\\_baza-dannih-opredelenie-i-klassifikatsiya.html](https://studopedia.ru/20_12244_baza-dannih-opredelenie-i-klassifikatsiya.html). – Дата доступа: 05.11.2021.
- [13] Google Firebase: что это за сервис и для чего его можно использовать. [Электронный ресурс]. – Режим доступа: [https://codernet.ru/articles/drugoe/google\\_firebase\\_что\\_это\\_за\\_servis\\_i\\_dlya\\_chego\\_ego\\_mozhno\\_ispolzovat/](https://codernet.ru/articles/drugoe/google_firebase_что_это_за_servis_i_dlya_chego_ego_mozhno_ispolzovat/). – Дата доступа: 05.11.2021.

[14] База данных Firebase в реальном времени. [Электронный ресурс]. – Режим доступа: <https://firebase.google.com/docs/database>. – Дата доступа: 06.11.2021.

[15] Обмен сообщениями Firebase Cloud. [Электронный ресурс]. – Режим доступа: <https://firebase.google.com/docs/cloud-messaging>. – Дата доступа: 06.11.2021.

[16] Пользовательский интерфейс. [Электронный ресурс]. – Режим доступа: <https://askusers.ru/blog/obuchenie/polzovatelskiy-interfeys/>. – Дата доступа: 08.11.2021.

[17] Что такое UX/UI-дизайн и как попасть в эти профессии. [Электронный ресурс]. – Режим доступа: [https://skillbox.ru/media/design/ux\\_ui\\_dizayn\\_chno\\_eto\\_takoe/](https://skillbox.ru/media/design/ux_ui_dizayn_chno_eto_takoe/). – Дата доступа: 08.11.2021.

[18] Гид по Фигме для начинающих веб-дизайнеров. [Электронный ресурс]. – Режим доступа: <https://tilda.education/articles-figma>. – Дата доступа: 09.11.2021.

[19] Что такое Material Design и как делать анимацию в стиле Google. [Электронный ресурс]. – Режим доступа: [https://skillbox.ru/media/design/chno\\_takoe\\_material\\_design/](https://skillbox.ru/media/design/chno_takoe_material_design/). – Дата доступа: 10.11.2021

[20] Алгоритмы в программировании. [Электронный ресурс]. – Режим доступа: <http://www.interface.ru/home.asp?artId=35216>. – Дата доступа: 15.11.2021.

[21] Lucidchart - интеллектуальное построение диаграмм. [Электронный ресурс]. – Режим доступа: <https://www.lucidchart.com/pages/ru>. – Дата доступа: 15.11.2021.

[22] Выполнение схем алгоритмов в Visio. [Электронный ресурс]. – Режим доступа: <https://www.cherchenie.by/information/executing-algorithms-in-visio>. – Дата доступа: 16.11.2021.

[23] Юзабилити мобильных приложений: практические советы. [Электронный ресурс]. – Режим доступа: <https://pear-advert.ru/yuzabiliti-mobilnykh-prilozhenij-prakticheskie-sovety>. – Дата доступа: 10.11.2021.

[24] Гугл Аналитика. [Электронный ресурс]. – Режим доступа: <https://firebase.google.com/docs/analytics>. – Дата доступа: 22.11.2021.

[25] App startup time. [Электронный ресурс]. – Режим доступа: <https://developer.android.com/topic/performance/vitals/launch-time>. – Дата доступа: 22.11.2021.

[26] The Android Profiler. [Электронный ресурс]. – Режим доступа: <https://developer.android.com/studio/profile/android-profiler>. – Дата доступа: 23.11.2021.

[27] Inspect your app's memory usage with Memory Profiler. [Электронный ресурс]. – Режим доступа: <https://developer.android.com/studio/profile/memory-profiler>. – Дата доступа: 23.11.2021.

[28] Inspect network traffic with Network Profiler. [Электронный ресурс]. – Режим доступа: <https://developer.android.com/studio/profile/network-profiler>. – Дата доступа: 24.11.2021.

[29] Надежность. [Электронный ресурс]. Режим доступа: <https://studfile.net/preview/837772/page:22/>. – Дата доступа: 24.11.2021.

[30] Обзор Crashlytics. [Электронный ресурс]. Режим доступа: <https://coba.tools/crashlytics>. – Дата доступа: 25.11.2021.

[31] Поддержка формата публикации мобильных приложений Android App Bundle (AAB). [Электронный ресурс]. Режим доступа: <https://wonderland.v8.1c.ru/blog/podderzhka-formata-publikatsii-mobilnykh-prilozheniy-android-app-bundle-aab/>. – Дата доступа: 10.12.2021.

[32] Android Adaptive Icon Template. [Электронный ресурс]. Режим доступа: <https://www.figma.com/community/file/778486426750404931>. – Дата доступа: 11.12.2021.

## ПРИЛОЖЕНИЕ А (обязательное)

### Отчет по анализу заимствования материала пояснительной записки

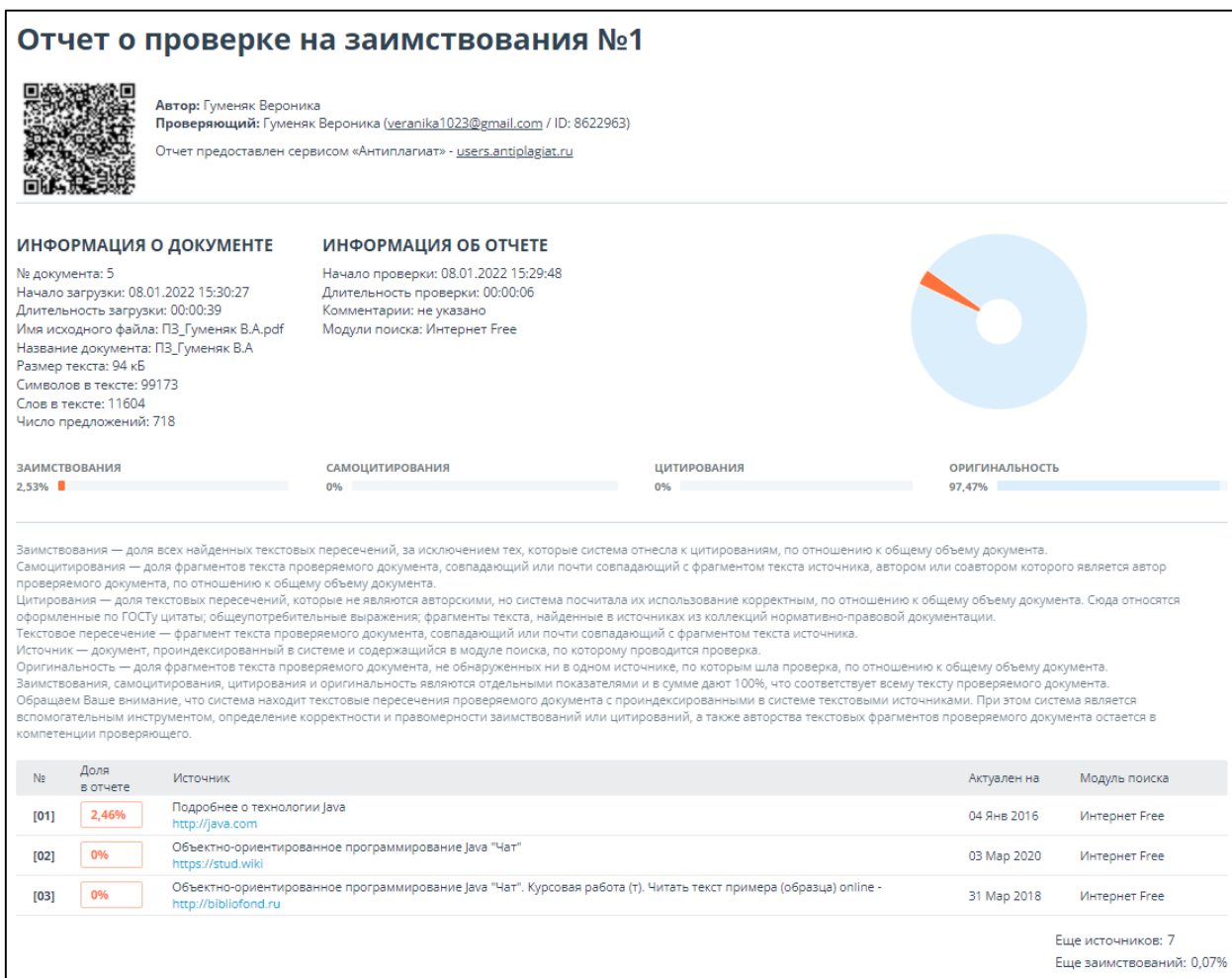


Рисунок А.1 – Отчет по анализу заимствования

## ПРИЛОЖЕНИЕ Б

### (обязательное)

#### Листинги программного кода

##### Код класса MainActivity:

```
package com.example.myapplication.ui

import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.core.view.isVisible
import androidx.drawerlayout.widget.DrawerLayout
import androidx.navigation.NavController
import androidx.navigation.findNavController
import androidx.navigation.ui.AppBarConfiguration
import androidx.navigation.ui.setupActionBarWithNavController
import androidx.navigation.ui.setupWithNavController
import by.kirich1409.viewbindingdelegate.viewBinding
import com.example.myapplication.Login
import com.example.myapplication.R
import com.example.myapplication.databinding.ActivityMainBinding
import com.google.android.material.bottomnavigation.BottomNavigationView
import com.google.firebase.auth.FirebaseAuth

class MainActivity : AppCompatActivity() {
    //Все вьюхи достаются через этот объект по именам айдишников
    private val binding by viewBinding(ActivityMainBinding::bind)
    private val auth: FirebaseAuth = FirebaseAuth.getInstance()
    private val navController: NavController by lazy { findNavController(
        R.id.nav_host_fragment) }

    private val drawerLayout: DrawerLayout
        get() = binding.drawerLayout

    private val toolbar: Toolbar
        get() = binding.content.toolbar

    private val bottomNav: BottomNavigationView
        get() = binding.content.contentMain.bottomNav

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //Установка ToolBar
        setSupportActionBar(toolbar)

        setupView()
    }

    override fun onStart() {
        super.onStart()
        navigateToLoginIfNeeded()
    }

    /**
     * Получает текущего авторизованного пользователя
     * если он равен null, то пользователь не авторизован и
     * переходим на экран авторизации/регистрации
     */
    private fun navigateToLoginIfNeeded() {
```

```

        val user = auth.currentUser
        if (user == null) {
            startActivity(Intent(this@MainActivity, Login::class.java))
            finish()
        } else {
            supportActionBar?.let { actionBar ->
                actionBar.title = getString(R.string.title_toolbar_hello,
user.displayName)
                actionBar.subtitle = getString(R.string.subtitle_toolbar)
            }
        }
    }

    private fun setupView() {
        addOnDestinationChangedListener()
        addDrawerItemSelectedListener()

        val appBarConf = AppBarConfiguration(
            topLevelDestinationIds = setOf(
                R.id.fragment_home,
                R.id.fragment_advice,
                R.id.fragment_products
            ),
            drawerLayout = drawerLayout
        )
        setupActionBarWithNavController(navController, appBarConf)
        bottomNav.setupWithNavController(navController)
        bottomNav.setOnItemReselectedListener { }

        toolbar.setNavigationOnClickListener { binding.drawerLayout.open() }
    }

    private fun addDrawerItemSelectedListener() {
        //Слушатель за нажатиями на элементы в Drawer
        binding.navView.setNavigationItemSelectedListener { item ->
            val navId = when (item.itemId) {
                R.id.drawer_test -> R.id.fragment_test
                R.id.drawer_favourites -> R.id.fragment_favourites
                R.id.drawer_favourites_stats -> R.id.fragment_favourite_stats
                else -> 0
            }
            navigateFromDrawer(navId)
            true
        }
    }

    private fun navigateFromDrawer(navId: Int) {
        if (navId == 0) return
        val currentNavId = navController.currentDestination?.id ?: 0
        if (currentNavId != navId) {
            navController.navigate(navId)
        }
        drawerLayout.closeDrawers()
    }

    private fun addOnDestinationChangedListener() {
        //Слушатель за сменой экрана
        navController.addOnDestinationChangedListener { _, destination, _ ->
            val isVisible = when (destination.id) {
                R.id.fragment_test -> false
                R.id.fragment_test_result -> false
                else -> true
            }
            toolbar.isVisible = isVisible
        }
    }

```



```

        val isBottomNavVisible = when (destination.id) {
            R.id.fragment_test -> false
            R.id.fragment_test_result -> false
            R.id.fragment_advice_details -> false
            else -> true
        }
        bottomNav.isVisible = isBottomNavVisible
    }
}

override fun onBackPressed() {
    if (drawerLayout.isOpen) {
        drawerLayout.closeDrawers()
    } else {
        super.onBackPressed()
    }
}
}

```

### Код класса Login:

```

package com.example.myapplication;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.example.myapplication.ui.MainActivity;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.android.material.textfield.TextInputEditText;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;

public class Login extends AppCompatActivity {

    TextInputEditText etLoginEmail;
    TextInputEditText etLoginPassword;
    TextView tvRegisterHere;
    Button btnLogin;

    FirebaseAuth mAuth;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        etLoginEmail = findViewById(R.id.etLoginEmail);
        etLoginPassword = findViewById(R.id.etLoginPass);
        tvRegisterHere = findViewById(R.id.tvRegisterHere);
        btnLogin = findViewById(R.id.btnLogin);

        mAuth = FirebaseAuth.getInstance();

        btnLogin.setOnClickListener(view -> {
            loginUser();
        });
        tvRegisterHere.setOnClickListener(view -> {
            startActivity(new Intent(Login.this, SignUp.class));
        });
    }
}

```

```

        });
    }

    private void loginUser(){
        String email = etLoginEmail.getText().toString();
        String password = etLoginPassword.getText().toString();

        if (TextUtils.isEmpty(email)){
            etLoginEmail.setError("Email cannot be empty");
            etLoginEmail.requestFocus();
        }else if (TextUtils.isEmpty(password)){
            etLoginPassword.setError("Password cannot be empty");
            etLoginPassword.requestFocus();
        }else{
            mAuth.signInWithEmailAndPassword(email,password).addOnCom-
pleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()){
                        Toast.makeText(Login.this, "User logged in success-
fully", Toast.LENGTH_SHORT).show();
                        startActivity(new Intent(Login.this, MainActiv-
ity.class));

                        finish();
                    }else{
                        Toast.makeText(Login.this, "Log in Error: " +
task.getException().getMessage(), Toast.LENGTH_SHORT).show();
                    }
                }
            });
        }
    }
}

```

### Код класса SignUp:

```

package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;

import com.google.android.material.textfield.TextInputEditText;
import com.google.firebase.auth.FirebaseAuth;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.android.material.textfield.TextInputEditText;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;

```

```

public class SignUp extends AppCompatActivity {

    TextInputEditText etRegEmail;
    TextInputEditText etRegPassword;
    TextView tvLoginHere;
    Button btnRegister;

    FirebaseAuth mAuth;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sign_up);

        etRegEmail = findViewById(R.id.etRegEmail);
        etRegPassword = findViewById(R.id.etRegPass);
        tvLoginHere = findViewById(R.id.tvLoginHere);
        btnRegister = findViewById(R.id.btnRegister);

        mAuth = FirebaseAuth.getInstance();

        btnRegister.setOnClickListener(view ->{
            createUser();
        });

        tvLoginHere.setOnClickListener(view ->{
            startActivity(new Intent(SignUp.this, Login.class));
        });
    }

    private void createUser(){
        String email = etRegEmail.getText().toString();
        String password = etRegPassword.getText().toString();

        if (TextUtils.isEmpty(email)){
            etRegEmail.setError("Email cannot be empty");
            etRegEmail.requestFocus();
        }else if (TextUtils.isEmpty(password)){
            etRegPassword.setError("Password cannot be empty");
            etRegPassword.requestFocus();
        }else{
            mAuth.createUserWithEmailAndPassword(email,password).addOnCom-
            pleteListener(new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()){
                        Toast.makeText(SignUp.this, "User registered success-
                        fully", Toast.LENGTH_SHORT).show();
                        startActivity(new Intent(SignUp.this, Login.class));
                    }else{
                        Toast.makeText(SignUp.this, "Registration Error: " +
                        task.getException().getMessage(), Toast.LENGTH_SHORT).show();
                    }
                }
            });
        }
    }
}

```

## Код класса HomeFragment:

```
package com.example.myapplication.ui.home

import android.content.res.ColorStateList
import android.os.Bundle
import android.view.View
import androidx.core.content.ContextCompat
import androidx.fragment.app.Fragment
import by.kirich1409.viewbindingdelegate.viewBinding
import com.example.myapplication.R
import com.example.myapplication.databinding.FragmentHomeBinding

class HomeFragment : Fragment(R.layout.fragment_home) {
    private val binding: FragmentHomeBinding by viewBinding(FragmentHomeBind-
ing::bind)

    override fun onCreateView(view: View, savedInstanceState: Bundle?) {
        super.onCreateView(view, savedInstanceState)
        setupView()
    }

    private fun setupView() {
        with(binding) {
            tabDay.setOnClickListener { changeTabColor(TabMode.Day) }
            tabNight.setOnClickListener { changeTabColor(TabMode.Night) }
        }

        private fun changeTabColor(mode: TabMode) {
            val tabBackgroundColorRes = when (mode) {
                TabMode.Day -> R.color.tab_day_color
                TabMode.Night -> R.color.tab_night_color
            }
            val tabBackgroundColor = ContextCompat.getColor(requireContext(),
tabBackgroundColorRes)

            val checkBoxColorRes = when (mode) {
                TabMode.Day -> R.color.checkbox_day_color
                TabMode.Night -> R.color.checkbox_night_color
            }
            val checkBoxColor = ContextCompat.getColor(requireContext(), check-
BoxColorRes)

            with(binding) {
                dayNightLayout.backgroundTintList = ColorStateList.valueOf(tab-
BackgroundColor)
                listOf(
                    cleanserCheckboxes,
                    vitaminCCheckboxes,
                    moisturizerCheckboxes,
                    spfCheckboxes,
                    makeUpCheckboxes
                ).forEach { checkBoxes ->
                    checkBoxes.setCheckboxesTint(checkBoxColor)
                }
            }
        }

        sealed interface TabMode {
            object Day : TabMode
            object Night : TabMode
        }
    }
}
```

## Код класса Advice:

```
package com.example.myapplication.model

import android.os.Parcelable
import com.google.firebase.database.DataSnapshot
import kotlinx.parcelize.Parcelize

@Parcelize
data class Advice(
    val title: String = "",
    val advice: String = "",
    val image: String = "",
    val type: Int = 0
) : Parcelable {
    companion object Types {
        const val Spf = 0
        const val Cleaning = 1
        const val Vitamins = 2
    }
}

fun DataSnapshot.mapToAdvices(): List<Advice> {
    return children.mapNotNull { dataSnapshot ->
        dataSnapshot.getValue(Advice::class.java)
    }
}
```

## Код класса AdvicesFragment:

```
package com.example.myapplication.ui.advices

import android.os.Bundle
import android.view.View
import androidx.core.view.isVisible
import androidx.fragment.app.Fragment
import androidx.lifecycle.LifecycleScope
import androidx.navigation.fragment.findNavController
import by.kirich1409.viewbindingdelegate.viewBinding
import com.example.myapplication.R
import com.example.myapplication.adapter.AdvicesAdapter
import com.example.myapplication.databinding.FragmentAdvicesBinding
import com.example.myapplication.model.Advice
import com.example.myapplication.model.mapToAdvices
import com.example.myapplication.ui.product.ProductsFragmentDirections
import com.example.myapplication.utils.showSnackBar
import com.google.firebase.database.ktx.database
import com.google.firebase.ktx.Firebase
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

class AdvicesFragment : Fragment(R.layout.fragment_advices) {
    private val binding by viewBinding<FragmentAdvicesBinding>::bind
    private val spfAdapter = AdvicesAdapter(::handleAdviceClick)
    private val cleaningAdapter = AdvicesAdapter(::handleAdviceClick)
    private val vitaminsAdapter = AdvicesAdapter(::handleAdviceClick)

    override fun onCreateView(view: View, savedInstanceState: Bundle?) {
        super.onCreateView(view, savedInstanceState)
        setupView()
        loadAdvices()
    }
}
```

```

    }

    private fun setupView() {
        with(binding) {
            spfList.adapter = spfAdapter
            cleaningList.adapter = cleaningAdapter
            vitaminsList.adapter = vitaminsAdapter
        }
    }

    private fun loadAdvices() {
        viewLifecycleOwner.lifecycleScope.launch(Dispatchers.IO) {
            Firebase.database.getReference("Advices")
                .get()
                .addOnCompleteListener { binding.progress.isVisible = false }
                .addOnSuccessListener { dataSnapshot -> showAdvices(dataSnapshot.mapToAdvices()) }
                .addOnFailureListener { showSnackbar(it.localizedMessage.orEmpty()) }
        }
    }

    private fun showAdvices(list: List<Advice>) {
        viewLifecycleOwner.lifecycleScope.launch(Dispatchers.Default) {
            val spfList = list.filter { it.type == Advice.Spf }
            val cleaningList = list.filter { it.type == Advice.Cleaning }
            val vitaminsList = list.filter { it.type == Advice.Vitamins }

            withContext(Dispatchers.Main) {
                spfAdapter.submitList(spfList) {
                    binding.spfTitle.isVisible = true
                    binding.spfList.isVisible = true
                }
                cleaningAdapter.submitList(cleaningList) {
                    binding.cleaningTitle.isVisible = true
                    binding.cleaningList.isVisible = true
                }
                vitaminsAdapter.submitList(vitaminsList) {
                    binding.vitaminsTitle.isVisible = true
                    binding.vitaminsList.isVisible = true
                }
            }
        }
    }

    private fun handleAdviceClick(advice: Advice) {
        val directions = AdvicesFragmentDirections.actionFragmentAdvicesToFragmentAdviceDetails(advice)
        findNavController().navigate(directions)
    }
}

```

## Код класса AdviceDetailsFragment:

```

package com.example.myapplication.ui.advices

import android.os.Bundle
import android.view.View
import androidx.fragment.app.Fragment
import androidx.navigation.fragment.findNavController
import androidx.navigation.fragment.navArgs
import by.kirich1409.viewbindingdelegate.viewBinding

```

```

import com.bumptech.glide.Glide
import com.example.myapplication.R
import com.example.myapplication.databinding.FragmentAdviceDetailsBinding
import com.example.myapplication.model.Advice

class AdviceDetailsFragment: Fragment(R.layout.fragment_advice_details) {
    private val binding by viewBinding(FragmentAdviceDetailsBinding::bind)
    private val args: AdviceDetailsFragmentArgs by navArgs()

    override fun onCreateView(view: View, savedInstanceState: Bundle?) {
        super.onCreateView(view, savedInstanceState)
        setupView()
    }
    private fun setupView() {
        populateAdviceInfo(args.advice)
        binding.btnBackToAdvices.setOnClickListener {
            findNavController().popBackStack()
        }
    }
    private fun populateAdviceInfo(advice: Advice) {
        with(binding) {
            Glide.with(this@AdviceDetailsFragment)
                .load(advice.image)
                .into(adviceCard.image)
            adviceCard.title.text = advice.title
            adviceText.text = advice.advice
        }
    }
}

```

## Код класса Product:

```

package com.example.myapplication.model

import android.os.Parcelable
import com.google.firebase.database.DataSnapshot
import kotlinx.parcelize.Parcelize

@Parcelize
data class Product(
    var id: String = "",
    val name: String = "",
    val description: String = "",
    val type: Int = 0,
    val price: String = "",
    val image: String = ""
) : Parcelable {
    companion object Types {
        const val Spf = 0
        const val Cleaner = 1
        const val Serum = 2
    }
}

fun DataSnapshot.mapToProducts(): List<Product> {
    return children.mapNotNull { dataSnapshot ->
        dataSnapshot.getValue(Product::class.java)?.apply {
            id = dataSnapshot.key.orEmpty()
        }
    }
}

```

## Код класса ProductsFragment:

```
package com.example.myapplication.ui.product

import android.os.Bundle
import android.view.View
import androidx.core.view.isVisible
import androidx.fragment.app.Fragment
import androidx.lifecycle.LifecycleScope
import androidx.navigation.fragment.findNavController
import by.kirich1409.viewbindingdelegate.viewBinding
import com.example.myapplication.R
import com.example.myapplication.adapter.ProductsAdapter
import com.example.myapplication.databinding.FragmentProductsBinding
import com.example.myapplication.model.Product
import com.example.myapplication.model.mapToProducts
import com.example.myapplication.utils.showSnackBar
import com.google.firebase.database.ktx.database
import com.google.firebase.ktx.Firebase
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

class ProductsFragment : Fragment(R.layout.fragment_products) {
    private val binding by viewBinding(FragmentProductsBinding::bind)
    private val spfAdapter = ProductsAdapter(::handleProductClick)
    private val cleanerAdapter = ProductsAdapter(::handleProductClick)
    private val serumAdapter = ProductsAdapter(::handleProductClick)

    override fun onCreateView(view: View, savedInstanceState: Bundle?) {
        super.onCreateView(view, savedInstanceState)
        setupView()
        loadProducts()
    }

    private fun setupView() {
        with(binding) {
            spfList.adapter = spfAdapter
            cleansersList.adapter = cleanerAdapter
            serumsList.adapter = serumAdapter
        }
    }

    private fun loadProducts() {
        viewLifecycleOwner.lifecycleScope.launch(Dispatchers.IO) {
            Firebase.database.getReference("Products")
                .get()
                .addOnCompleteListener { binding.progress.isVisible = false }
                .addOnSuccessListener { dataSnapshot -> showProducts(dataSnapshot.mapToProducts()) }
                .addOnFailureListener { showSnackBar(it.localizedMessage.orEmpty()) }
        }
    }

    private fun showProducts(list: List<Product>) {
        viewLifecycleOwner.lifecycleScope.launch(Dispatchers.Default) {
            val spfList = list.filter { it.type == Product.Spf }
            val cleanerList = list.filter { it.type == Product.Cleaner }
            val serumList = list.filter { it.type == Product.Serum }

            withContext(Dispatchers.Main) {
                spfAdapter.submitList(spfList) {
                    binding.cleansersTitle.isVisible = true
                }
            }
        }
    }
}
```



```

        binding.cleansersList.isVisible = true
    }
    cleanerAdapter.submitList(cleanerList) {
        binding.spfTitle.isVisible = true
        binding.spfList.isVisible = true
    }
    serumAdapter.submitList(serumList) {
        binding.serumsTitle.isVisible = true
        binding.serumsList.isVisible = true
    }
}

}

private fun handleProductClick(product: Product) {
    val directions = ProductsFragmentDirections.actionFragmentProductsToFragmentProductDetails(product)
    findNavController().navigate(directions)
}
}

```

### Код класса ProductDetailsFragment:

```

package com.example.myapplication.ui.product

import android.os.Bundle
import android.view.View
import androidx.core.view.isVisible
import androidx.core.view.isInvisible
import androidx.fragment.app.Fragment
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.LifecycleScope
import androidx.navigation.fragment.findNavController
import androidx.navigation.fragment.navArgs
import by.kirich1409.viewbindingdelegate.viewBinding
import com.bumptech.glide.Glide
import com.example.myapplication.R
import com.example.myapplication.databinding.FragmentProductDetailsBinding
import com.example.myapplication.model.Product
import com.example.myapplication.model.mapToFavourites
import com.example.myapplication.utils.FirebaseUtils
import com.example.myapplication.utils.showSnackBar
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.ktx.auth
import com.google.firebase.database.DataSnapshot
import com.google.firebase.ktx.Firebase
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext

class ProductDetailsFragment : Fragment(R.layout.fragment_product_details) {
    private val binding by viewBinding(FragmentProductDetailsBinding::bind)
    private val args: ProductDetailsFragmentArgs by navArgs()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        setupView()
        loadFavourites()
    }

    private fun setupView() {
        populateProductInfo(args.product)
    }
}

```

```

private fun loadFavourites() {
    val currentUser = Firebase.auth.currentUser
    if (currentUser == null) {
        showUi()
    } else {
        loadFavouritesForUser(currentUser)
    }
}

private fun populateProductInfo(product: Product) {
    with(binding) {
        Glide.with(this@ProductDetailsFragment)
            .load(product.image)
            .into(image)
        name.text = product.name
        description.text = product.description
        price.text = product.price

        btnToFavourites.setOnClickListener { addToFavourites() }
        btnFromFavourites.setOnClickListener { removeFromFavourites() }
        btnShowFavourites.setOnClickListener { showFavourites() }
    }
}

private fun addToFavourites() {
    fun handleSuccess() {
        with(binding) {
            btnToFavourites.isEnabled = true
            btnToFavourites.isInvisible = true
            btnFromFavourites.isVisible = true
        }
    }

    fun handleError(e: Exception) {
        showSnackbar(e.localizedMessage.orEmpty())
        binding.btnFromFavourites.isEnabled = true
    }

    binding.btnToFavourites.isEnabled = false
    viewLifecycleOwner.lifecycleScope.launch(Dispatchers.IO) {
        FirebaseUtils.addProductToFavourites(
            product = args.product,
            successListener = { handleSuccess() },
            errorListener = { handleError(it) }
        )
    }
}

private fun removeFromFavourites() {
    fun handleSuccess() {
        with(binding) {
            btnFromFavourites.isEnabled = true
            btnFromFavourites.isVisible = false
            btnToFavourites.isVisible = true
        }
    }

    fun handleError(e: Exception) {
        showSnackbar(e.localizedMessage.orEmpty())
        binding.btnFromFavourites.isEnabled = true
    }

    binding.btnFromFavourites.isEnabled = false
}

```

```

        viewLifecycleOwner.lifecycleScope.launch(Dispatchers.IO) {
            FirebaseUtils.removeProductFromFavourites(
                currentUser = Firebase.auth.currentUser,
                product = args.product,
                successListener = { handleSuccess() },
                errorListener = { handleError(it) },
            )
        }
    }

    private fun showFavourites() {
        findNavController().navigate(R.id.fragment_favourites)
    }

    private fun loadFavouritesForUser(currentUser: FirebaseUser) {
        fun handleSuccess(dataSnapshot: DataSnapshot) {
            viewLifecycleOwner.lifecycleScope.launch {
                val isAdded = withContext(Dispatchers.Default) {
                    val favourites = dataSnapshot.mapToFavourites()
                    favourites.any { favourite -> favourite.productId ==
args.product.id }
                }
                with(binding) {
                    btnToFavourites.isVisible = !isAdded
                    btnFromFavourites.isVisible = isAdded
                }
                showUi()
            }
        }

        fun handleError(exception: Exception) {
            showUi()
            showSnackbar(exception.localizedMessage.orEmpty())
        }

        viewLifecycleOwner.lifecycleScope.launch(Dispatchers.IO) {
            FirebaseUtils.loadFavouritesForUser(
                currentUser = currentUser,
                successListener = { handleSuccess(it) },
                errorListener = { handleError(it) }
            )
        }
    }

    private fun showUi() {
        binding.uiInfo.isVisible = true
        binding.progress.isVisible = false
    }
}

```

**ПРИЛОЖЕНИЕ В**  
**(обязательное)**  
**Ведомость дипломного проекта**