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

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка Android-приложения для контроля питания на Kotlin

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-351.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ Н.С. Силкина

Автор работы,
студент группы КЭ-403
_____ В.А. Серяков

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

Челябинск, 2024 г.

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

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

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Серякову Владиславу Александровичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка Android-приложения для контроля питания на Kotlin.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. Documentation Android Developers. [Электронный ресурс] URL: https://developer.android.com/docs (дата обращения: 07.02.2024 г.).
3.2. Руководство по упрощенному подсчету КБЖУ. [Электронный ресурс] URL: https://yandex.ru/health/turbo/articles?id=6012 (дата обращения: 07.02.2024 г.).
3.3. Firebase Documentation. [Электронный ресурс] URL: https://firebase.google.com/docs?authuser=0&hl=ru (дата обращения: 07.02.2024 г.).
3.4. Kotlin docs. [Электронный ресурс] URL: https://kotlinlang.org/docs/home.html (дата обращения: 18.02.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Изучить предметную область.
4.2. Изучить методы создания Android-приложений.
4.3. Спроектировать базу данных.
4.4. Спроектировать приложение.

4.5. Реализовать приложение для подсчета калорий.

4.6. Протестировать реализованное приложения.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП Н.С. Силкина

Задание принял к исполнению В.А. Серяков

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В современном мире сохранение здоровья и поддержание правильного питания становятся все более актуальными задачами. По данным Всемирной организации здравоохранения (ВОЗ), более 2,6 миллиарда взрослых людей в мире имеют избыточный вес. Ожирение является риск-фактором для развития многих заболеваний, включая сахарный диабет, сердечно-сосудистые заболевания, некоторые виды рака и другие хронические заболевания. Контроль над потребляемыми продуктами является ключевым фактором для сохранения своего здоровья [1].

Наиболее подходящим решением будет разработка приложения контроля питания на популярной операционной системе Andriod, так как это наиболее популярная операционная система для мобильных устройств в мире. Согласно статистике, на конец 2023 года Android занимает около 70% доли рынка мобильных устройств [2]. Это позволит охватить большую аудиторию. К тому же мобильность данной платформы позволит пользователям всегда иметь постоянный свободный доступ к приложению.

Таким образом, разработка Android-приложения для контроля питания представляет собой актуальную задачу, которая позволит пользователям более эффективно следить за своим рационом питания и улучшить свое здоровье.

Постановка задачи

Целью выпускной квалификационной работы является создание приложения для отслеживания потребления питательных веществ, калорий и других показателей питания, включающего в себя ввод информации о приеме пищи, установку целей потребления и отслеживания прогресса достижения этих целей. В решение этой задачи входит разработка интерфейса пользователя для удобного и интуитивно понятного использования приложения, включающего в себя разработку функционала для создания и отслеживания планов питания.

Для достижения поставленной цели необходимо решить следующие задачи:

1) проанализировать предметную область;
2) спроектировать базу данных;
3) спроектировать мобильное приложение;
4) реализовать и протестировать работу реализованного приложения.

Структура и содержание работы

Работа состоит из введения, шести глав, заключения и списка литературы. Объем работы составляет 56 страниц, объем списка литературы – 18 источников.

В первой главе описывается предметная область приложения, производится анализ аналогов и обзор технологий реализации проекта.

Вторая глава посвящена проектированию системы. В ней произведен анализ требований системы и разработан дизайн.

В третьей главе описано проектирование хранилища данных: произведен обзор баз данных для платформы Android и создана модель.

В четвертой главе описана архитектура системы приложения.

В пятой главе описана реализация мобильного приложения и облачной базы данных.

В шестой главе описано произведенное тестирование системы.

В приложении А содержится спецификация вариантов использования.

В приложении Б представлен листинг авторизации и аутентификации в приложении.

В приложении В представлены скриншоты готового приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор аналогичных приложений

Перед началом разработки функционала для отслеживания питания необходимо провести анализ аналогичных проектов и существующих решений на рынке. Это позволит выявить сильные и слабые стороны конкурентов, а также понять, какие функции и возможности уже есть на рынке и что может быть добавлено в новый проект.

Существует множество приложений и сервисов для отслеживания питания. Рассмотрение самых популярных из них, таких как MyFitnessPal, FatSecret, Lifesum, необходимо для разработки лаконичного, функционального и привлекательного приложения. Каждое из этих приложений имеет свои уникальные особенности и функции.

Приложение My FitnessPal

Лидирующую позицию в списке самых популярных приложения для подсчета калорий уверенно занимает приложение MyFitnessPal [3]. Страница приложения MyFitnessPal в магазине Google Play представлена на рисунке 1.

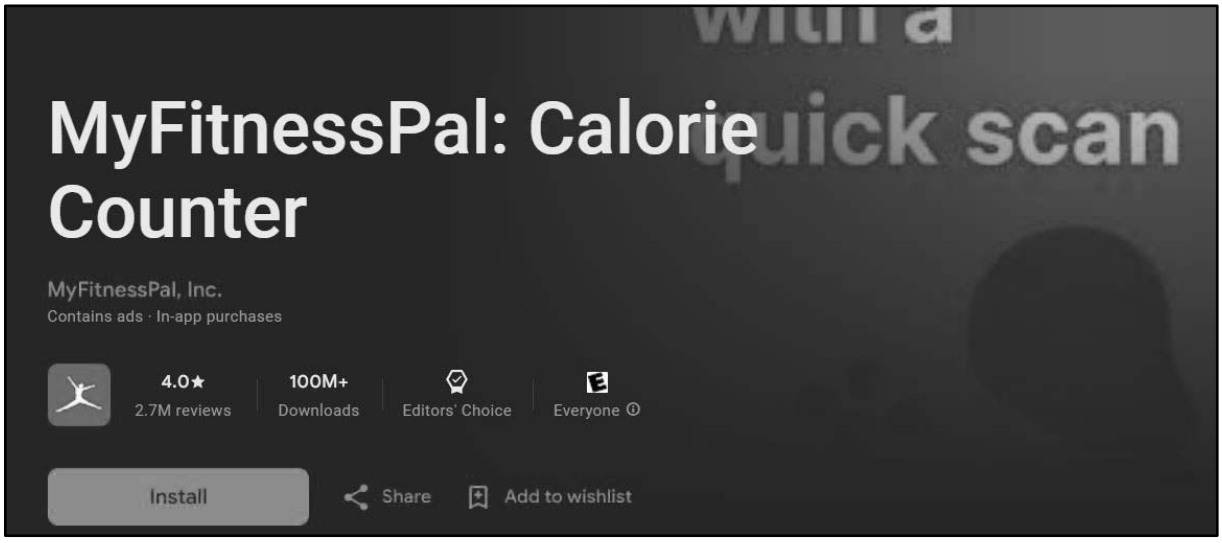

Рисунок 1 – Страница приложения MyFitnessPal на Google Play

По заявлению разработчиков, программа имеет крупнейшую базу данных (более 14 млн. наименований продуктов), которая пополняется ежедневно. В приложении реализован полный комплект функций: создание неограниченного количества собственных блюд, удобная статистика и отчеты о динамике веса, сканер штрих-кодов, статистика по основным питательным веществам, включая белки, жиры, углеводы, сахара, клетчатку и холестерин. Скриншоты приложения MyFitnessPal представлены на рисунке 2.

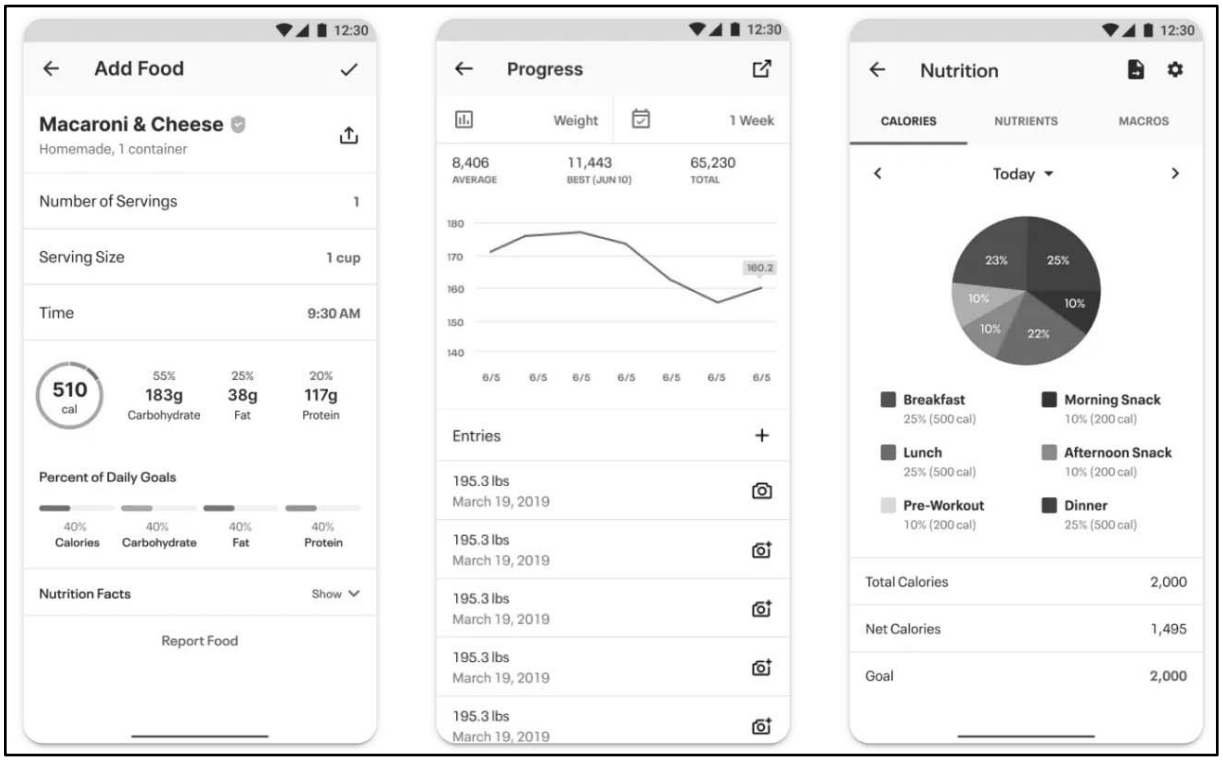

Рисунок 2 – Скриншоты приложения MyFitnessPal

Пользователи приложения отмечают широкий функционал и большую базу данных продуктов как плюсы, но малую функциональность бесплатной версии как минус.

Приложение Fat Secret

Fat Secret – это абсолютно бесплатное приложение для подсчета калорий без премиум-аккаунтов, подписок и рекламы [4]. Страница приложения Fat Secret в магазине Google Play представлена на рисунке 3.

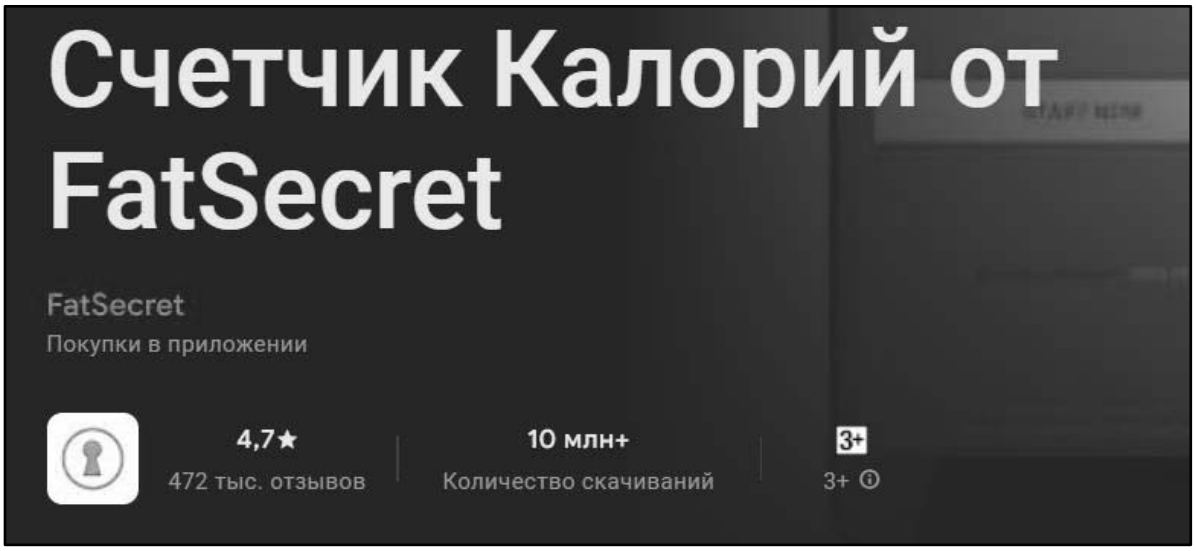

Рисунок 3 – Страница приложения Fat Secret на Google Play

Одним из главных преимуществ программы является приятный, лаконичный и информативный интерфейс. Fat Secret имеет большую продуктовую базу (в том числе и ввод продуктов по штрих-коду), которая разделена на категории: продукты, ресторанные сети, популярные марки, супермаркеты. Помимо стандартных макросов предлагается информация по количеству сахара, натрия, холестерина, клетчатки. Скриншоты приложения FatSecret представлены на рисунке 4.

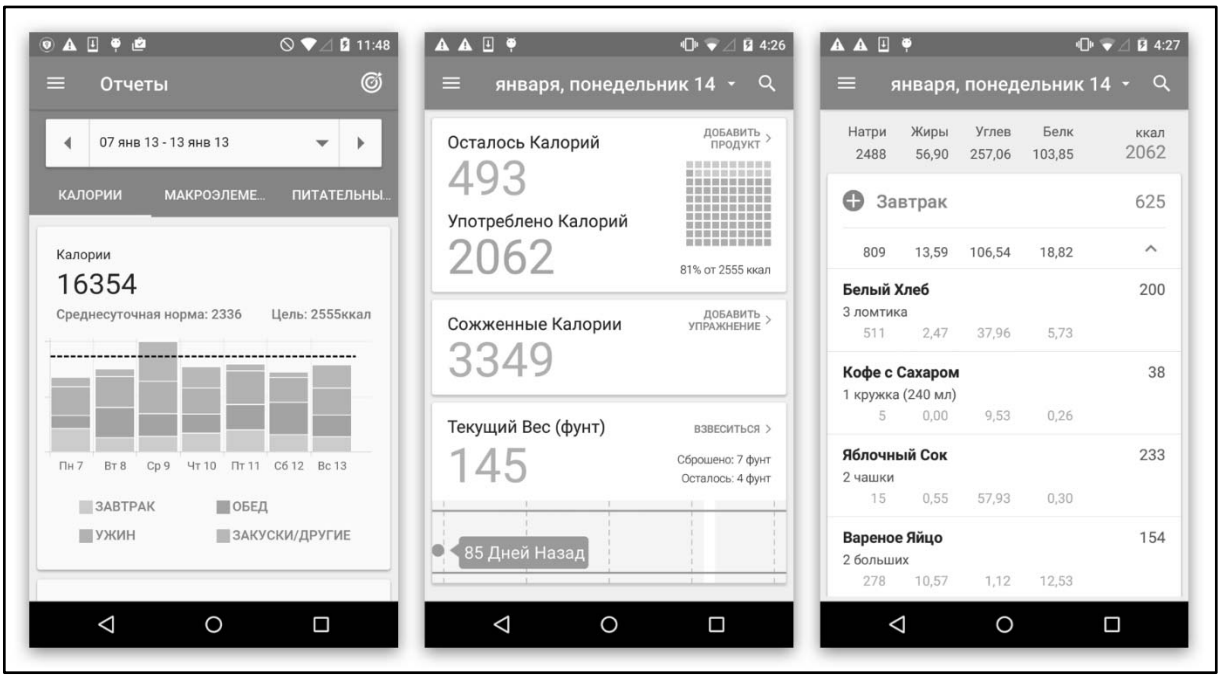

Рисунок 4 – Скриншоты приложения FatSecret

Пользователи приложения отмечают удобный и простой интерфейс как плюс, но недостаточный функционал и местами плохую оптимизацию в качестве недостатков.

Приложение Lifesum

Lifesum– это еще одно очень популярное приложение для подсчета калорий, имеющее привлекательный дизайн [5]. Страница приложения Lifesum в магазине Google Play представлена на рисунке 5.

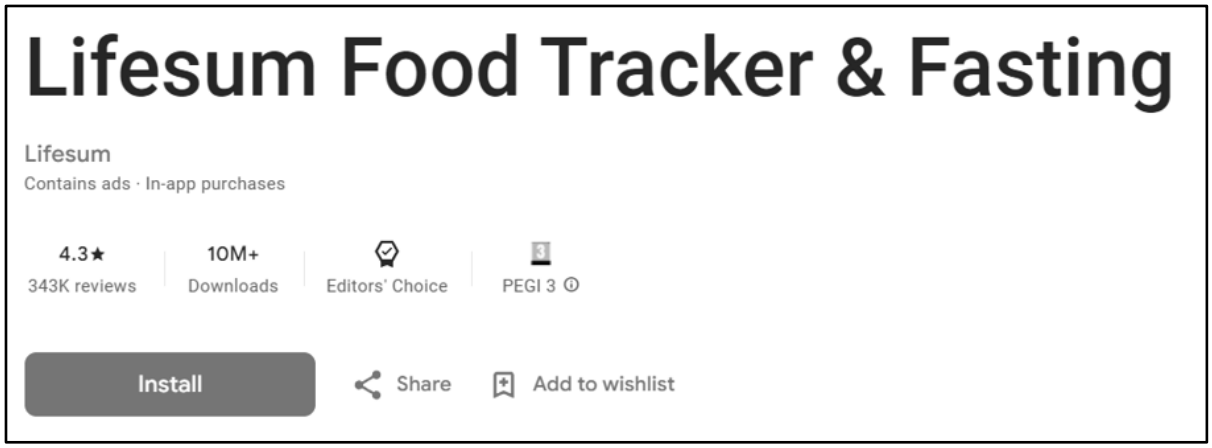

Рисунок 5 – Страница приложения Lifesum на Google Play

В программе большая база продуктов, возможность добавления рецептов с указанием порций и устройство для считывания штрих-кодов. Lifesum также запоминает, какие продукты были употреблены в пищу, и это дополнительно упрощает контроль питания. В приложении реализована удобная система напоминаний о ежедневном взвешивании, приемах пищи и употреблении воды.

Программа бесплатная, но можно приобрести премиум-аккаунт, в котором открывается доступ к дополнительной информации по продуктам (клетчатка, сахар, холестерин, натрий, калий), учет объемов тела и процент жировой ткани, рейтинг продуктов. Скриншоты приложения Lifesum представлены на рисунке 6.

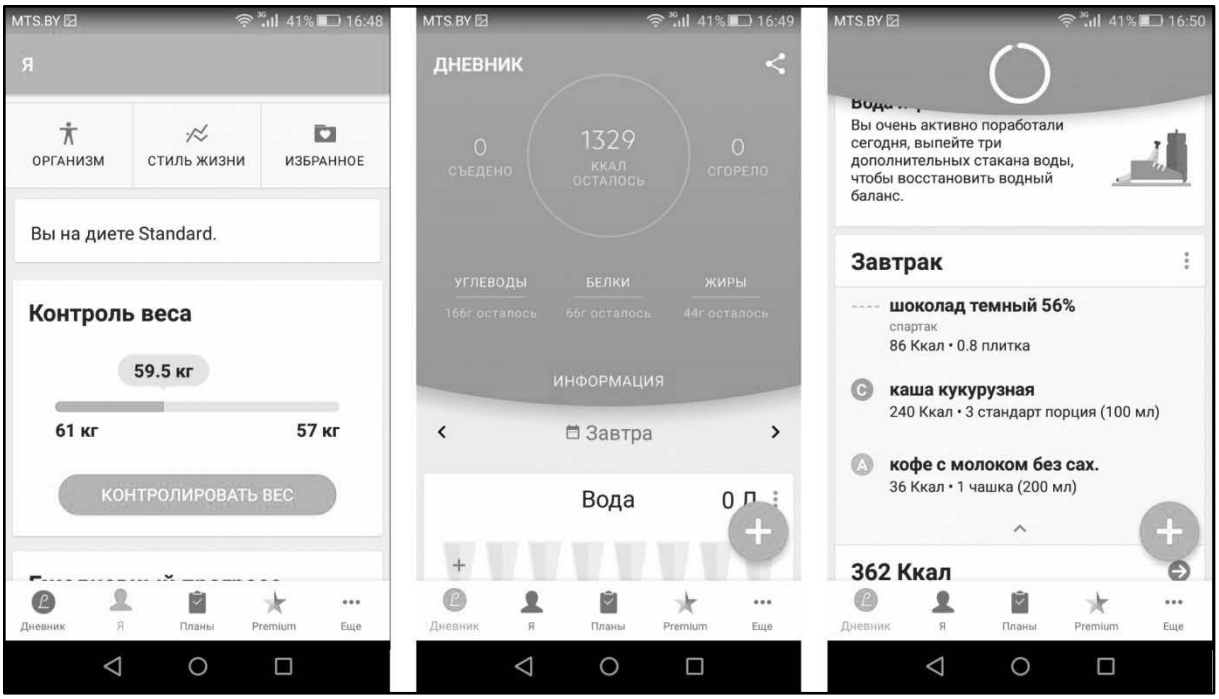

Рисунок 6 – Скриншоты приложения Lifesum

Пользователи приложения отмечают удобный интерфейс и широкий набор функций в качестве достоинства, но плохую оптимизацию в качестве основного недостатка.

1.2. Обзор технологий реализации проекта

Для разработки Android-приложений существует несколько сред и языков программирования. Некоторые из наиболее популярных сред разработки и языков программирования для Android-приложений представлены ниже.

1. Android Studio. Официальная среда разработки для Android-приложений, разработанная Google. Она основана на IntelliJ IDEA и предоставляет инструменты для создания, тестирования и отладки приложений для Android. Интегрирована поддержка языков Java и Kotlin.

2. Eclipse. Eclipse – это среда разработки на языке Java, которая также может использоваться для разработки приложений для Android. Она обла-

дает рядом плагинов, позволяющих разработчикам создавать Android-приложения. Тем не менее, Google официально прекратила поддержку Eclipse в 2015 году, что может привести к ограниченной функциональности.

3. Xamarin. Среда разработки на языке C#, которая позволяет разработчикам создавать кроссплатформенные приложения, в том числе для Android. Xamarin предоставляет доступ к набору библиотек .NET и обеспечивает интеграцию с Visual Studio.

4. React Native. Открытая среда разработки на языке JavaScript, позволяющая создавать кроссплатформенные приложения, в том числе для Android. Она предоставляет набор готовых компонентов и инструменты для создания пользовательских интерфейсов.

Языки программирования, наиболее часто используемые для разработки Android-приложений представлены ниже.

1. Java. Основной язык программирования для Android-приложений, который используется в Android Studio. Он является объектно-ориентированным языком, популярным и широко используется во всем мире.

2. Kotlin. Недавно появившийся язык программирования, который был разработан JetBrains в 2011 году и в 2017 году был объявлен Google основным языком программирования для Android-приложений. Kotlin является языком программирования на основе JVM, совместимым с Java.

3. C++. Язык программирования, используемый для создания высокопроизводительных приложений для Android. Он может использоваться для разработки нативных приложений для Android с помощью Android NDK.

Исходя из вышеперечисленного, наиболее подходящей средой разработки для Android-приложения под данную задачу будут являться: интегрированная среда разработки – Android Studio и статически типизированный объектно-ориентированный язык – Kotlin.

Причинами выбора этой IDE являются:

1) открытый исходный код;

2) большое сообщество;

3) интеграция с Android SDK, который включает в себя все необходимые инструменты для разработки приложений под Android;

4) наличие встроенных эмуляторов Android.

При разработке используется соответствующая документация [6].

Ниже перечислены причины, по которым для реализации приложения был выбран язык Kotlin.

1. Тот факт, что Kotlin является официальным языком разработки для Android, и имеет богатую документацию и поддержку со стороны разработчиков.

2. Kotlin имеет множество библиотек и фреймворков, которые могут быть использованы для разработки Android-приложений.

3. Kotlin является языком высокого уровня, который предоставляет разработчикам множество инструментов для управления памятью, сборки мусора и других задач, что позволяет сосредоточиться на разработке приложения, а не на тонкостях управления ресурсами.

При разработке также используется соответствующая документация [7].

Вывод по первой главе

В результате обзора существующих решений был выявлен следующий набор необходимых требований к приложению:

1) не перегруженный и интуитивно понятный интерфейс;

2) наличие базы данных с продуктами питания;

3) возможность добавлять продукты питания в базу приложения;

4) возможность ввести и изменить данные профиля;

5) возможность просмотреть статистику потребленных калорий.

Помимо этого, приложение должно выполнять свою главную функцию – подсчет калорий.

2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1. Анализ требований

Нефункциональные требования – требования, определяющие свойства, которые система должна демонстрировать, или ограничения, которые она должна соблюдать, не относящиеся к поведению системы [8]. Для реализации данного проекта были выделены следующие нефункциональные требования:

1) система должна быть написана на языке Kotlin;
2) система должна поддерживать базу данных из Firebase;
3) система должна поддерживать версию Android 5.00+;
4) система должна иметь графический интерфейс (GUI).

Функциональные требования – описание требуемого поведения системы в определенных условиях. Функциональные требования определяют, каким должно быть поведение продукта в тех или иных условиях [8]. Они определяют, что разработчики должны создать, чтобы пользователи смогли выполнить свои задачи в рамках требований. Для реализации данного проекта были выделены следующие функциональные требования, перечисленные ниже.

1. Система должна рассчитывать калории, содержащиеся в продукте из введенных данных пользователя о весе и калорийности продуктов.
2. Система должна предоставлять пользователю возможность анализировать свой прогресс в отношении достижения целей по потреблению калорий и предоставлять графики и диаграммы.
3. Система должна иметь список продуктов с указанием их калорийности, который может быть расширен пользователем.
4. Система должна позволять пользователю вводить свой вес, рост, возраст и уровень активности для расчета дневной нормы калорий.
5. Система должна позволять пользователю выбрать цель использования приложения, из которой будет строиться формула расчета калорий.

2.2. Диаграмма вариантов использования

В ходе анализа функциональных требований была построена диаграмма вариантов использования [8], представленная на рисунке 7.

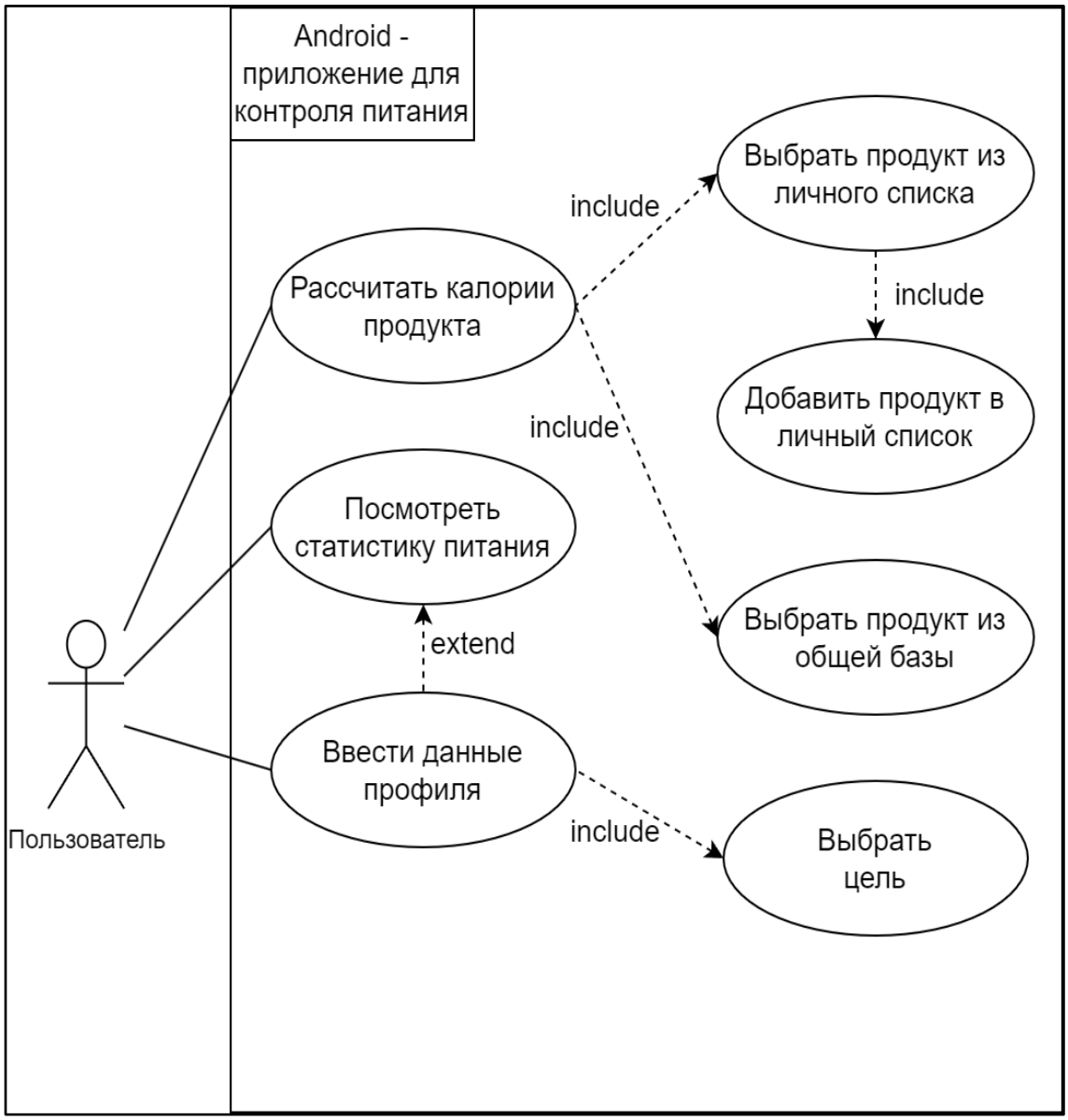

Рисунок 7 – UML диаграмма вариантов использования

Список ниже подробно описывают UML диаграмму вариантов использования приложения, в которой описаны возможные взаимодействия пользователя с приложением.

1. Рассчитать пищевую и энергетическую ценность продукта – пользователь может рассчитать калории приема пищи, для этого он может либо

выбрать продукт из личного списка, который заполняется самим пользователем путем ввода данных о продуктах (КЖБУ – калории, жиры, белки, углеводы); либо ввести вес и выбрать нужный продукт из списка продуктов; пользователю необходимо выбрать продукт и ввести данные о потребляемом продукте. Помимо этого, пользователь получит пищевую ценность продукта, то есть содержание и соотношение белков, жиров и углеводов, содержащихся в продукте [9].

2. Просмотреть статистику питания – пользователь может ознакомиться со статистикой за определенный период (неделя / месяц / 3 месяца).

3. Ввести антропометрические данные – пользователь может ввести/обновить свои данные, такие, как: вес, рост, пол, возраст, для получения рекомендаций и советов для улучшения рациона, кроме того, выполнение этого пункта сделает статистические данные более точными.

4. Выбрать цель – выбрать цель использования приложения (например, похудение) для того, чтобы получить более точные рекомендации и подробные данные статистики, адаптированные к индивидуальным потребностям и желаемым результатам.

2.3. Дизайн приложения

В мире современных мобильных технологий дизайн играет решающую роль в опыте пользователя. Создание эффективного дизайна для Android-приложений требует не только креативности, но и глубокого понимания принципов пользовательского интерфейса (UI) и пользовательского опыта (UX). В данном разделе представлены ключевые аспекты дизайна Android-приложения.

Дизайн – это процесс планирования и создания объекта, системы, компонента или структуры, который представляет собой концепцию или конечный результат (как план, продукт или композиция), который преобразуется в реальность [10].

В разрабатываемом Android – приложении для контроля питания содержится шесть основных компонентов, представленные через «Activity»: «Профиль», «Калькулятор», «Статистика», «Вход», «Список продуктов», а также компонент, присутствующий на всех разделах приложения – «Меню». Ниже приведет подробный разбор данных компонентов.

Активность или «Activity» – это основной компонент Android-приложения, представляющий собой экран с пользовательским интерфейсом, где происходит взаимодействие пользователя с приложением. Каждая активность обычно соответствует одному экрану приложения, и приложение может содержать несколько активностей для обеспечения различных функциональных возможностей и переходов между различными частями приложения. Жизненный цикл активности включает стадии создания (onCreate), запуска (onStart), восстановления (onResume), паузы (onPause), остановки (onStop) и уничтожения (onDestroy), что позволяет управлять тем, что происходит на экране, например, сохранением состояния или освобождением ресурсов, когда активность больше не видна пользователю. Также активность управляет пользовательским интерфейсом и обрабатывает взаимодействие с ним, включая нажатия на кнопки, ввод данных, прокрутку списков и другие действия, используя слушатели событий.

Профиль

Эта активность предназначена для управления пользовательским профилем. В данном компоненте пользователи могут изменять свои биометрические данные, устанавливать цели и коэффициент активности, просматривать персональную информацию. На рисунке 8 представлены скриншоты дизайна данного компонента, включающего экраны для просмотра и редактирования личной информации.

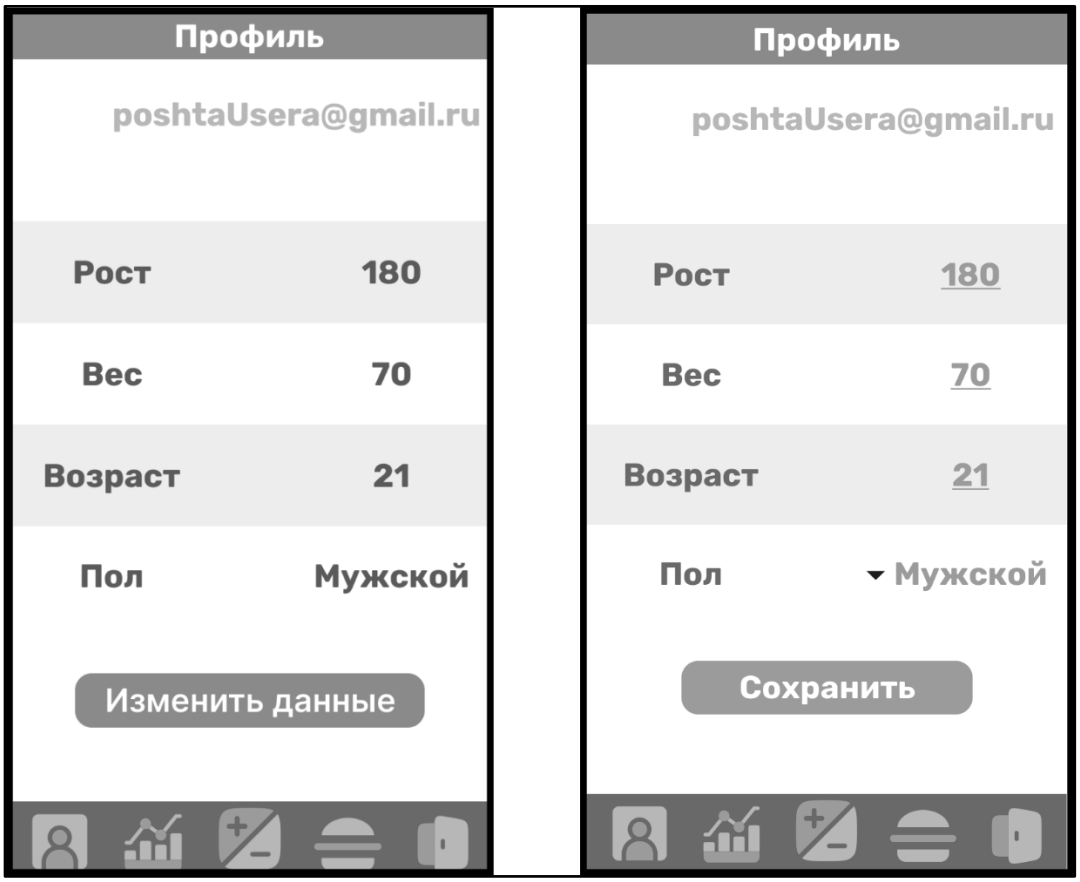

Рисунок 8 – Дизайн компонента «Профиль»

На первом скриншоте показано первичное состояние компонента, когда пользователь только зашел в данный раздел. Пользователь может проверить свои данные, или, нажав на кнопку «Изменить данные», перейти во второе состояние. Во втором состоянии пользователь может внести изменения в личные данные.

Калькулятор

В этом разделе пользователи могут использовать функционал калькулятора для выполнения расчета КЖБУ приема пищи. На рисунке 9 представлены скриншоты дизайна данного компонента, которые демонстрируют интерфейс калькулятора.

Рисунок 9 – Дизайн компонента «Калькулятор»

На первом скриншоте показано первичное состояние компонента. Данное состояние состоит из двух разделов. Пользователь может увидеть норму калорий за текущий день в первом разделе, или рассчитать КЖБУ приема пищи во втором разделе.

Для того, чтобы рассчитать прием пищи во втором разделе, пользователю необходимо нажать на кнопку «+» для того, чтобы добавить блюдо в список. В списке представлены все добавленный блюда на данный прием пищи. После добавления всех продуктов, пользователю необходимо нажать на кнопку «Рассчитать». Результаты расчета будут внесены в статистику пользователя.

Второе состояние компонента активируется при нажатии на кнопку «+». В нем пользователь может внести данные о продукте и нажать на кнопку «Добавить свой продукт», чтобы добавить в список калькулятора

продукт с введенными пользователем значениями. Также пользователь может нажать на кнопку «Выбрать из списка» для того, чтобы выбрать продукт из облачной базы продуктов.

Статистика

В этом разделе пользователи могут просмотреть свою статистику питания. На рисунке 10 представлен скриншот дизайна данного компонента.

Рисунок 10 – Дизайн компонента «Статистика»

Компонент состоит из двух разделов: первый включает прогресс-бар для отслеживания дневного потребления калорий и круговую диаграмму для отображения соотношения потребленных жиров, белков и углеводов (КЖБУ), второй раздел содержит диаграмму потребленных калорий за период, выбранный пользователем с помощью выпадающего списка «Выберите период».

Список продуктов

В этом компоненте пользователь может увидеть список продуктов и добавить продукт в список калькулятора. На рисунке 11 представлен скриншот дизайна данного компонента.

Рисунок 11 – Дизайн компонента «Список продуктов»

Компонент состоит из двух разделов. В первом разделе представлен список всех продуктов, находящихся в облачной базе данных Firebase данного приложения. Во втором разделе представлены характеристики выбранного пользователем продукта, а также кнопка «добавить», при нажатии на которую продукт добавится в список калькулятора.

Вход

Данный компонент появляется при первом входе пользователя в приложение, или при повторном входе, после выхода из аккаунта. Компонент позволяет пользователю войти в аккаунт через сервисы авторизации и

аутентификации пользователей Google Authentication и Firebase Authentication. На рисунке 12 представлен скриншот дизайна данного компонента.

Рисунок 12 – Дизайн компонента «Вход»

Данный компонент предоставляет пользователю возможность авторизации с помощью Google. Он включает одну кнопку с надписью «Войти с помощью Google». При нажатии на эту кнопку открывается окно, в котором пользователь может выбрать аккаунт Google для привязки его к своему профилю в приложении. Если пользователь еще не привязал свой аккаунт Google, он может выбрать нужный аккаунт из списка доступных или ввести данные нового аккаунта. В случае, если аккаунт уже привязан, нажатие на кнопку приведет к автоматическому входу в личный профиль пользователя. Этот механизм обеспечивает удобный и быстрый способ авторизации, исключая необходимость ввода логина и пароля, и повышает безопасность, используя двухфакторную аутентификацию Google.

Меню

Данный компонент не является активностью, как предыдущие. Он присутствует на всех предыдущих компонентах, кроме «Вход» и позволяет перемещаться из одного компонента в другой. На рисунке 13 представлен скриншот дизайна компонента «Меню».

Рисунок 13 – Дизайн компонента «Меню»

В данном компоненте находится 5 символических кнопок, направляющих пользователя в соответствующие активности. Их значения слева направо: «Профиль» – переводит пользователя в активность «Профиль», «Статистика» – переводит пользователя в активность «Статистика», «Калькулятор» – переводит пользователя в активность «Калькулятор», «Список продуктов» – переводит пользователя в активность «Список продуктов», «Выход» – осуществляет выход из аккаунта приложения.

Вывод по второй главе

В ходе тщательной проработки второй главы были определены как функциональные, так и нефункциональные требования к разрабатываемому приложению. Также одним из важных результатов этого этапа стало составление диаграммы вариантов использования, которая позволила наглядно представить все возможные сценарии взаимодействия пользователя с приложением. Параллельно с этим был разработан макет дизайна приложения, который отражал не только его внешний вид, но и структуру пользовательского интерфейса. Важным аспектом здесь было обеспечение легкости восприятия информации, интуитивно понятного навигационного меню и привлекательного дизайна, способного привлечь внимание пользователей.

3. ПРОЕКТИРОВАНИЕ ХРАНИЛИЩА ДАННЫХ

3.1. Обзор БД для платформы Android

База данных (БД) – это организованное хранилище данных, которое используется для хранения, организации, управления и обновления больших объемов информации. Она может быть использована для хранения данных любых типов, от простых текстовых документов до сложных мультимедийных файлов [11].

БД для приложений на Android можно разделить на две категории: локальные и облачные. Для данного приложения был выбран облачный тип БД. К облачным СУБД относятся Firebase, Amazon Web Services (AWS) и Microsoft Azure. Ниже представлен обзор этих СУБД.

Amazon Web Services (AWS). Это облачная платформа, включающая в себя большое количество различных сервисов, включая СУБД. AWS предлагает несколько СУБД на выбор, таких как Amazon Aurora, Amazon RDS, Amazon DynamoDB, Amazon ElastiCache и Amazon Neptune. Все они предоставляют высокую производительность и масштабируемость, а также хранение и обработку данных в облачной среде [12].

Microsoft Azure. Это облачная платформа от Microsoft, которая также предоставляет множество возможностей, включая СУБД. В Azure доступны СУБД Microsoft SQL Server и Azure Cosmos DB. Они обеспечивают масштабируемость, высокую производительность и доступность данных в любой точке мира [13].

Firebase. Это облачная платформа от Google, которая обеспечивает не только хранение и обработку данных, но и другие функции, такие как аутентификация, аналитика и уведомления. Firebase использует СУБД Cloud Firestore, которая является поддерживаемой документ-ориентированной СУБД. Она обеспечивает гибкость и масштабируемость, а также более простое управление данными в реальном времени [14].

Из перечисленных БД наиболее подходящей для приложений на Android является Firebase, так как она предоставляет широкий набор инструментов для разработки мобильных приложений и хранения данных в облаке, а также обладает удобным API для работы с ней из приложения. Для работы с Firebase будет использоваться соответствующая документация [15].

3.2. Модель базы данных

Для хранения данных приложения используется облачная NoSQL база данных Firebase Realtime Database, которая сохраняет информацию в формате JSON. Поэтому для наглядной и удобной визуализации базы данных ее можно представить в виде дерева. Данная база данных разделена на две основные ветви: `Products`, которая содержит информацию о продуктах, и `Users`, которая хранит данные о пользователях.

Ветка `Products` представлена на рисунке 14.

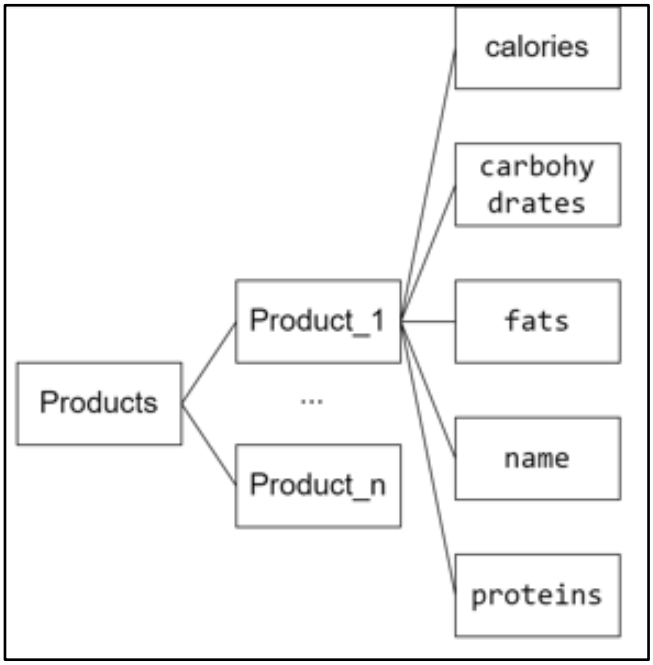

Рисунок 14 – Схема ветви `Products`

Эта ветка содержит в себе n продуктов. Каждый продукт имеет следующие поля: `calories` (калории), `carbohydrates` (углеводы), `fats` (жиры), `name` (наименование), `proteins` (белки).

Ветка `Users` представлена на рисунке 15.

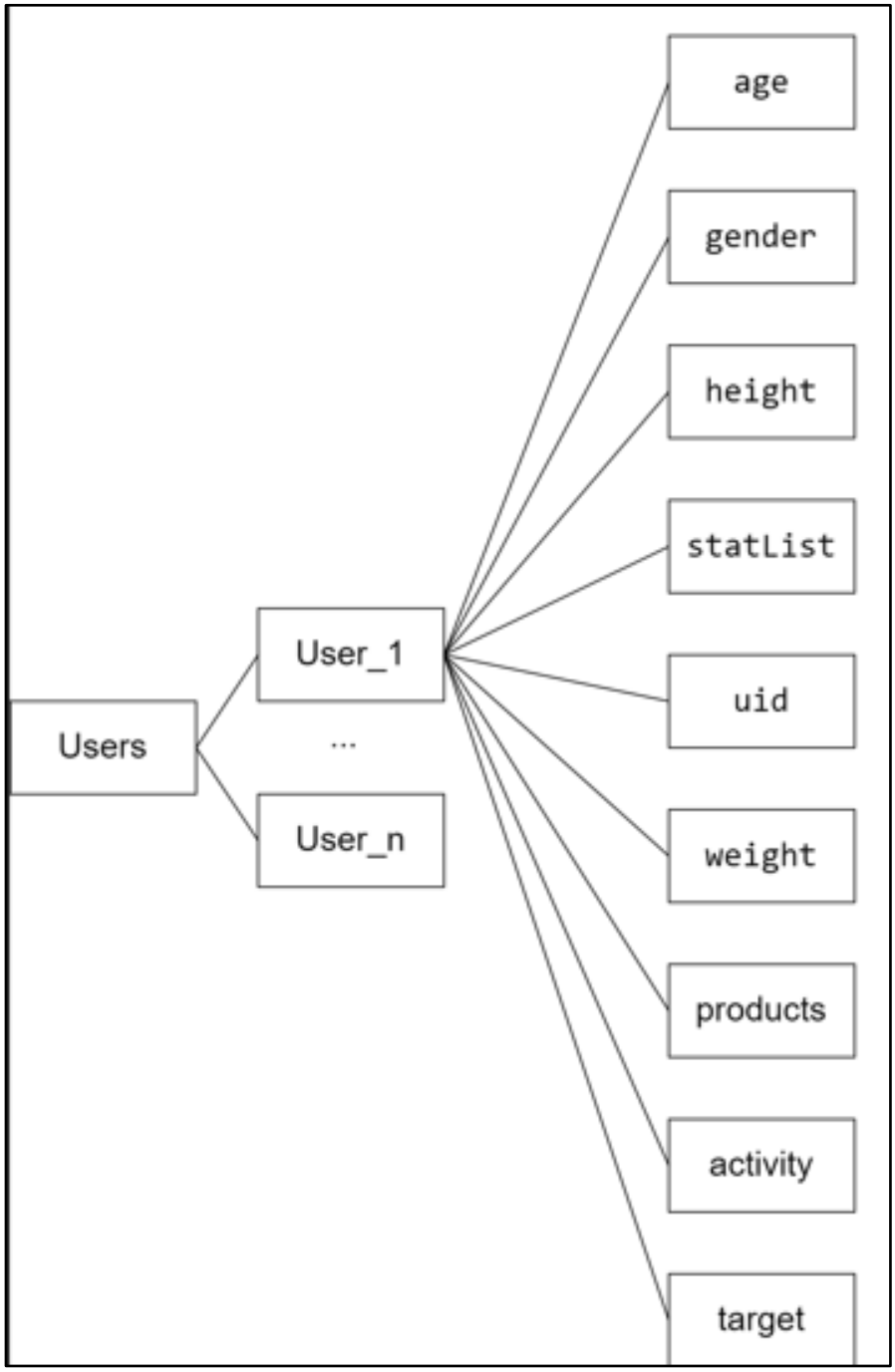

Рисунок 15 – Схема ветви `Users`

Ветка `Users` содержит в себе n пользователей. Каждый пользователь имеет следующие поля: `age` (возраст), `gender` (пол), `height` (рост), `uid` (уникальный идентификатор), `weight` (вес), ветвь `products` (список личных продуктов), схема которого аналогична схеме ветви `Products`, `activity`

(коэффициент активности), `target` (цель), а также ветвь `statList`, которая рассмотрена отдельно ниже.

Схема ветви `statList` представлена на рисунке 16.

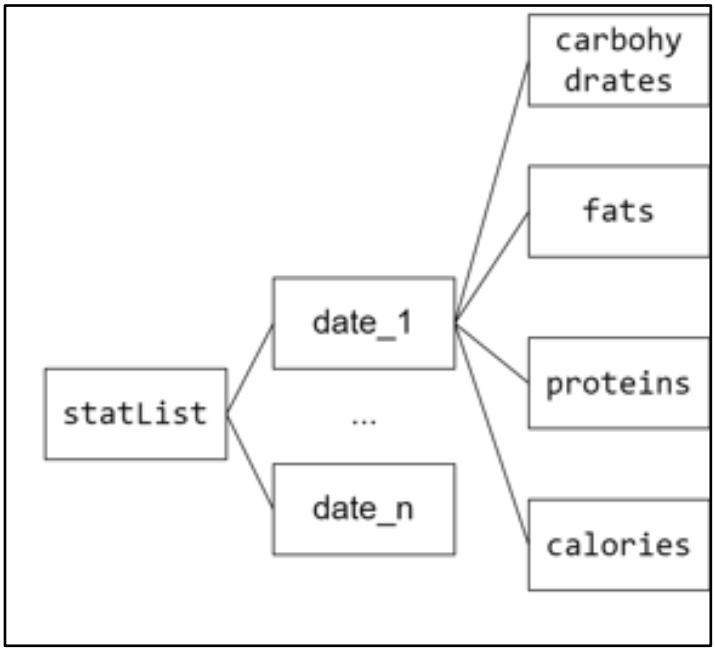

Рисунок 16 – Схема ветви `statList`

Данная ветвь содержит n дневных статистических записей, которые состоят из полей: `calories` (калории), `carbohydrates` (углеводы), `fats` (жиры), `proteins` (белки).

Вывод по третьей главе

В ходе работы над третьей главной был проведён обзор БД для Andriod-приложения для контроля питания и выбрана облачная база данных Firebase. Также была разработана модель хранения данных в выбранной БД.

4. АРХИТЕКТУРА СИСТЕМЫ

Приложение будет строится на базе архитектуры MVC (ModelView-Controller). MVC состоит из модели – это объект приложения, содержащий данные приложения и «бизнес-логику», представление, отображающие данные на экран пользователя и контроллер, реагирующий на действия пользователя, оповещая модель о необходимости изменений [16]. Общий принцип работы архитектуры MVC изображен на рисунке 17.

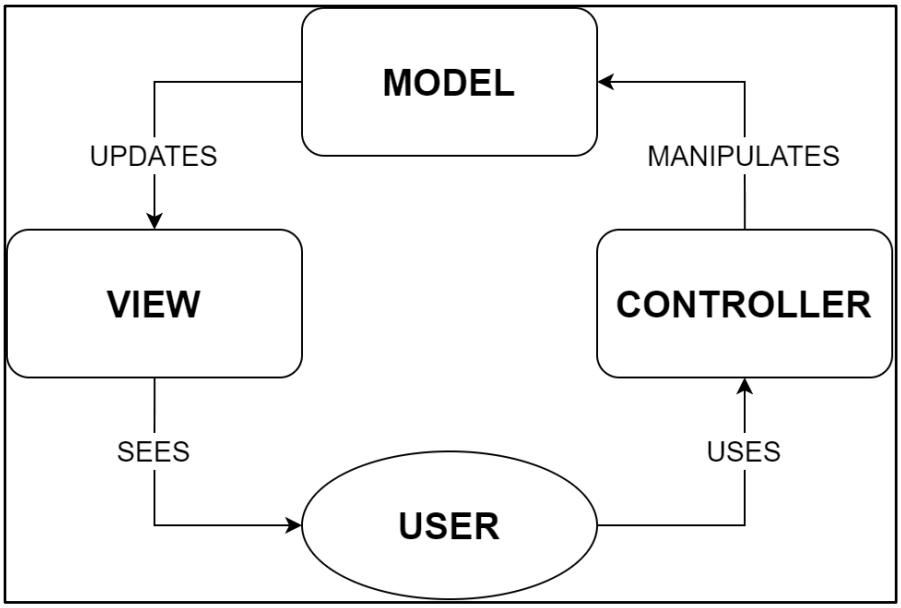

Рисунок 17 – Принцип работы архитектуры MVC

MVC предоставляет значительные преимущества при разработке приложений. При добавлении нового функционала в приложение, разработчику необходимо всего лишь сосредоточиться на модели, что упрощает задачу в целом. Кроме того, использование MVC способствует повторному использованию кода, поскольку классы с узкой функциональностью более подходят для повторного использования, чем классы, которые пытаются выполнять сразу много задач.

Подробнее о элементах MVC описано ниже.

1. Модель (Model) – это компонент, который представляет данные и бизнес-логику приложения. Это место, где данные обрабатываются, хра-

нятся и обновляются. Этот компонент ничего не знает о других компонентах, которые имеются в приложении, таких как представления и контроллеры.

2. Представление (View) – это компонент, который предоставляет пользовательский интерфейс и отображает данные модели в удобном для пользователя виде. Этот компонент получает данные из модели и отображает их. Он не может напрямую изменять данные модели.

3. Контроллер (Controller) – это компонент, который управляет взаимодействием между моделью и представлением. Контроллер получает запросы от пользователей, обрабатывает их и, если необходимо, обновляет модель. Он также обновляет данные, отображаемые в представлении, чтобы отразить изменения в модели.

Диаграмма основных компонентов приложения представлена на рисунке 18.

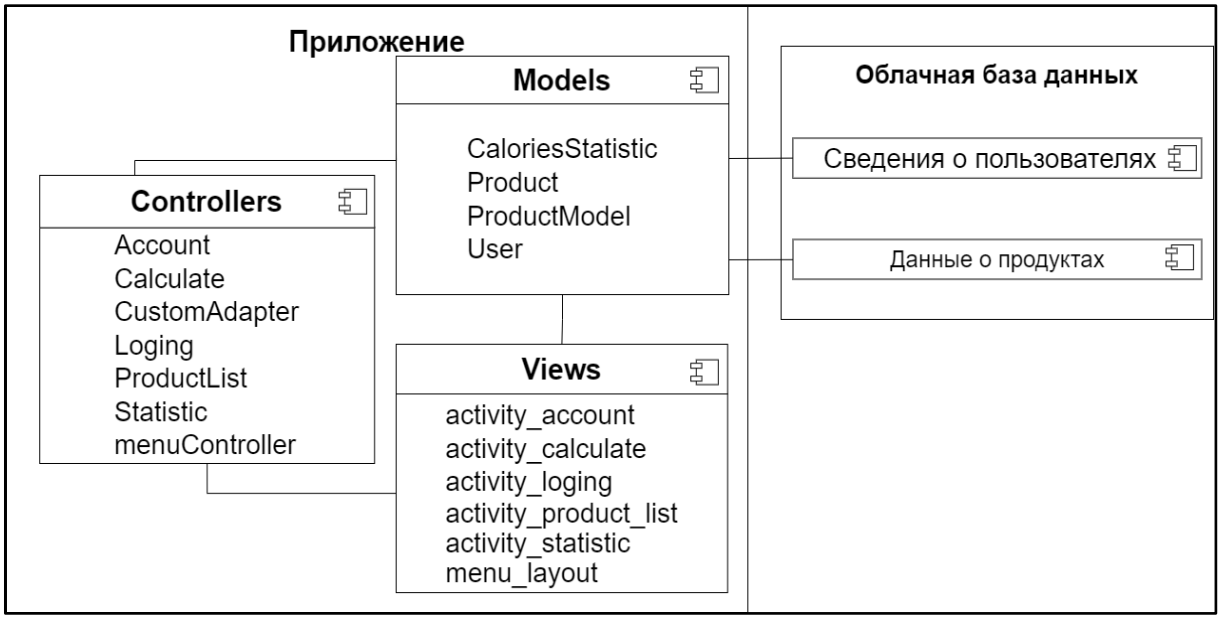

Рисунок 18 – Диаграмма компонентов системы

Ниже представлены основные элементы приложения.

Модели (Models)

Были выделены восемь моделей приложения:

1) `CaloriesStatistic` – модель хранения статистики;

2) `Product` – модель для хранения информации о продукте;

3) `User` – модель для хранения информации о пользователе;

4) `ProductModel` – модель, отвечающая за загрузку данных из базы данных и их преобразование в объекты `Product`.

Представления (Views)

Были выделены пять представлений для приложения:

1) `activity_account` – отображает данные модели `Account`;

2) `activity_calculate` – отображает данные модели `Calculate`;

3) `activity_loging` – отображает данные модели `Loging`;

4) `activity_statistic` – отображает данные модели `CaloriesStatistic`;

5) `activity_product_list` – отображает данные модели `ProductList`;

6) `menu_layout` – отображает меню приложения.

Контроллеры (Controllers)

Были выделены пять контроллеров для приложения:

1) `Calculate` – управляет взаимодействием между моделями `User` и `CaloriesStatistic` и `CaloriesStatistic` и представлением `activity_calculate`;

2) `Loging` – управляет взаимодействием между моделью `User` и представлением `activity_loging`;

3) `ProductList` – управляет взаимодействием между моделью `Product` и `ProductModel` и представлением `activity_product_list`;

4) `menuController` – управляет представлением `menu_layout`;

5) `Statistic` – управляет взаимодействием между моделью `CaloriesStatistic` и представлением `activity_statistic`;

6) `CustomAdapter` – управляет списками в представлениях `activity_calculate` и `activity_product_list`;

7) `Account` – управляет взаимодействием между моделью `User` и представлением `activity_account`.

Вывод по четвертой главе

В результате работы над четвертой главой была разработана и представлена архитектура системы, основанная на модели MVC (Model-View-Controller). Эта архитектура позволяет эффективно организовать структуру приложения, разделяя его на три основных компонента: модель, представление и контроллер.

Были выделены и описаны основные компоненты системы, включая модели, представления и контроллеры. Каждый компонент был подробно рассмотрен с указанием его функциональности и взаимодействия с другими компонентами. Это позволило четко определить ответственности каждого элемента системы и обеспечить их взаимодействие в соответствии с требованиями проекта.

В целом, разработанная архитектура системы обеспечивает удобство сопровождения и расширения приложения в будущем. Каждый компонент системы явно определен, что упрощает добавление нового функционала и модификацию существующего. Таким образом, предложенная архитектура является основой для дальнейшей разработки и реализации системы, соответствующей поставленным требованиям и ожиданиям пользователей.

5. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

5.1. Реализация облачной базы данных

В разделе 3.1 обзора БД для платформы Android была выбрана облачная СУБД Firebase для хранения данных приложения. Для хранения текстовых данных использовался сервис Realtime Database, который позволяет обновлять данные в режиме реального времени на всех устройствах, использующих приложение. Кроме того, для хранения данных пользователей, которые авторизовались или зарегистрировались в приложении, использовался сервис Authentication. Firebase Authentication упрощает быструю авторизацию через сервисы Google.

Примеры хранения данных

Хранение данных о пользователях с помощью облачной базы данных Firebase Realtime Database представлено на рисунке 19.

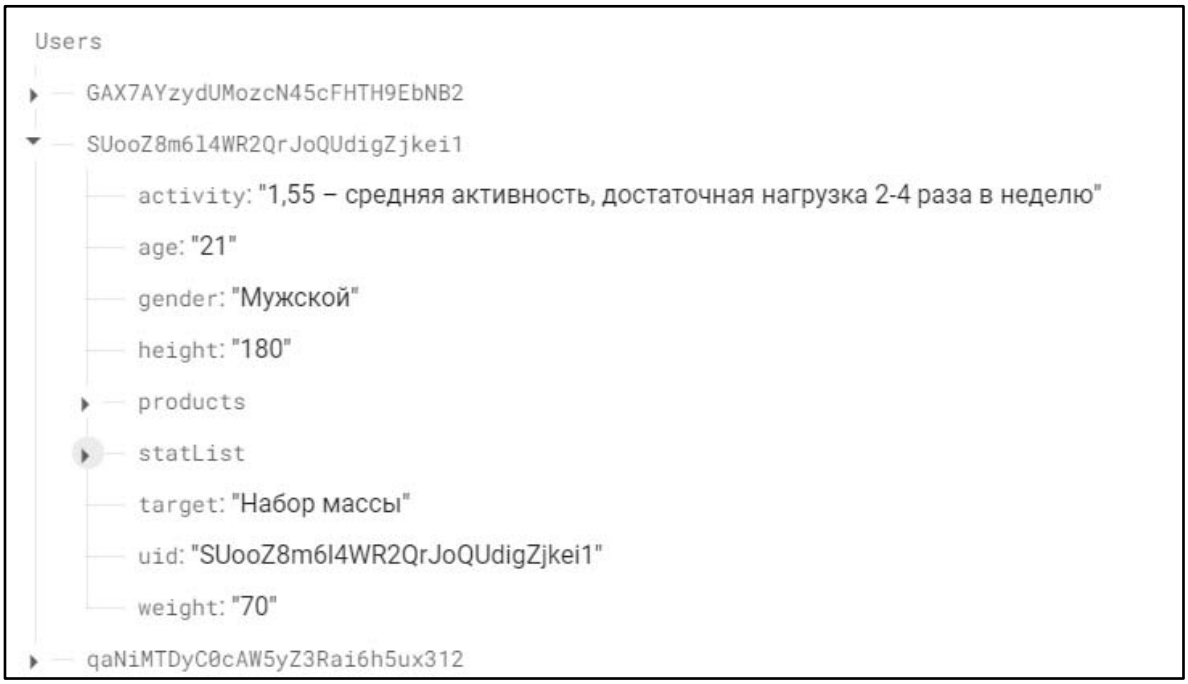

Рисунок 19 – Хранения данных о пользователях

Хранение данных о продуктах с помощью облачной базы данных Firebase Realtime Database представлено на рисунке 20.

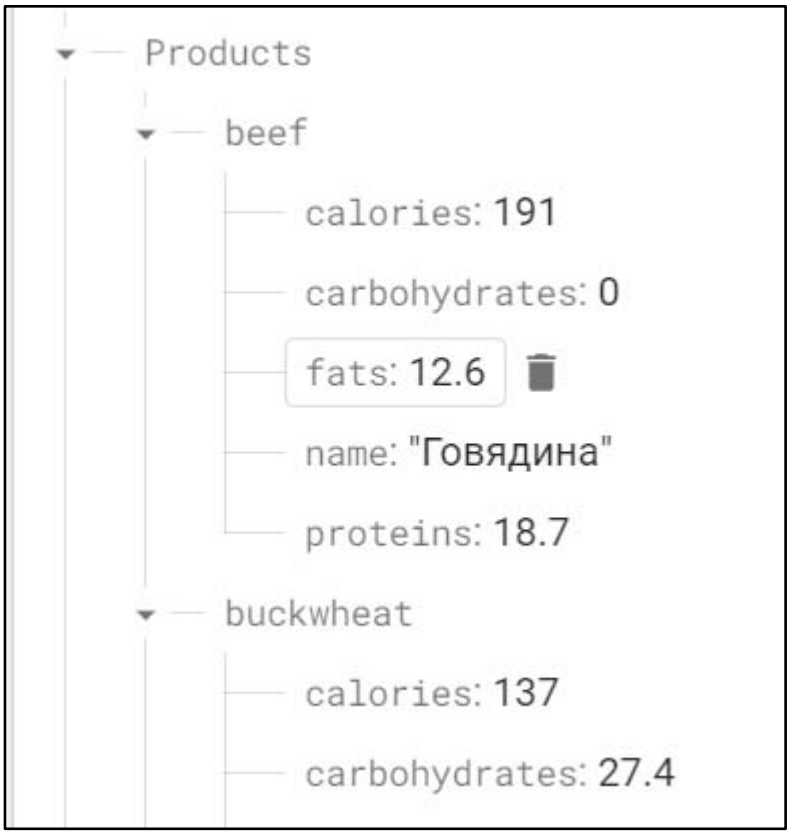

Рисунок 20 – Хранение данных о продуктах

На рисунке 21 показано, как с помощью Firebase Authentication хранятся данные об авторизованных через Google пользователях.

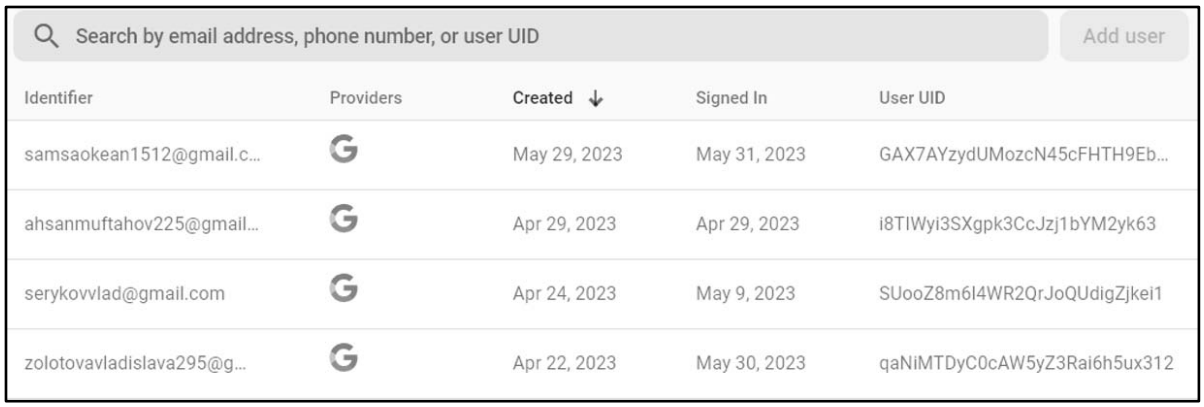

Рисунок 21 – Хранение данных аккаунтов

5.2. Реализация мобильного приложения

Чтобы использовать мобильное приложение, необходимо подключение к интернету, поскольку оно загружает данные из облачной базы данных

Firebase. Activity (активность) является одним из основных компонентов для Android-приложений.

В ходе реализации приложения были разработаны шаблонные решения для взаимодействия с облачной базой данных. На рисунке 22 представлена диаграмма деятельности Android – приложения для записи данных профиля в базу данных.

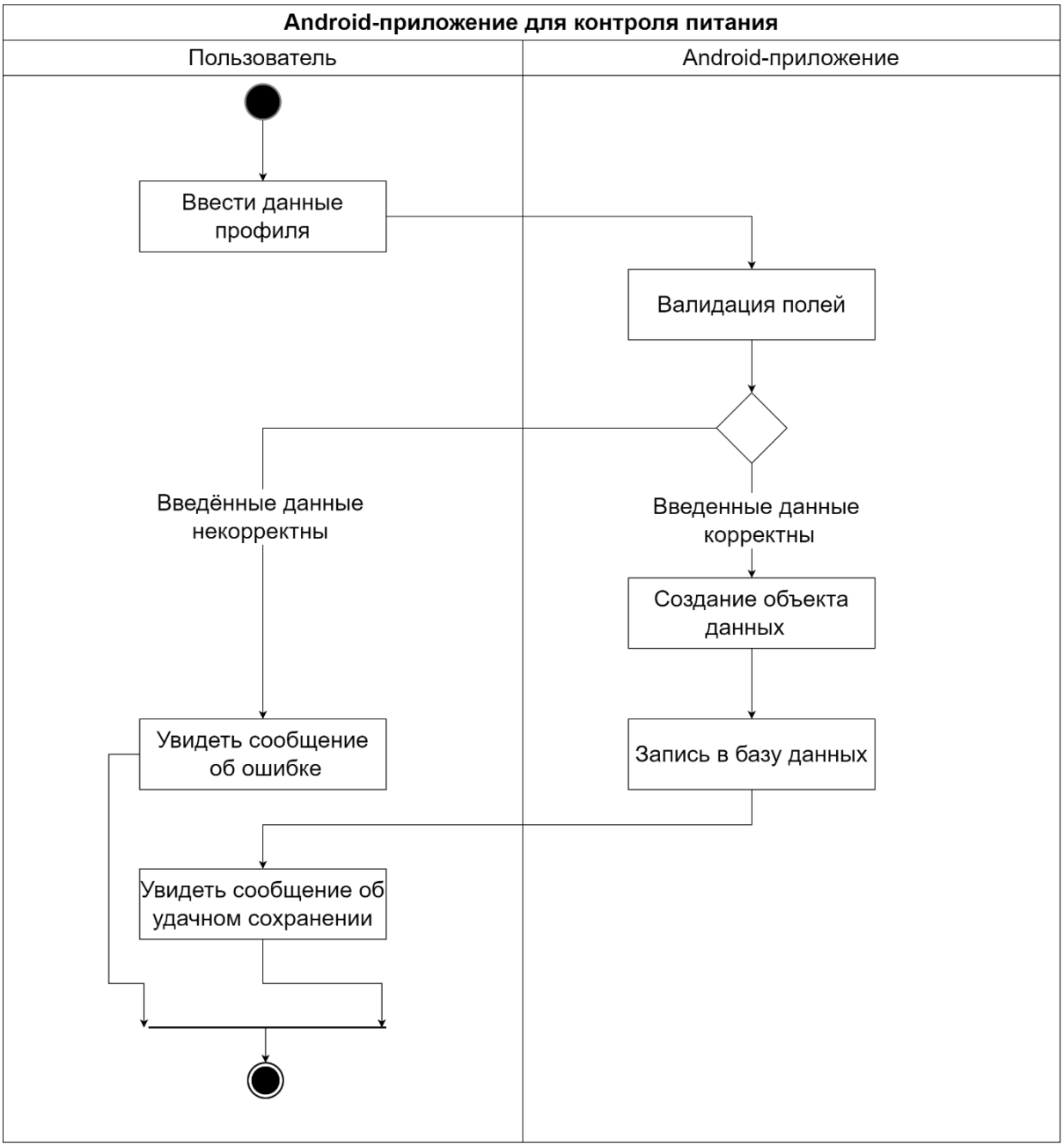

Рисунок 22 – Ввод данных профиля

При введении пользователем данных в форму и подтверждении отправки данных, приложение проводит валидацию полей, и при удачном прохождении проверок создает объект данных, который впоследствии отправляется в облачную базу данных. При удачной отправке данных пользователь увидит сообщении об удачном сохранении данных. При неудачном прохождении валидации пользователь получить сообщение об ошибке. В листинге 1 представлен код метода `addBiometricToBase`, который дополнительно объясняет диаграмму деятельности ввода данных профиля.

Листинг 1 – Метод `addBiometricToBase` в классе `Account`

```
private fun addBiometricToBase(age: EditText, weight: EditText, height:
EditText, gender: Spinner, target: Spinner, activity: Spinner): Boolean {
    if (age.text.toString() == "" || height.text.toString() == "" ||
weight.text.toString() == "") {
        Toast.makeText(this, "Остались пустые поля",
Toast.LENGTH_LONG).show()
        return false
    }

    val updates = hashMapOf<String, Any>(
        "age" to age.text.toString(),
        "gender" to gender.selectedItem.toString(),
        "target" to target.selectedItem.toString(),
        "activity" to activity.selectedItem.toString(),
        "height" to height.text.toString(),
        "weight" to weight.text.toString(),
        "uid" to Loging.signedInAccountId!!
    )

    val currentUser = mAuth!!.currentUser
    currentUser?.let {
        users!!.child(it.uid).updateChildren(updates)
    }

    return true
}
```

Метод `addBiometricToBase` принимает биометрические данные, введенные пользователем, и сохраняет их в БД Firebase.

На рисунке 23 представлена диаграмма деятельности Android-приложения для считывания данных из базы данных.

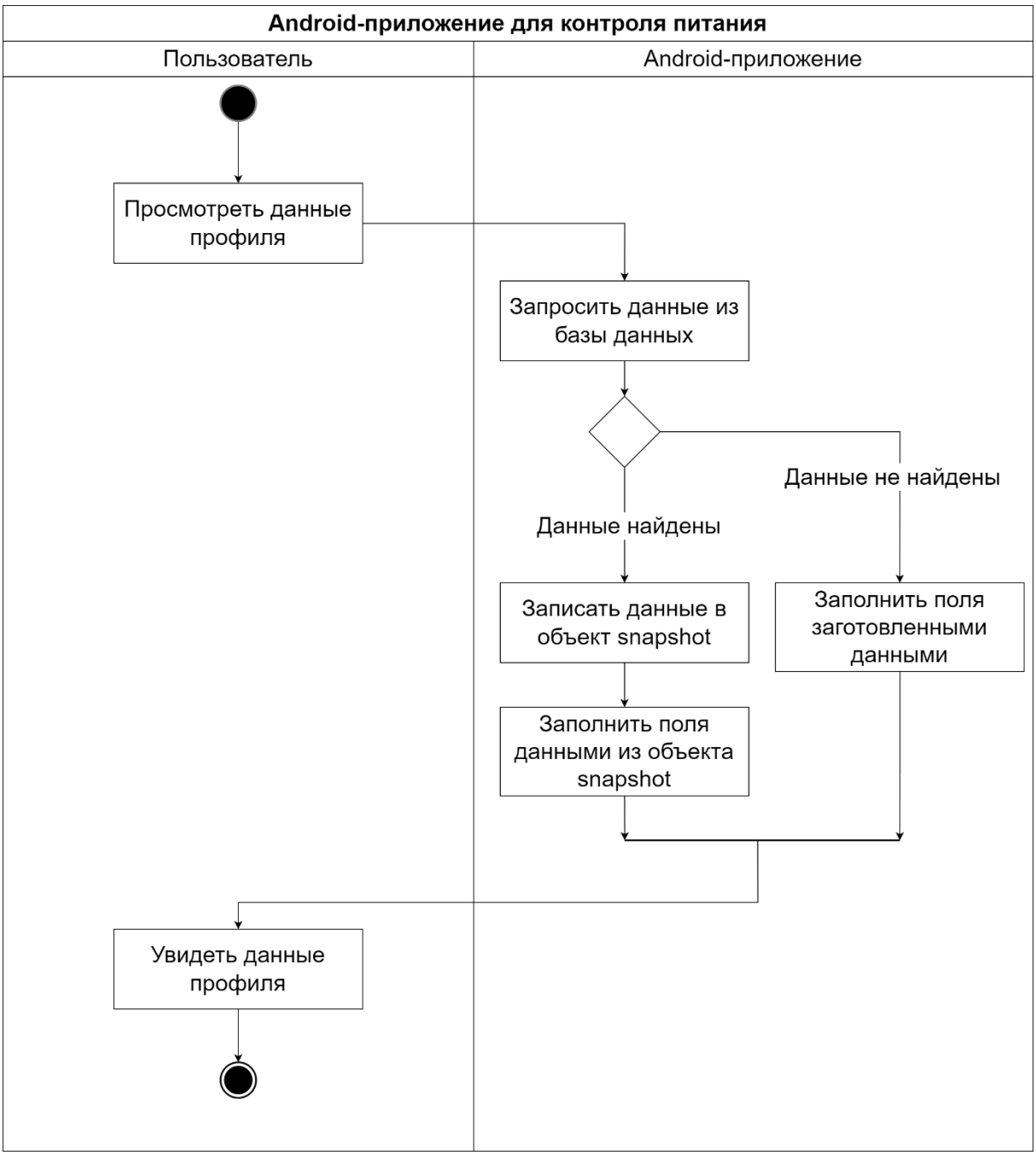

Рисунок 23 – Загрузка данных профиля

При заходе пользователем в окне «Профиль» приложение запрашивает данные пользователя из базы данных. В случае, если данные найдены, создается объект snapshot базы данных, после чего из этого объекта данные записываются в поля представления. Если данные не найдены, поля будут заполнены заготовленными стандартными значениями. В листинге 1 приложения Б представлен код метода `onCreate`, который дополнительно объясняет диаграмму деятельности загрузки данных профиля.

Метод `onCreate` определяет основные переменные и содержит переопределенный метод `onDataChange`, который считывает данные с базы данных и выводит их в окно приложения.

Данные примеры описывают шаблонные решения в приложении и все остальное взаимодействие приложения и базы данных построено по тем же принципам.

В листинге 2 приложения Б представлен класс `Loging`, который содержит методы: `firebaseAuthWithGoogle` – для аутентификации в БД Firebase с помощью аккаунта Google; `signIn` для входа в аккаунт; `singOut` для выхода из аккаунта.

В листинге 2 предоставлен класс `MenuController` с методом перехода на активности приложения при нажатии на соответствующие кнопки.

Листинг 2 – Класс главного окна приложения `MenuController`

```
class MenuController(private val context: Context) {
    fun onMenuItemClicked(menuItem: String) {
        when (menuItem) {
            "calculate" -> {
                val intent = Intent(context, Calculate::class.java)
                context.startActivity(intent)
            }
            "account" -> {
                val intent = Intent(context, Account::class.java)
                context.startActivity(intent)
            }
            "productList" -> {
                val intent = Intent(context, ProductList::class.java)
                context.startActivity(intent)
            }
            "statistic" -> {
                val intent = Intent(context, Statistic::class.java)
                context.startActivity(intent)
            }
            "signOut" -> {
                var loging = Loging()
                loging.singOut()
                val intent = Intent(context, Loging::class.java)
                context.startActivity(intent)
            }
        }
    }
}
```

Для расчета нормы калорий пользователя были выбраны формулы (1, 2) Миффлина-Сан Жеора [17]:

$$K_m = (9.99 * W + 6.25 * H - 4.92 * Ag + 5) * kfa, \quad (1)$$

$$K_w = (9.99 * W + 6.25 * H - 4.92 * Ag - 161) * kfa, \quad (2)$$

где K_m – норма калорий для мужчин;

K_w – норма калорий для женщин;

W – вес;

H – рост;

Ag – возраст;

kfa – коэффициент активности.

В листингах 3 и 4 представлены основные метода класса `Calculate`: `calculateNorm` и `calculate`.

Листинг 3 – Метод `calculateNorm` в классе `Calculate`

```
fun calculateNorm(thisUser: User): Double {
    var dailyCalories: Double?
    dailyCalories = 2500.0
    val activityString = thisUser.activity?.toString()?.replace(",", ".")
    val activity = if (!activityString.isNullOrEmpty() && activityString != 
"null") {
        try {
            activityString.toFloat()
        } catch (e: NumberFormatException) {
            // Логируем ошибку и устанавливаем значение по умолчанию
            Log.e("CalculateNorm", "Некорректный формат числа для activity: 
$activityString")
            1.375f // Значение по умолчанию в случае исключения
        }
    } else {
        1.375f // Значение по умолчанию, если строка пуста или равна "null"
    }
    if (thisUser.uid != null) {
        if (thisUser.gender == "Мужской") {
            dailyCalories = (thisUser.height!!.toFloat() * 6.25 - this-
User.age!!.toFloat() * 4.92 + thisUser.weight!!.toFloat() * 9.99 + 5) * ac-
tivity
        } else if (thisUser.gender == "Женский") {
            dailyCalories = (thisUser.height!!.toFloat() * 6.25 - this-
User.age!!.toFloat() * 4.92 + thisUser.weight!!.toFloat() * 9.99 - 161) * 
activity
        }
    }
    when (thisUser.target) {
        "Похудение" -> {
            dailyCalories -= 500;
        }
        "Набор массы" -> {
            dailyCalories += 500;
        }
    }
    return dailyCalories
}
```

Метод `calculateNorm` рассчитывает норму калорий пользователя в зависимости от его биометрических данных, цели и коэффициента активности.

Листинг 4 – Метод `calculate` в классе `Calculate`

```
private fun calculate() {
    for (i in selectedProducts.indices) {
        caloriesInThisDay += selectedProducts[i]!!.calories / 100f *
weightList[i]
        fatsInThisDay += selectedProducts[i]!!.fats / 100f * weightList[i]
        proteinsInThisDay += selectedProducts[i]!!.proteins / 100f *
weightList[i]
        carbohydratesInThisDay += selectedProducts[i]!!.carbohydrates /
100f * weightList[i]
    }
}
```

Метод `calculate` рассчитывает потребленные КЖБУ пользователем.

В листингах 5 и 6 представлены методы класса `CustomAdapter`: `getView` и `getFilter`.

Листинг 5 – Метод `getView` в классе `CustomAdapter`

```
override fun getView(position: Int, convertView: View?, parent: ViewGroup):
View {
    val inflater = context.getSystemService(Context.LAYOUT_INFLATER_SER-
VICE) as LayoutInflater
    val view = inflater.inflate(R.layout.list_item, parent, false)

    val listItem = view.findViewById<TextView>(R.id.listItemTextView)

    if (position % 2 == 0) {
        val color = Color.argb(25, 0, 102, 94)
        listItem.setBackgroundColor(color)
    } else {
        val color = Color.argb(0, 0, 102, 94)
        listItem.setBackgroundColor(color)
    }

    if (position < filteredItems.size) {
        val item = filteredItems[position]
        listItem.text = item ?: ""
    }

    return view
}
```

Этот метод отвечает за создание и обновление представления для каждого элемента списка (`listView`) в пользовательском интерфейсе.

Листинг 6 – Метод `getFilter` в классе `CustomAdapter`

```
override fun getFilter(): Filter {
    return object : Filter() {
        override fun performFiltering(constraint: CharSequence?): FilterRe-
sults {
            val results = FilterResults()
            val searchText = constraint?.toString()?.toLowerCase()
            val tempList = if (searchText.isNullOrEmpty()) {
                ArrayList(items)
            } else {
                items.filter { it?.toLowerCase()?.contains(searchText) ?:
false } as ArrayList<String?>
            }
            results.values = tempList
            results.count = tempList.size
            return results
        }

        override fun publishResults(constraint: CharSequence?, results:
FilterResults?) {
            filteredItems = results?.values as ArrayList<String?>
            notifyDataSetChanged()
        }
    }
}
```

Этот метод представляет функциональность фильтрации данных в списке для работы поиска продукта по названию.

В листинге 7 представлен метод `createKZHBUChart`.

Листинг 7 – Метод `createKZHBUChart` в классе `Statistic`

```
fun createKZHBUChart(fats: Float, proteins: Float, carbohydrates: Float) {
    val pieChart = findViewById<PieChart>(R.id.pieChart)
    val entries = listOf(PieEntry(fats, "Жиры"),PieEntry(proteins,"Белки"),
        PieEntry(carbohydrates, "Углеводы")
    )
    val dataSet = PieDataSet(entries, "КЖБУ")
    dataSet.valueTextSize = 18f
    // Adding colors
    val colors = listOf(
        Color.rgb(93, 206, 198),
        Color.rgb(255, 176, 115),
        Color.rgb(255, 111, 0)
    )
    dataSet.colors = colors
    val data = PieData(dataSet)
    pieChart.data = data
    pieChart.setDrawEntryLabels(false)
    pieChart.description.isEnabled = false
    pieChart.isDrawHoleEnabled = true
    pieChart.setTransparentCircleAlpha(0)
    pieChart.setHoleRadius(40f)
    pieChart.legend.isEnabled = false
    pieChart.setTransparentCircleRadius(25f)
    pieChart.animateY(1400)
    pieChart.invalidate()
}
```

Данный метод создает круговую диаграмму с помощью библиотеки «MPAndroidChart». Остальные диаграммы в приложении строятся по подобному принципу.

Вывод по пятой главе

В этой главе была представлена реализация различных компонентов облачной базы данных. Кроме того, в рамках этой главы была проведена разработка мобильного приложения, специально адаптированного для платформы Android. Это приложение было разработано с учетом предварительно определенных требований и дизайна, что включало в себя функциональность, интерфейс пользователя и обеспечение совместимости с различными устройствами на платформе Android.

При разработке мобильного приложения особое внимание уделялось его соответствию установленным требованиям, а также интеграции с облачной базой данных. Эта интеграция была необходима для обеспечения эффективной синхронизации данных между мобильным приложением и облачной базой данных, что является ключевым аспектом функциональности приложения. Все эти меры были предприняты для того, чтобы пользователи могли беспрепятственно взаимодействовать с приложением, независимо от используемого устройства, и получать доступ к актуальной информации в режиме реального времени.

Также в ходе реализации дизайн претерпел значительные изменения. Первоначальный дизайн был пересмотрен и доработан с учётом пользовательского опыта и тестов, полученных на различных этапах разработки. Это позволило улучшить не только визуальную привлекательность, но и удобство использования приложения. Изображения готового приложения представлены в приложении В.

6. ТЕСТИРОВАНИЕ СИСТЕМЫ

6.1. Функциональное тестирование

Было проведено функциональное тестирование приложения на соответствие функциональным требованиям. Методология функционального тестирования использовалась для проверки работы мобильного приложения.

Для тестирования приложения было использовано мобильное устройство Samsung Galaxy J4, а также эмулятор устройства Google Pixel XL. Набор тестов приведен в таблице 1.

Таблица 1 – Набор тестов

№	Функция	Шаги	Ожидаемый результат
1	Корректный запуск приложения	Запустить приложение	Приложение запустилось
2	Авторизация через Google	1. Запустить приложение 2. Авторизоваться через Google	Пользователь авторизован
3	Регистрация	1. Запустить приложение 2. Зарегистрироваться	Происходит автоматическая регистрация и авторизация через Google
4	Корректное отображение экранов	1. Запустить приложение 2. Перейти на разные экраны приложения	Экраны приложения отображаются корректно
5	Корректный расчет калорий пользователя без введенных биометрических данных	1. Запустить приложение 2. Перейти на экран расчета приема пищи 3. Добавить продукты из базы данных, а также из личного списка пользователя 4. Рассчитать прием пищи	Статистические данные запишутся успешно, не будет рассчитана норма калорий, вместо нее выведется среднее значение
6	Корректный расчет калорий пользователя с введенными биометрическими данными	1. Запустить приложение 2. Перейти на экран расчета приема пищи 3. Добавить продукты из базы данных, а также из личного списка пользователя 4. Рассчитать прием пищи	Статистические данные запишутся успешно
7	Корректная работа окна «Список продуктов»	1. Запустить приложение 2. Перейти на экран со списком продуктов 3. В поиске начать вводить наименование продукта 4. Нажать на нужный продукт	При начале ввода названия, список начнет меняться, исключай несоответствующие по названию продукты. При нажатии на продукт снизу от списка отобразятся его характеристики

Окончание таблицы 1

№	Функция	Шаги	Ожидаемый результат
8	Корректная работа функции добавления продуктов в личный список	1. Запустить приложение 2. Перейти на экран со списком продуктов 3. Нажать на кнопку «+» и ввести данные продукта 4. Подтвердить добавление продукта	Продукт добавится в личный список продуктов в базе данных, а также отобразится в списке в приложении
9	Корректная работа окна «Аккаунт»	1. Запустить приложение 2. Перейти на экран аккаунта 3. Ввести данные профиля 4. Выйти из аккаунта	После ввода данных профиля, они сохранятся в базу данных и отобразятся в окне «Аккаунт».
10	Корректная работа функции выхода из аккаунта	1. Запустить приложение 2. Нажать на кнопку выхода из аккаунта в меню	После нажатия на кнопку выхода приложение выйдет из аккаунта пользователя и вернет его в окно входа.

В результате функционального тестирования полученные результаты полностью удовлетворяют ожидаемым. Все тесты пройдены успешно.

6.2. Юзабилити тестирование

Юзабилити-тестирование [18] – это процесс оценки удобства использования интерфейса продукта с участием конечных пользователей. Этот вид тестирования помогает определить, насколько продукт удовлетворяет ожиданиям пользователей, выявить проблемные аспекты интерфейса и понять восприятие продукта с точки зрения пользователей. Во время юзабилити-тестирования пользователь выполняет типичные задачи с продуктом под наблюдением тестировщика.

Юзабилити-тестирование было проведено с участием друзей и знакомых людей. В общей сложности в тестировании участвовали пять человек.

Для проверки разработанного мобильного приложения участникам юзабилити-тестирования были предложены следующие задачи:

1) зарегистрироваться и войти в профиль;
2) внести свои данные в профиль;
3) рассчитать один прием пищи;

4) внести в личный список продуктов 20 продуктов;
5) просмотреть статистику питания;
6) выйти из аккаунта в приложении.

Участники тестирования выполнили все задачи без каких-либо сложностей. В ходе данного тестирование были получены предложения по улучшениям, которые окна списка продуктов. Также участники тестирования отметили удобный и интуитивно понятный интерфейс мобильного приложения. В результате тестирования было выявлено, что функционал мобильного приложения полностью соответствует разработанным функциональным требованиям.

Вывод по шестой главе

В процессе тестирования приложения был использован разнообразный набор тестов для проверки его функциональности и соответствия требованиям. Каждый тест содержал определенные шаги и ожидаемый результат, что позволяло детально оценить работу приложения в различных сценариях использования.

Особое внимание уделялось корректности работы различных окон приложения, таких как «Список продуктов» и «Аккаунт». Проверялась возможность ввода данных, их отображение на экране, а также корректность выхода из аккаунта. Полученные результаты тестирования полностью соответствовали ожиданиям, что свидетельствует о успешном прохождении всех тестов и готовности приложения к дальнейшему использованию.

Таким образом, в результате функционального тестирования было подтверждено соответствие полученных результатов ожидаемым, что позволяет сделать вывод об успешном завершении данного этапа разработки и готовности приложения к выпуску.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано Android-приложение для контроля питания. При этом были решены следующие задачи:

1) изучены методы создания приложений на мобильной операционной системе Android;
2) проведен анализ предметной области и сделан обзор существующих приложений по данной тематике;
3) спроектирована база данных для хранения информации приложения;
4) спроектировано мобильное приложение;
5) реализовано мобильное приложение;
6) протестирована работа реализованного приложения.

Также был изучен язык программирования Kotlin для разработки Android-приложений.

Планируется дальнейшее развитие проекта, включающее в себя расширение базы данных с продуктами. Разработанное приложение планируется разместить в магазине RuStore и HuaweiGallery.

ЛИТЕРАТУРА

1. Ожирение и избыточный вес. [Электронный ресурс] URL: https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight (дата обращения: 11.02.2024 г.).

2. Mobile Operating System Market Share Worldwide. [Электронный ресурс] URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата обращения: 11.02.2024 г.).

3. Google Play: MyFitnessPal: Счетчик калорий. [Электронный ресурс] URL: https://play.google.com/store/apps/details?id=com.myfitnesspal.android&hl=en_US (дата обращения: 13.05.2024 г.).

4. Google Play: Счетчик Калорий от FatSecret. [Электронный ресурс] URL: https://play.google.com/store/apps/details?id=com.fatsecret.android&hl=en_US (дата обращения: 13.05.2024 г.).

5. Google Play: Lifesum: здоровое питание. [Электронный ресурс] URL: https://play.google.com/store/apps/details?id=com.sillens.shapeupclub&hl=en_US (дата обращения: 13.05.2024 г.).

6. Documentation Android Developers. [Электронный ресурс] URL: https://developer.android.com/docs (дата обращения: 20.05.2024 г.).

7. Kotlin Documentation. [Электронный ресурс] URL: https://kotlinlang.org/docs/home.html (дата обращения: 20.05.2024 г.).

8. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. // Москва: ДМК Пресс, 2008. – 496 с.

9. Руководство по упрощенному подсчету КБЖУ. [Электронный ресурс] URL: https://yandex.ru/health/turbo/articles?id=6012 (дата обращения: 20.05.2024 г.).

10. Словарь Уэбстера [Электронный ресурс] URL: https://www.merriam-webster.com/ (дата обращения: 04.04.2024 г.).

11. Дейт К.Д. Введение в системы баз данных. // Диалектика, 2019. – 1328 с.

12. Облачные базы данных AWS. [Электронный ресурс] URL: https://aws.amazon.com/ru/products/databases/ (дата обращения: 20.05.2024 г.).

13. База данных SQL Azure. [Электронный ресурс] URL: https://azure.microsoft.com/ru-ru/products/azure-sql/database/ (дата обращения: 20.05.2024 г.).

14. Облако Firestore. [Электронный ресурс] URL: https://firebase.google.com/docs/firestore?hl=ru (дата обращения: 20.05.2024 г.).

15. Firebase Documentation. [Электронный ресурс] URL: https://firebase.google.com/docs?authuser=0&hl=ru (дата обращения: 20.05.2024 г.).

16. Паттерны разработки: MVC vs MVP vs MVVM vs MVI. [Электронный ресурс] URL: https://habr.com/ru/articles/344184/ (дата обращения: 20.05.2024 г.).

17. Доктор Борменталь. Расчет калорий. [Электронный ресурс] URL: https://doctorbormental.ru/kb/telo/raschet-kaloriy/ (дата обращения: 20.05.2024 г.).

18. Басок Б.М. Системы тестирования программного обеспечения. // Москва: РТУ МИРЭА, 2021. – 47 с.

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–7.

Таблица 1 – Спецификация ВИ «Рассчитать калории продукта»

Прецедент: Рассчитать калории продукта
ID: 1
Кратное описание: Расчет калорийности приема пищи по формулам.
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Решение Пользователя рассчитать калорийность приема пищи.
Основной поток: 1. Прецедент начинается, когда Пользователь нажал кнопку «Рассчитать» в окне «Калькулятор». 2. Система производит расчет употребленных калорий и заносит полученные данные в статистику.
Постусловия: Пользователь получает итог расчета и измененную статистику
Альтернативные потоки: отсутствуют

Таблица 2 – Спецификация ВИ «Выбрать продукт из общей базы»

Прецедент: Выбрать продукт из общей базы
ID: 1.1
Кратное описание: Пользователь выбирает продукт из списка БД
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Решение Пользователя выбрать продукт из БД
Основной поток: 1. Прецендент начинается, когда Пользователь зашел в окно «Список продуктов». 2. Пользователь выбирает продукт из общего списка нажатием на него. 3. Система отображает данные выбранного продукта. 4. Пользователь нажимает кнопку «выбрать». 5. Система добавляет продукт в список выбранных продуктов.
Постусловие: Выбранный продукт добавляется в список выбранных продуктов приема пищи
Альтернативные потоки: отсутствуют

Таблица 3 – Спецификация ВИ «Выбрать продукт из личного списка»

Прецедент: Выбрать продукт из личного списка
ID: 1.2
Кратное описание: Пользователь выбирает продукт из личного списка
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Решение Пользователя выбрать продукт из личного списка
Основной поток: 1. Прецендент начинается, когда Пользователь зашел в окно «Список продуктов». 2. Пользователь выбирает продукт из личного списка нажатием на него. 3. Система отображает данные выбранного продукта. 4. Пользователь нажимает кнопку «выбрать». 5. Система добавляет продукт в список выбранных продуктов.
Постусловие: Выбранный продукт добавляется в список выбранных продуктов приема пищи
Альтернативные потоки: отсутствуют

Таблица 4 – Спецификация ВИ «Добавить продукт в личный список»

Прецедент: Добавить продукт в личный список
ID: 1.2.1
Кратное описание: Ввод данных о продукте для добавления в личный список продуктов
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Пользователь не нашел нужный продукт в списке продуктов из БД
Основной поток: 1. Прецендент начинается, когда Пользователь зашел в окно «Список продуктов». 2. Пользователь нажимает на кнопку «+». 3. Система отображает слой с полями для ввода данных продукта. 4. Пользователь вводит данные продукта и нажимает кнопку «Добавить свой продукт». 5. Система добавляет продукт в личный список продуктов.
Постусловие: Введенный продукт добавляется в личный список продуктов пользователя
Альтернативные потоки: отсутствуют

Таблица 5 – Спецификация ВИ «Посмотреть статистику питания»

Прецедент: Посмотреть статистику питания
ID: 2
Кратное описание: Просмотр статистики питания за день
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Решение Пользователя посмотреть статистику питания
Основной поток: 1. Прецендент начинается, когда Пользователь зашел в окно «Статистика». 2. Система отображает статистику питания пользователя.
Постусловия: Пользователь видит статистику питания за день
Альтернативные потоки: I. Пользователь не ввел биометрические данные. 1. Не будет рассчитана рекомендуемое количество калорий в сутки, вместо это будет выведено среднее число

Таблица 6 – Спецификация ВИ «Ввести данные профиля»

Прецедент: Ввести данные профиля
ID: 3
Кратное описание: Пользователь вводит данные профиля
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Решение Пользователя ввести свои антропометрические данные и остальные данные профиля
Основной поток: 1. Прецендент начинается, когда Пользователь зашел в окно «Профиль». 2. Система отображает данные профиля пользователя. 3. Пользователь нажимает кнопку «Ввести свои данные». 4. Система отображает слой с полями для ввода биометрических данных, цели и коэффициента активности пользователя. 5. Пользователь вводит данные и нажимает кнопку «Сохранить» 6. Система сохраняет данные профиля
Постусловие: Пользователь может видеть обновленные данные профиля и рассчитанную для него норму калорий
Альтернативные потоки: отсутствуют

Таблица 7 – Спецификация ВИ «Выбрать цель»

Прецедент: Выбрать цель
ID: 3.1
Кратное описание: Пользователь выбирает цель использования приложения
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Решение Пользователя выбрать цель использования приложения
Основной поток: 1. Прецендент начинается, когда Пользователь зашел в окно «Профиль». 2. Система отображает данные профиля пользователя. 3. Пользователь нажимает кнопку «Ввести свои данные». 4. Система отображает слой с полями для ввода биометрических данных, цели и коэффициента активности пользователя. 5. Пользователь выбирает цель и нажимает кнопку «Сохранить» 6. Система сохраняет данные профиля
Постусловие: Система меняет коэффициенты в формуле для помощи пользователю в достижении цели
Альтернативные потоки: отсутствуют

Приложение Б. Листинг авторизации и аутентификации

Метод `onCreate` класса `Account`, включающий в себя код считывания данных из облачной базы данных, представлен в листинге 1.

Листинг 1 – Метод `onCreate` в классе `Account`

```
override fun onCreate(savedInstanceState: Bundle?) {
    db = FirebaseDatabase.getInstance()
    users = db!!.getReference("Users")
    users!!.child(uid).addValueEventListener(object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            if (snapshot.exists()) {
                val discrAge = snapshot.child("age").value.toString()
                showAge?.let {it.text = discrAge}
                age?.let {it.text = Editable.Factory.get-
Instance().newEditable(discrAge)}
                val discrWeight = snapshot.child("weight").value.toString()
                showWeight?.let {it.text = discrWeight }
                weight?.let {it.text = Editable.Factory.get-
Instance().newEditable(discrWeight)}
                val discrGender = snapshot.child("gender").value.toString()
                showGender?.let {it.text = discrGender}
                val genderArray = resources.getStringArray(R.array.sex)
                val indexGender = genderArray.indexOf(discrGender)
                gender?.setSelection(indexGender)
                var discrTarget = snapshot.child("target").value.toString()
                if (discrTarget == "null" || discrTarget == "") {
                    discrTarget = "Нет данных"
                }
                showTarget?.let {it.text = discrTarget}
                val targetArray = resources.getStringArray(R.array.target)
                val indexTarget = targetArray.indexOf(discrTarget)
                target?.setSelection(indexTarget)
                var discrActivity = snapshot.child("activ-
ity").value.toString()
                if (discrActivity == "null" || discrActivity == "") {
                    discrActivity = "Нет данных"
                }
                showActivity?.let {it.text = discrActivity}
                val activityArray = resources.getStringArray(R.array.activ-
ityKoeff)
                val indexActivity = activityArray.indexOf(discrActivity)
                activity?.setSelection(indexActivity)
                val discrHeight = snapshot.child("height").value.toString()
                showHeight?.let {it.text = discrHeight}
                height?.let {it.text = Editable.Factory.get-
Instance().newEditable(discrHeight)}
            } else {
                showWeight?.let {it.text = "Нет данных" }
                showAge?.let {it.text = "Нет данных" }
                showGender?.let {it.text = "Нет данных" }
                showTarget?.let {it.text = "Нет данных" }
                showActivity?.let {it.text = "Нет данных" }
                showHeight?.let {it.text = "Нет данных" }
            }
        }
        override fun onCancelled(error: DatabaseError) {}
    })
}
```

Код авторизации и аутентификации показан в листинге 2.

Листинг 2 – Класс авторизации и аутентификации `Loging`

```
class Loging : AppCompatActivity() {
    lateinit var launcher: ActivityResultLauncher<Intent>
    lateinit var auth: FirebaseAuth

    private var mAuth: FirebaseAuth? = null

    // [END declare_auth]
    private var mGoogleSignInClient: GoogleSignInClient? = null
    @SuppressLint("SuspiciousIndentation")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_loging)
        auth = Firebase.auth

        launcher = registerForActivityResult(ActivityResultContracts.Star-
tActivityForResult()){
            val task = getSignedInAccountFromIntent(it.data)
            try {
                val account = task.getResult(ApiException::class.java)
                if (account != null){
                    firebaseAuthWithGoogle(account.idToken!!)
                }
            } catch (e: ApiException){
                Log.d("authLog", "Api exception")
            }
        }
    }

    private fun getClient():GoogleSignInClient{
        val gso = GoogleSignInOptions
            .Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken(getString(R.string.default_web_client_id))
            .requestEmail()
            .build()
        return GoogleSignIn.getClient(this, gso)
    }

    private fun signIgWithGoogle(){
        val signInClient = getClient()
        launcher.launch(signInClient.signInIntent)
    }

    private fun firebaseAuthWithGoogle(idToken: String){
        val credential = GoogleAuthProvider.getCredential(idToken, null)
        auth.signInWithCredential(credential).addOnCompleteListener{
            if(it.isSuccessful){
                Log.d("authLog", "Google signIn done")
                val intent = Intent(this, Calculate::class.java)
                startActivity(intent)
            } else {
                Log.d("authLog", "Google signIn error")
            }
        }
    }
```

```
    public override fun onStart() {
        super.onStart()
        // Check if user is signed in (non-null) and update UI accordingly.
        val currentUser = auth.currentUser
        updateUI(currentUser)
        if (currentUser != null) {
            Companion.signedInAccountId = currentUser.uid
            Log.d("UserEmail", currentUser.email!!)
            val intent = Intent(this, Calculate::class.java)
            startActivity(intent)
        }
    }

    fun singOut() {
        FirebaseAuth.getInstance().signOut()
    }
}
```

Приложение В. Скриншоты приложения

Скриншоты приложения показаны на рисунках 1–3.

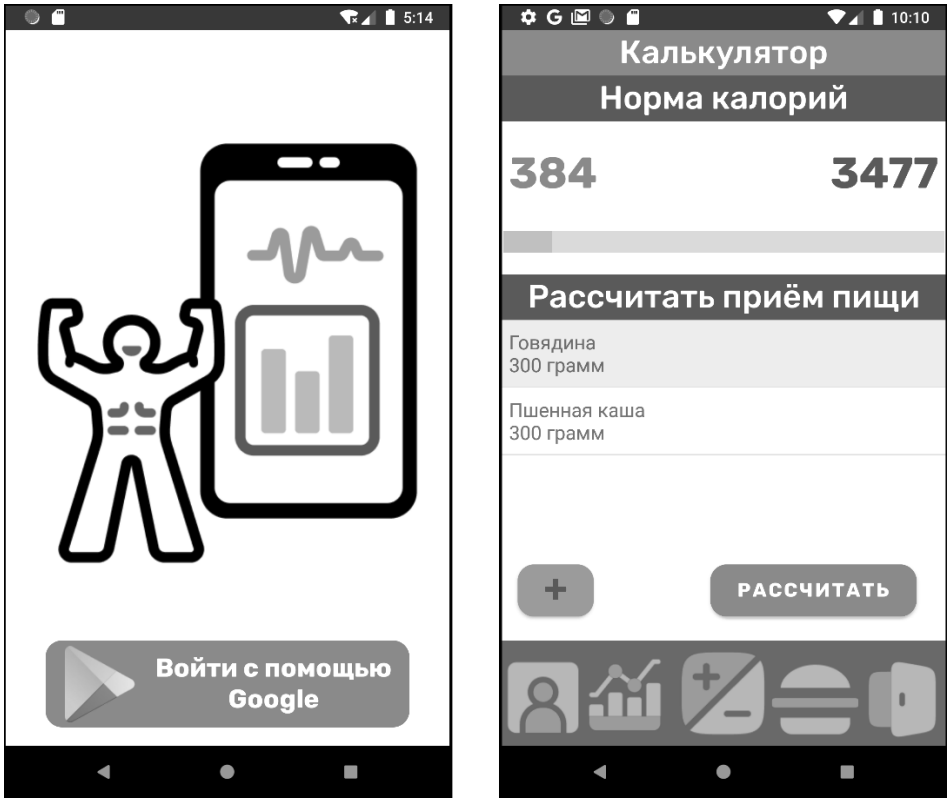

Рисунок 1 – Скриншоты окон «Вход» и «Калькулятор»

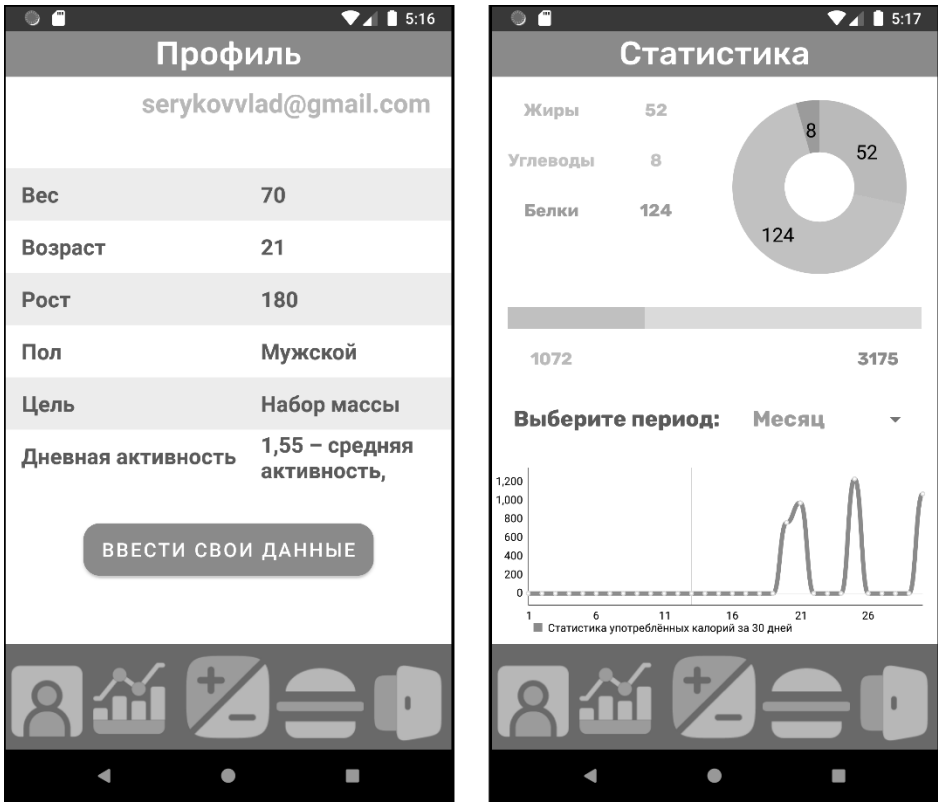

Рисунок 2 – Скриншоты окон «Профиль» и «Статистика»

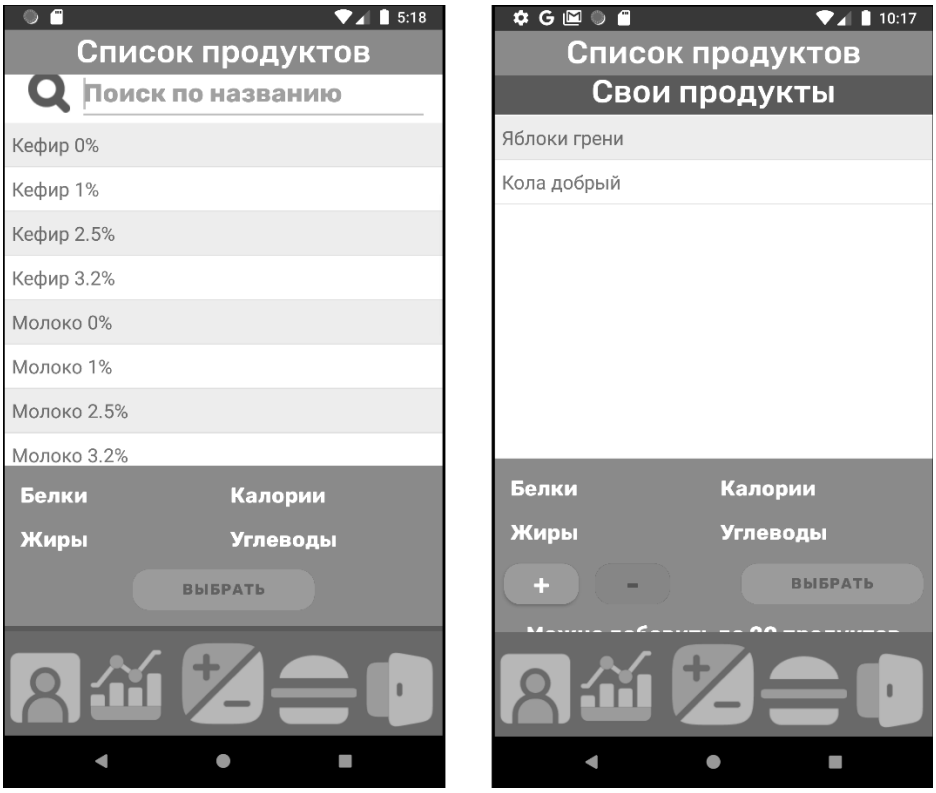

Рисунок 3 – Скриншоты окна «Список продуктов»