## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

На сегодняшний день такое заболевание, как сахарный диабет, приобретает все большее и большее распространение. Это обусловлено изменением стиля жизни современного человека, уменьшением подвижности его тела в связи с преобладанием умственного труда над физическим, потреблением нездоровой пищи и развитием других вредных привычек. Число людей с диабетом возросло со 108 миллионов в 1980 году до 422 миллионов в 2014 году [1]. Наблюдая такую динамику распространения болезни, невозможно легкомысленно относиться к этому явлению.

За последние 10 лет смартфоны прочно закрепились в повседневной жизни людей, так как эти устройства поддерживают огромное количество функций и заменили собой множество других гаджетов, которые ранее приходилось приобретать отдельно, такие как: калькулятор, камера, *mp3*-плеер и множество других устройств. Современные смартфоны способны заменить даже персональный компьютер. Такую универсальность смартфоны имеют благодаря наличию различных приложений и возможности расширять функционал устройства благодаря установке дополнительных программ. Если принять во внимание этот момент, то хорошим решением является использование смартфона и в оздоровительных целях, а именно использование устройства в качестве трекера здоровья, куда человек будет заносить данные о своем здоровье и, анализируя предоставляемую приложением статистику, корректировать свой образ жизни для того, чтобы избегать причин возникновения заболевания.

Целью данной работы является разработка мобильного приложения для операционной системы *Android*, которое позволит пользователю вести учет потребляемых им лекарственных средств, калорий, количества времени физической активности, уровня глюкозы в крови. Также приложение должно иметь функции напоминания пользователю о времени принятия необходимых лекарственных средств, сигнализирования пользователю о превышении или занижении нормы потребляемых микроэлементов и уровня глюкозы в крови. Для того, чтобы в процесс наблюдения здоровья было легче вовлечь детей, приложение должно иметь игровую форму с вознаграждением пользователя за постоянный ввод данных.

Для выполнения приведённых целей необходимо создать *RESTful API* серверное приложение с подключением к базе данных для обеспечения возможности получения клиентским приложением данных. Далее реализовать клиентское приложение с удобным пользовательским интерфейсом, которое бы обращалось к серверу за данными, получало их, сохраняло в памяти смартфона и предоставляло возможности для выполнения описанных выше функций.

Дипломный проект выполнен самостоятельно, проверен в системе «Антиплагиат». Скриншот отчета приведен в приложении А. Процент оригинальности составляет 86,25%. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников».

# 1 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И ПОСТАНОВКА ЗАДАЧ

## 1.1 Анализ исходных данных и функциональных возможностей программного средства

Основными функциями разрабатываемого в данном проекте приложения является система учета данных о состоянии здоровья пользователя, сбор статистических данных, вносимых им, осуществление напоминаний пользователю о необходимости принятия лекарственных средств, отображение среднестатистических данных на основе информации, введенной пользователем, выдача подробной детализированной информации, которая была внесена в течение определенного промежутка времени. Также важной функцией является создание учетной записи пользователя с привязкой к адресу электронной почты и номеру телефона, что позволит пользователю осуществлять доступ к приложению на более высоком уровне безопасности. В случае утери пароля пользователь сможет продолжать осуществлять доступ к приложению посредством привязанного номера телефона.

Анализируя существующие аналоги разрабатываемого приложения, можно сделать вывод о том, что вышеописанные функции присутствуют во всех похожих приложениях в том или ином виде. Все рассмотренные аналоги имеют различный интерфейс, но практически идентичны по функциям. Каждое приложение имеет функции сбора данных, напоминания, оценку состояния пользователя на основе введенных данных и т.д.

Все вышеупомянутые функции безусловно очень нужны и приносят большую пользу людям, но если немного проанализировать потенциальную аудиторию пользователей такого рода приложений, то можно обнаружить, что большую долю составляют люди подросткового и даже детского возраста. По данным Международной диабетической федерации на 2016 год, только в Европе проживает более 140 тысяч детей в возрасте до 14 лет, имеющих диабет первого типа [3]. Существующие приложения учета здоровья имеют интерфейс и логику, понятные для взрослого человека. Но далеко не каждые ребенок или подросток сможет разобраться с тем, как пользоваться такого рода приложением. Поэтому в разрабатываемом в данном проекте приложении было решено добавить игровую логику с виртуальным персонажем, взаимодействуя с которым, пользователю будет проще понять суть приложения и выполнять необходимые действия. Было решено создать функцию чат-бота для симуляции общения пользователя и виртуального персонажа, а также систему вознаграждения пользователя за ввод данных.

Приложение, которое должно получиться в результате выполнения дипломного проекта на данный момент не имеет аналогов в магазинах приложений.

## 1.2 Обзор существующих программных средств по теме дипломного проекта

Если начать поиск похожих программных средств по теме дипломного проекта в магазинах приложений, то можно найти достаточно большое количество продуктов, которые очень схожи по своим функциям. Рассмотрим некоторые из них.

Первое из обнаруженных приложений – «*Diabetes:M*». Скриншот одного из экранов можно видеть ниже на рисунке 1.1.

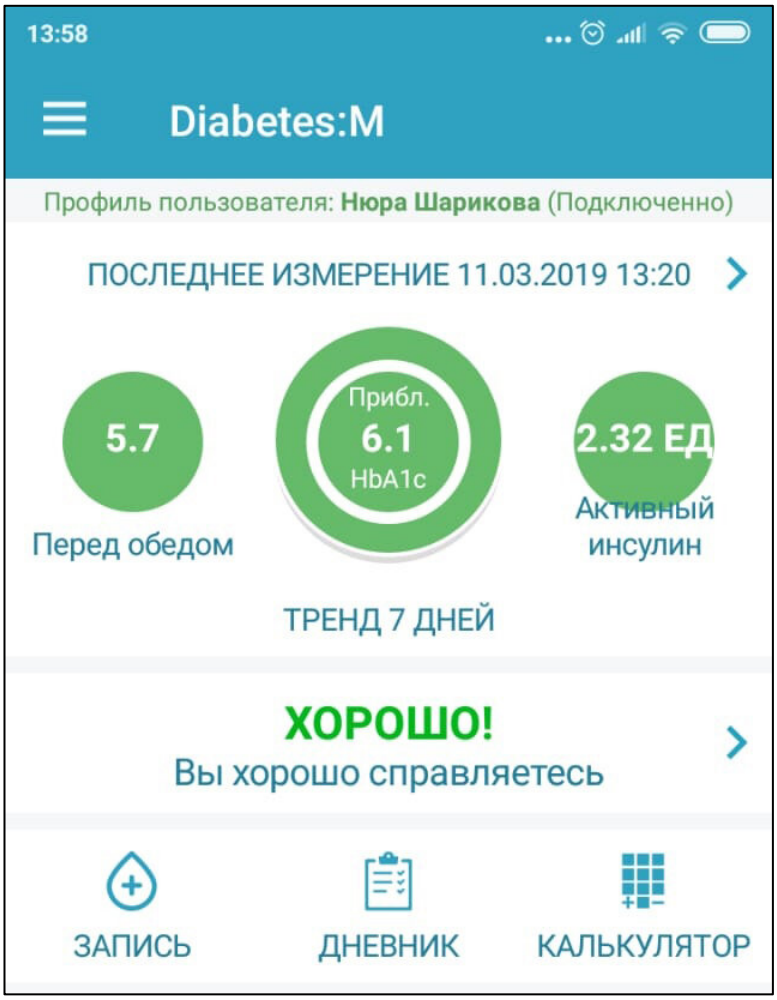



Рисунок 1.1 – Скриншот приложения «*Diabetes:M*»

Данное приложение имеет интеграцию с сервисом *Google Fit*, который отслеживает данные о здоровье пользователя со смартфона и других носимых устройств. Также можно вводить в нее лабораторные исследования крови и мочи, записывать давление, задавать значения на занятия спортом и данные о болезни. Данное приложение обладает следующими функциями:

– калькулятор, рассчитывающий ту дозу инсулина, которую необходимо ввести, опираясь на заданные ей значения;

– расчет активного инсулина в организме;

– дневник питания;

– наглядная визуализация данных дневника в виде графиков и диаграмм за период от недели до трех месяцев.

Рассмотрим другое приложение – «*DiaMeter*». Это не менее удобное приложение для ведения записей об уровне сахара, количестве съеденных вами хлебных единиц, инъекциях короткого и продлённого инсулина, а также о самочувствии в целом. Есть наглядная статистика и возможность синхронизации всех данных в облаке. Дополнительно *DiaMeter* предлагает ряд интерактивных статей, в которых можно найти немало полезной информации о правильном питании при диабете, физических нагрузках, влиянии алкоголя и другом. Скриншот данного приложения изображен на рисунке 1.2.

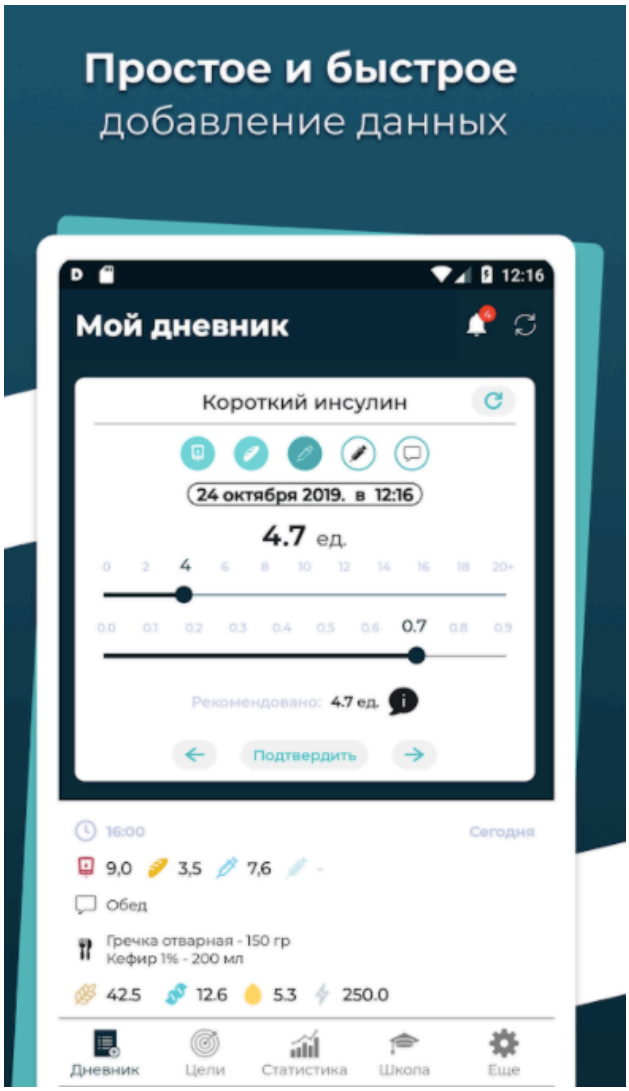



Рисунок 1.2 – Скриншот приложения *«DiaMeter»*

Оба приложения, рассмотренные выше, очень хороши, удобны и полезны для взрослого пользователя, который осознает свое заболевание и достаточно замотивирован в лечении и наблюдении за своим состоянием здоровья. Однако рассмотренные приложения будут очень сложны в восприятии более юной аудиторией.

## 1.3 Обоснование выбора языка программирования и средств разработки

При разработке сложных программных средств под мобильные операционные системы следует с особым вниманием подходить к выбору средств разработки, так как от этого зависит скорость и качество процесса создания программного средства.

Для выбора подходящей среды разработки необходимо учесть ряд определённых факторов, которые могут оказать сильное влияние на разработку:

1 Интерфейс. Этот первый компонент, с которым сталкивается пользователь после установки и который формирует первое впечатление о программе, и, на основании которого, может быть осуществлен окончательный выбор. Здесь оценивается не только общий дизайн, хотя, безусловно, он тоже сказывается определенным образом, но и удобство расположения и настройки таких компонент как окно исходного кода, окно проекта;

2 Настройка. Соответственно после установки и первого запуска среды разработки выполняется ее настройка, т.е. указываются пути, по которым располагаются установленные *SDK*, *DOCS*, *J2EE*. В этом компоненте, как правило, существенных различий не наблюдается. Более того, современные средства, как правило, самостоятельно определяют установленные компоненты;

3 Редактор кода. Настройка отображения исходных кодов, как правило, тоже не отличается разнообразием, в любом средстве легко можно настроить кеглю и ее размер, а также цвет. Немаловажным преимуществом является наличие помощника, когда, при «зависании» мышки на любой переменной или методе всплывает довольно подробный *ToolTip* (контекстное окно) в котором развернуто, описаны все параметры объекта. Также есть масса приятных мелочей, вывод нумерации строк, отображение структуры класса, показ символов абзаца, проверка орфографии.

На сегодняшний день среди мобильных операционных систем получили наибольшее распространение *Android* и *iOS*. Большинство мобильных устройств работают под одной из данных систем. Согласно мировой статистике [4], 74.3% пользователей смартфонов используют операционную систему *Android* и 24.7% – *iOS*. Благодаря тому, что преобладает всего две операционные системы, широкое распространение сегодня получают различные кроссплатформенные фреймворки, позволяющие из одной и той же кодовой базы создать одновременно приложение как под одну операционную систему, так и под другую. Использование такого подхода позволяет сэкономить не только временные ресурсы, но также и финансовые. Производительность таких кроссплатформенных приложений очень зависит от выбранной технологии. Лучшие технологии для кроссплатформенной разработки на сегодняшний день мало чем уступают нативным приложениям в производительности. Исходя из этого было решено для разработки

приложения данного дипломного проекта использовать кроссплатформенный фреймворк.

Одним из наиболее популярных фреймворков для разработки мобильных кроссплатформенных приложений является *React Native* [5]. *React Native* – фреймворк с открытым исходным кодом, разработанный для создания кроссплатформенных мобильных приложений для операционных систем *iOS* и *Android*. Данная платформа использует язык *JavaScript* для создания приложений. Структура платформы отображена ниже на рисунке 1.3.

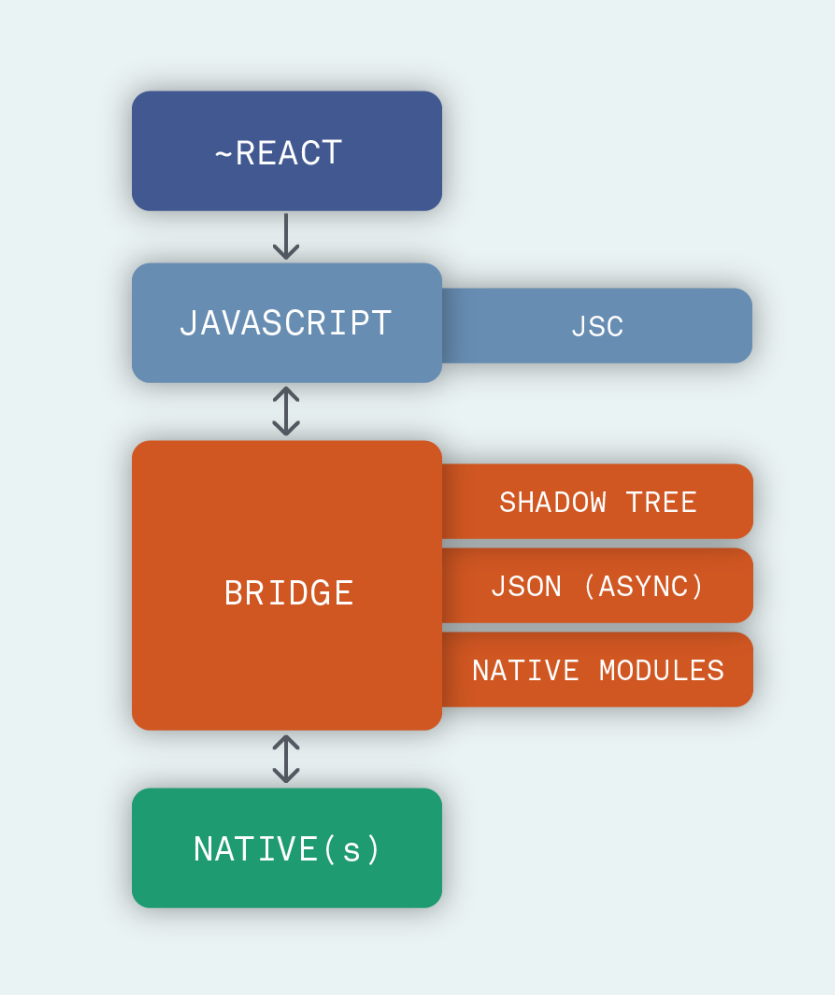



Рисунок 1.3 – Структура платформы *React Native* [5]

*React Native* использует библиотеку *React* в качестве основы для написания исходного кода приложения. Код, написанный с использованием этой библиотеки, компилируется в один файл, так называемый *bundle*.

Также на рисунке можно видеть, что собранный *JavaScript*код взаимодействует с «нативной» платформой посредством моста, который содержит в себе всю логику работы фреймворка.

Приложение, разработанное с использованием *React Native*, очень сложно отличить от разработанного с использованием родных платформе инструментов. Это обусловлено строением архитектуры данного фреймворка и его работы. Полностью готовое *React Native* приложение представляет собой

«нативное» *Android* или *iOS* приложение, которое при открытии передает управление *JavaScript*потоку, содержащему всю бизнес-логику приложения, а также пользовательский интерфейс. Данный поток может управлять работой родных для мобильной платформы компонентов. Например, открыть модальное окно, добавить *Drawer*-меню и так далее. Фреймворк также поддерживает обработку специфических для каждой платформы жестов и анимации.

```
77  render() {
78    const options = {
79      shouldSort: true,
80      threshold: 0.4,
81      location: 0,
82      distance: 100,
83      maxPatternLength: 32,
84      minMatchCharLength: 1,
85      keys: ['name', 'number', 'verses.rows', 'chorus.rows', 'bridge.rows'],
86    };
87    const fuse = new Fuse(this.props.items, options);
88    const result = fuse.search(this.props.searchRow);
89
90    const data = this.props.searchRow.length !== 0 ? result : this.props.items;
91
92    return (
93      <>
94        <SongsList
95          parentComponentId={this.props.componentId}
96          isLoading={this.props.isLoading}
97          isNetConnected={this.state.isNetConnected}
98          data={data}
99          fetchSongs={this.props.fetchSongs}
100         favorites={this.props.favorites}
101         ListEmptyComponent={(
102           this.props.searchRow.length !== 0 && result.length === 0
103             ? (
104               <Text style={styles.title}>Ничего не найдено...</Text>
105             )
106             : (
107               <View style={styles.emptyScrollViewContainer}>
108                 <Text style={styles.title}>Список песен пуст...</Text>
109                 <Text style={styles.title}>Потяните вниз, чтобы синхронизировать его.</Text>
110               </View>
111             )
112         )}
```

Рисунок 1.4 – Пример программного кода

С точки зрения разработчика, пишущего непосредственно код приложения, *React Native* практически не отличается от библиотеки *React*. Эта библиотека является отличным примером реактивного программирования, ориентированного на реакцию к определенным событиям. Также особенностью *React* и *React Native* является *JSX*-разметка компонентов, используемая при построении пользовательского интерфейса. Данная разметка похожа на разметку *html*-документа, что облегчает освоение данных платформ разработчиками, которые имели опыт работы в *web*-разработке. На рисунке 1.4 отображен пример программного кода, написанного при использовании платформы *React Native*.

В сущности хакерской культуры *Facebook*, *React Native* начинался как проект хакатона летом 2013 года. Подобно *React* , *React Native* казался смелой нетрадиционной идеей. Не было ясно, что это действительно сработает. Как работали сенсорные переговоры между *JS* и собственными *ScrollViews*? Как насчет производительности, а как насчет отладки? Ни одна из этих проблем не помешала инженерам сосредоточиться и двигаться вперед.

Менеджер рекламы *Facebook* для *iOS* вышел в феврале 2015 года, менее чем через шесть месяцев после того, как команда разработчиков начала работать над ним. Одновременно все, кто сосредоточился на *JS* или *iOS*, работали над открытым исходным кодом реализации *iOS*. В январе 2015 года на *React.js Conf* состоялся первый публичный предварительный просмотр , а на конференции разработчиков *F8* в марте 2015 года был открыт код для всех.

Сразу после этого инженеры по продукту *Ads Manager* начали переносить свой код *JavaScript* на *Android*, тесно сотрудничая с командой *React Native Android* в Лондоне. *Facebook* не стремился к совместному использованию кода между платформами при работе с *iOS Ads Manager*, но ожидалось, что это может быть хорошим побочным эффектом использования *React Native*. Когда *Android Ads Manager* был готов к поставке было понятно, что около 85 процентов кода было поделено между двумя приложениями.

В июне 2015 года, после трех месяцев разработки и месяца внутренних собеседований, была выпущена первая версия менеджера рекламы на *Android* . Учитывая, как взлетела *React Native* для *iOS*, *Facebook* сразу же переключили наше внимание на открытый исходный код *React Native* для *Android*, и ожидали большой интерес к выпуску *Android*. В конце концов, необходимость создавать одно и то же приложение отдельно для каждой платформы – проблема в отрасли, и это было известно из опыта работы с Менеджером рекламы, что *React Native* может помочь с этим [6].

Подобно запуску *iOS*, *Facebook* хотел выпустить *React Native* для *Android* как можно раньше, чтобы получить обратную связь. Поэтому они начали только с небольшого количества представлений и модулей (*Text*, *Image*, *ScrollView*, *Network*, *AsyncStorage* и некоторых других). 14 сентября было выдвинуто ядро среды выполнения *Android* и начальный набор модулей *Android* на *GitHub* и *npm*. И именно так *React Native* версии 0.11 был первым выпуском с поддержкой *Android*. Были добавлены следующие модули *Android* с тех пор, как открыли исходный код, приблизив паритет *API* к *iOS*: оповещения, *AppState*, *CameraRoll*, буфер обмена, указатели даты и времени, геолокация, намерение, модальные, *NetInfo*, извлечение для обновления представления, средство выбора, слайдер, просмотр пейджера и вебпросмотр.

Само собой разумеется, что принятие, которое последовало за пределами *Facebook*, очень вдохновило на дальнейшую работу в команде *React Native*.

На текущий момент последней стабильной версией *React Native* является 0.61.5 и с момента первого релиза изменилось многое. Но одно можно утверждать точно – *React Native* развивается и учитывает свои прошлые ошибки двигаясь вперед.

Платформа *React Native* обладает достаточно большим количеством достоинств. Благодаря существованию этих достоинств некоторые популярные приложения, таких как *Instagram, Airbnb, Walmart, UberEats, Tesla, Skype* и другие, уже активно используют этот фреймворк. *React Native* предоставляет все основные и передовые методы разработки гибридных мобильных приложений. Перейдем к рассмотрению некоторых причин, по которым использование *React Native* для разработки мобильных приложений является хорошим решением [7].

Далее будут освещены достоинства платформы *React Native.*

**Единая кодовая база для *Android* и *iOS.*** *React Native* позволяет писать код один раз и запускать как на *Android*, так и на *iOS*. Кодовая база будет одинаковой для обеих платформ, поэтому вам не придется тратить деньги и время на два проекта разработки.

Поскольку *React Native* имеет открытый исходный код и способствует повторному использованию, вы можете повторно использовать компоненты в любое время на любом уровне, не переписывая его и не перекомпилируя приложение.

**Возможность повторного использования.** Теперь поговорим о возможности повторного использования. *React Native* приносит блоки, которые состоят из повторно используемых «нативных компонентов». Почти все компоненты *Android* и *iOS* имеют свои аналоги в *React Native*. Все эти компоненты компилируются непосредственно в нативные приложения и, таким образом, позволяют разработчикам поддерживать внешний вид платформы.

Ранее в гибридных приложениях было невозможно получить структуру, специфичную для компонента. Но с *React Native* эта структура позволяет разработчикам создавать приложения, используя гибкие вебметодологии.

С другой стороны, *React Native* также позволяет разработчикам добавлять компоненты *«native»* в код своих существующих приложений. Им не придется кодировать с нуля, чтобы преобразовать существующее мобильное приложение в приложение *React Native*.

**Производительность.** *React Native* сокращает цикл разработки, позволяет разрабатывать приложения очень быстро и позволяет доставлять приложения максимально быстро. Он использует библиотеку *ReactJS UI*, разработанную *Facebook* для пользовательских интерфейсов. В то время как собственные приложения работают на центральном процессоре, *React Native* использует графический процессор.

Он поставляется с уникальной функцией под названием «живая перезагрузка», которая позволяет просматривать последние изменения кода сразу, разбивая экран на две части. Первый показывает код, а второй показывает результат кода в кадре мобильного телефона. В целом, с помощью *React Native* компании могут сократить расходы на разработку почти на 50% без ущерба для качества или производительности.

**Ориентация на пользовательский интерфейс.** *React Native* – полностью ориентированный на пользовательский интерфейс. Декларативный

*API* значительно упрощает понимание ваших требований и прогнозирование пользовательского интерфейса. Декларативный стиль позволяет вам контролировать поток и состояние в вашем приложении, говоря: «Это должно выглядеть так». Вам больше не придется беспокоиться о деталях реализации.

**Совместимость с сторонними плагинами.** Добавление аппаратных возможностей устройства в приложение является одним из наиболее распространенных требований в наши дни. В отличие от функций *WebView, React Native* позволяет напрямую связать плагин с собственным модулем через инфраструктуру. Это приводит к более плавному запуску приложения, более быстрой загрузке и меньшим требованиям к памяти.

**Сравнение *React Native* с другими платформами.** Для разработки приложения был выбран кроссплатформенный фреймворк *React Native*. По этому для сравнения с данной технологией, был выбран *Flutter*.

*Flutter – SDK* с открытым исходным кодом для создания мобильных приложений от компании *Google*. Он используется для разработки приложений под *Android* и *IOS*, а также это пока единственный способ разработки приложений под *Google Fuchsia* [8].

*React Native* создан для приложений на *IOS*, *Android* и *Windows*, при этом использует язык программирования *JavaScript* и библиотеку *React.js* как главное средство разработки. Фреймворк был создан и поддерживается компанией *Facebook*, которые разработали *React.js*.

*Flutter* в свою очередь создан для приложений *Android*, *IOS* и *Fuchsia*, в отличии от *React Native*, для разработки используется язык программирования *Dart*, который также служит для вебпрограммирования.

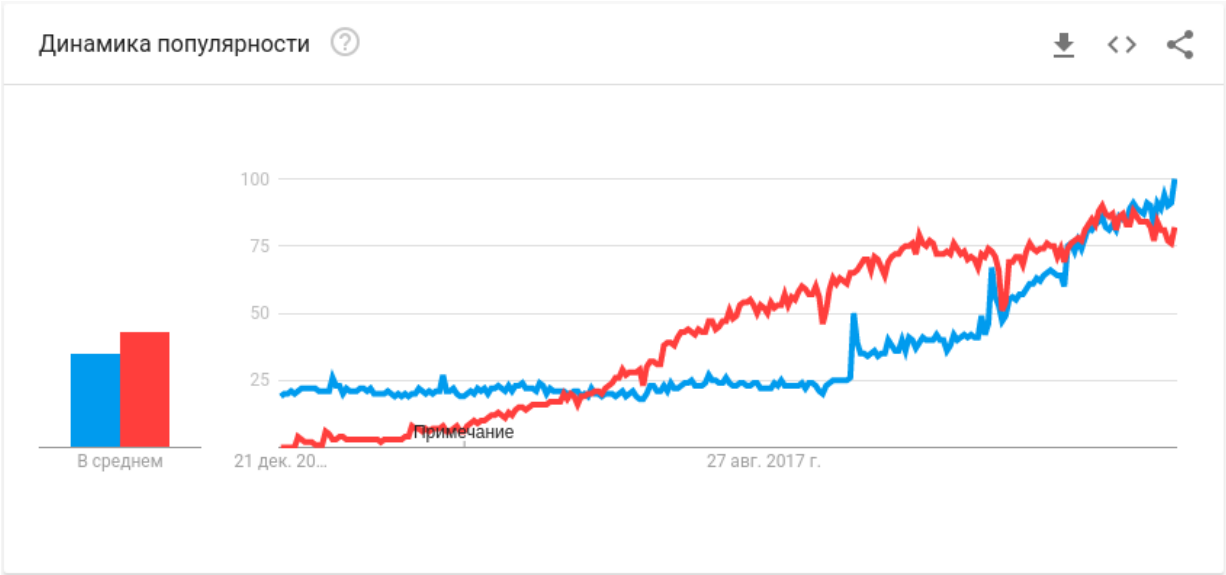



Рисунок 1.5 – График популярности *React Native* и *Flutter* [9]

На рисунке 1.5 приведен график популярности *React Native* (красный) и *Flutter* (синий), согласно статистике *Google Trends.*

Для сравнения *React Native* и *Flutter* компанией *SimbirSoft* была создана таблица 1.1 в зависимости от характера приложения.

Опираясь на выше перечисленные критерии для разработки небольшого приложения с использованием сторонних библиотек хорошим выборам будет *React Native* изза своей простоты и популярности.

В качестве основной среды разработки была выбрана *WebStorm* от компании *JetBrains* [11]. Данная интегрированная среда разработки является лучшей для написания кода на языках *JavaScript* и *TypeScript* на сегодняшний день.

Таблица 1.1 – Сравнение *React Native* и *Flutter* [10]

| | *React Native* | *Flutter* |
|---|---|---|
| Поддержка и сообщество | Крупное сообщество, большое количество библиотек. | Маленькое активно развивающееся сообщество. |
| Основные элементы | «Нативные» компоненты пользовательского интерфейса, которые адаптируются под каждый экран без каких-либо дополнительных усилий. | Собственные компоненты, близкие к «нативным» по виду |
| Работа с «нативными» функциями | Сторонние библиотеки для доступа к «нативным» функциям | Прямой доступ к основным функциям устройства |
| Представление | Используется *JS Bridge* для связи с платформой, поэтому скорость не высокая | Высокая производительность (60 кадров в секунду) |
| Написание кода | Есть возможность разделить логику и стили компонентов | Разметка неотделима от кода логики компонента |
| Популярность | Большое количество готовых решений, высокая популярность | Невысокая популярность. Растет достаточно высокими темпами. |

Так как в теме дипломного проекта обозначена разработка под систему *Android*, то для разработки приложения также будет необходима среда *Android Studio. Android Studio* это официальная интегрированная среда разработки (*IDE*) для разработки *Android*-приложений, основанная среде разработки *IntelliJ IDEA* [12].

### 1.4 Постановка задач по разработке программного средства

Разрабатываемое приложение должно иметь удобный, интуитивно понятный интерфейс, в котором самые важные функции находятся в ближайшей доступности, а более глубокие возможности не должны перегружать пользовательский интерфейс.

В результате анализа аналогов на рынке приложений, а также анализа предметной области сформирован следующий список требований к разрабатываемой системе:

- возможность авторизации в системе;
- возможность просмотра профиля пользователя;
- возможность внесения данных о потребленных пользователем калориях, уровне глюкозы в крови, времени физической активности;
- возможность устанавливать получение уведомлений с напоминанием о принятии лекарственных средств;
- возможность получения пользователем статистики своих данных;
- реализация игровой логики и системы вознаграждения пользователя;
- реализация чат-бота.

Реализация данного списка функций позволит создать программный продукт, не имеющий аналогов на рынке приложений. Игровая логика программного средства позволит привлечь детскую и подростковую аудитории целевого сегмента потребителей. В таком случае можно будет сделать вывод об успешном проектировании и разработке программного средства и выполнении поставленных целей для данного проекта.

# 2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 2.1 Разработка архитектуры программного средства

Программный код дипломного проекта будет разрабатываться в архитектурном стиле *Model-View-Controller* (*MVC*, «Модель-Представление-Контроллер»).

*Activity* – это компонент приложения, который выдает экран, и с которым пользователи могут взаимодействовать для выполнения каких-либо действий, например набрать номер телефона, сделать фото, отправить письмо или просмотреть карту. Каждой операции присваивается окно для прорисовки соответствующего пользовательского интерфейса. Обычно окно отображается во весь экран, однако его размер может быть меньше, и оно может размещаться поверх других окон [13].

Для эффективной работы системы *Android*, в ней были созданы жизненные циклы различных элементов программного средства. Жизненный цикл *Android*-приложения жёстко контролируется системой и зависит от нужд пользователя, доступных ресурсов и производительности устройства. Например, когда пользователь хочет запустить браузер, решение о запуске приложения принимает система. Хоть последнее слово и остаётся за системой, она подчиняется заданным и логическим правилам, позволяющим определить, можно ли загрузить или приостановить приложение, прекратить его работу. Когда в определённый момент пользователь работает с конкретным окном, система даёт приоритет соответствующему приложению. И наоборот, если окно невидимо, и система решает, что работу приложения необходимо остановить, чтобы освободить дополнительные ресурсы, будет остановлена работа приложения, имеющего более низкий приоритет. В *Android* ресурсы на мобильном устройстве ограниченны, поэтому он более жёстко контролирует работу приложений [14].

У активити есть свой жизненный цикл, который показан на рисунке 2.1.

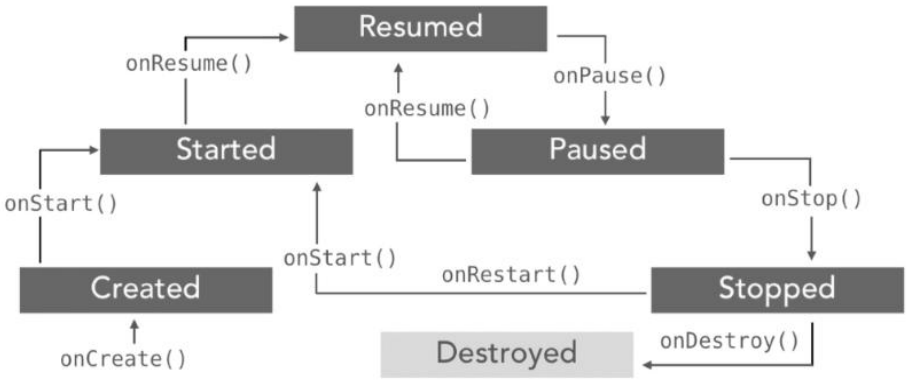



Рисунок 2.1 – Жизненный цикл *Activity*

Метод *onCreate()* вызывается при создании или перезагрузке активности. Система может запускать и приостанавливать текущие окна в

зависимости от происходящих событий. Внутри данного метода настраивают статический интерфейс окна. В методе инициализируются статические данные активности, связывают данные со списками и так далее. Связывает с необходимыми данными и ресурсами. Задаёт внешний вид через метод *setContentView()*. В этом методе загружаем пользовательский интерфейс, размещаем ссылки на свойства класса, связываем данные с элементами управления, создаваем сервисы и потоки. Метод *onCreate()* принимает объект *Bundle*, содержащий состояние пользовательского интерфейса, сохранённое в последнем вызове обработчика *onSaveInstanceState*. Для восстановления графического интерфейса в его предыдущем состоянии нужно задействовать эту переменную: внутри *onCreate()* или переопределив метод *onRestoreInstanceState()*. В методе можно сделать проверку, запущено ли приложение впервые или восстановлено из памяти. Если значение переменной *savedInstanceState* будет *null*, приложение запускается первый раз.

За *onCreate()* всегда следует вызов *onStart()*, но перед *onStart()* не обязательно должен идти *onCreate()*, так как *onStart()* может вызываться и для возобновления работы приостановленного приложения (приложение останавливается методом *onStop()*. При вызове *onStart()* окно еще не видно пользователю, но вскоре будет видно. Вызывается непосредственно перед тем, как активность становится видимой пользователю. Сопровождается вызовом метода *onResume()*, если активность получает передний план, или вызовом метода *onStop*(), если становится скрытой.

Метод *onResume()* вызывается после *onStart(),* даже когда окно работает в приоритетном режиме и пользователь может его наблюдать. В этот момент пользователь взаимодействует с созданным вами окном. Приложение получает монопольные ресурсы. Запускает воспроизведение анимации, аудио и видео. Также может вызываться после *onPause()*.

Система вызывает этот метод каждый раз, когда активность идет на переднем плане, в том числе, при первом создании. Таким образом, мы должны реализовать *onResume()* для инициализации компонентов, регистрации любых широковещательных приемников или других процессов, которые освободили или приостановили в *onPause()* и выполнять любые другие инициализации, которые должны происходить, когда активность вновь активна.

Необходимо пытаться размещать относительно быстрый и легковесный код, чтобы приложение оставалось отзывчивым при скрытии с экрана или выходе на передний план.

Когда пользователь решает перейти к работе с новым окном, система вызовет для прерываемого окна метод *onPause()*. По сути, происходит свёртывание активности. Сохраняет незафиксированные данные. Деактивирует и выпускает монопольные ресурсы. Останавливает воспроизведение видео, аудио и анимацию. От *onPause()* можно перейти к вызову либо *onResume(),* либо *onStop()*.

Метод *onStop()* вызывается, когда окно становится невидимым для пользователя. Это может произойти при её уничтожении, или если была запущена другая активность (существующая или новая), перекрывшая окно текущей активности. Всегда сопровождает любой вызов метода *onRestart()*, если активность возвращается, чтобы взаимодействовать с пользователем, или метода *onDestroy()*, если эта активность уничтожается.

Для передачи данных между активити, в *Android SDK* существуют интенты. *Intent* представляет собой объект обмена сообщениями, с помощью которого можно запросить выполнение действия у компонента другого приложения. Несмотря на то, что объекты *Intent* упрощают обмен данными между компонентами по нескольким аспектам. В разрабатываемом приложении интенты будут использоваться для запуска активностей и получения результатов с других компонентов. Компонент *Activity* представляет собой один экран в приложении. Для запуска нового экземпляра компонента *Activity* необходимо передать объект *Intent* методу *startActivity()*. Объект *Intent* описывает операцию, которую требуется запустить, а также содержит все остальные необходимые данные.

Если после завершения операции от нее требуется получить результат, вызовите метод *startActivityForResult()*. Ваша операция получит результат в виде отдельного объекта *Intent* в обратном вызове метода *onActivityResult()* операции.

Файл манифеста инкапсулирует всю архитектуру *Android* приложения, его функциональные возможности и конфигурацию. В процессе разработки приложения приходится постоянно редактировать данный файл, изменяя его структуру и дополняя новыми элементами и атрибутами.

Корневым элементом манифеста является <*manifest*>. Помимо данного элемента обязательными элементами является теги <*application*> и <*usessdk*>. Элемент <*application*> является основным элементом манифеста и содержит множество дочерних элементов, определяющих структуру и работу приложения. Порядок расположения элементов, находящихся на одном уровне, произвольный. Все значения устанавливаются через атрибуты элементов. Кроме обязательных элементов, упомянутых выше, в манифесте по мере необходимости используются другие элементы.

Элемент <*application*> один из главних элементов файла манифеста, он содержит описание компонентов приложения, доступных в пакете: стили, значок, строки и др. Содержит дочерние элементы, которые объявляют каждый из компонентов, входящих в состав приложения. Также в файле может быть только один элемент <*application*>.

Элемент <*activity*> объявляет активность. Если приложение содержит несколько активностей, то нужно обязательно объявлять их в манифесте, создавая для каждой из них свой элемент <*activity*>. Если активность не объявлена в манифесте, она не будет доступна системе и не будет запущена при выполнении приложения или будет выводиться сообщение об ошибке. Каждый тег <*activity*> поддерживает вложенные узлы <*intentfilter*>. Элемент <*intentfilter*> определяет типы намерений, на которые могут ответить

деятельность, сервис или приемник намерений. Фильтр намерений определяет возможности его родительского компонента, в нём описано, что могут сделать деятельность или служба и какие типы рассылок получатель может обработать. Фильтр намерений предоставляет для компонентов клиентов возможность получения намерений объявляемого типа, отфильтровывая те, которые не значимы для компонента, и содержит дочерние элементы <*action*>, <*category*>, <*data*>.

Элемент <*service*> объявляет службу как один из компонентов приложения. Все службы должны быть представлены элементом <*service*> в файле манифеста. Службы, которые не были объявлены, не будут обнаружены системой и никогда не будут запущены. Этот элемент имеет много атрибутов, определяющих имя, доступность, разрешения, процесс и т.д. Поддерживает вложенные узлы <*intentfiiter*> [15].

На рисунке 2.2 изображен скриншот кода файла *AndroidManifest.xml*, используемого в программном средстве.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.eddii_mobile">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />

    <application
        android:name=".MainApplication"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:allowBackup="false"
        android:showOnLockScreen="true"
        android:theme="@style/AppTheme">
      <activity
        android:name=".MainActivity"
        android:label="@string/app_name"
        android:launchMode="singleTop"
        android:screenOrientation="portrait"
        android:configChanges="keyboard|keyboardHidden|orientation|screenSize"
        android:windowSoftInputMode="adjustResize">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
      <activity android:name="com.facebook.react.devsupport.DevSettingsActivity" />
      <service android:name=".NotificationService">
        <intent-filter>
          <action android:name="com.google.firebase.MESSAGING_EVENT" />
        </intent-filter>
      </service>
      <service android:name="io.invertase.firebase.messaging.RNFirebaseInstanceIdService">
        <intent-filter>
          <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
        </intent-filter>
      </service>
```

Рисунок 2.2 – Скриншот файла *AndroidManifest.xml*

В архитектуре *React Native* приложений имеется характерная особенность в том, что все приложение находится в единственной активности. Так как вся бизнес-логика приложения написана на языке *JavaScript*, который не является родным для мобильных платформ, то создается одна активность, в которой обрисовывается весь интерфейс приложения, а также создается отдельный поток для выполнения *JavaScript*-кода. Поэтому все

вышеперечисленные методы жизненного цикла активности применимы к разрабатываемому приложению в единственном экземпляре.

Несмотря на то, что *React Native* приложения состоят всего из одной активности, паттерн *MVC* все же остается актуален. *JavaScript*код приложения состоит из компонентов, которые имеют в себе часть, ответственную за отображение, часть – за контроллер и часть – за модель.

На рисунке 2.3 изображен скриншот, показывающий пример компонента *React Native* приложения.

```
1   import ...
10
11  const LEVEL_ICONS = {1: FirstLevel...};
15
16  export default class EnergyBar extends Component {
17    static defaultProps = {...};
20
21    constructor(props) {...}
35
36    async componentDidMount() {...}
47
48    onChangeProgress = (newEnergy, newLevel) => {...};
57
58    updateLevel = level => {...};
69
70    changeLevelAnimation = newLevel => {...};
119
120   progressAnimation = (progress, callback) => {...};
127
128   render() {
129     const { currentLevel } = this.state;
130
131     return (
132       <View style={styles.energyContainer}>
133         {!!currentLevel && (
134           <Fragment>
135             <Animated.View style={[styles.progressContainer, { transform: [{ scale:
136               <Animated.View
137                 style={[
138                   styles.partial,
139                   {
140                     width: this.animatedProgressValue.interpolate( config: {
141                       inputRange: [0, 30],
142                       outputRange: ['0%', '100%'],
143                     }),
144                   },
145                 ]}
146                 />
147             </Animated.View>
```

Рисунок 2.3 – Пример компонента *React Native* приложения

В строке 16 кода можно видеть объявление класса *EnergyBar*, который и является компонентом. В строке 128 объявляется метод класса *render()*, выполнение которого приводит к отображению части пользовательского интерфейса, за которую ответственен данный компонент. Именно эта часть компонента и отвечает за составляющую отображения в паттерне *MVC*. В

строке 48 приведенного снимка кода присутствует объявление метода *onChangeProgress()*, который выполняет функцию контроллера, а именно при определенном взаимодействии пользователя с интерфейсом приложения изменяется уровень прогресса – и данный метод выполняется как реакция на действия пользователя. Составляющая, ответственная за выполнение функций модели находится в составе конструктора класса. Это состояние компонента. Которое изменяется посредством методов-контроллеров, например, *onChangeProgress()*. Изменение состояния компонента в свою очередь приводит к его перерисовке и обновлении составляющей отображения.

Чтобы вручную не собирать проект и не прописывать множество команд, разработчики создали системы сборки проектов. В нашем случае используется *Gradle*.

*Gradle* – система автоматической сборки, построенная на принципах *Apache Ant* и *Apache Maven*, но предоставляющая *DSL* на языке *Groovy* вместо традиционной *XML*образной формы представления конфигурации проекта.

В отличие от *Apache Maven*, основанного на концепции жизненного цикла проекта, и *Apache Ant*, в котором порядок выполнения задач (*targets*) определяется отношениями зависимости (*depends*), *Gradle* использует направленный ациклический граф для определения порядка выполнения задач.

*Gradle* был разработан для расширяемых многопроектных сборок, и поддерживает инкрементальные сборки, определяя, какие компоненты дерева сборки не изменились и какие задачи, зависимые от этих частей, не требуют перезапуска [16].

На рисунке 2.4 приведен снимок экрана, отображающего конфигурацию *build.gradle* файла.

```
15  android {
16      compileSdkVersion rootProject.ext.compileSdkVersion
17      buildToolsVersion rootProject.ext.buildToolsVersion
18
19      compileOptions {
20          sourceCompatibility JavaVersion.VERSION_1_8
21          targetCompatibility JavaVersion.VERSION_1_8
22      }
23
24      defaultConfig {
25          applicationId "com.eddii_mobile"
26          minSdkVersion rootProject.ext.minSdkVersion
27          targetSdkVersion rootProject.ext.targetSdkVersion
28          versionCode 104
29          versionName "2.1.1"
30          ndk {
31              abiFilters "armeabi-v7a", "x86", "arm64-v8a", "x86_64"
32          }
33      }
```

Рисунок 2.4 – Снимок экрана с конфигурацией *build.gradle* файла

В этом файле указаны все необходимые специфичные для запуска *Android* проекта опции, такие как *defaultConfig, compileOtions* и другие. Однако в контексте *React Native* приложения важным моментом конфигурации данного файла является пятая строка, в которой указан входной файл для *JavaScript* кода. Именно с этого файла начнется выполнение приложения после того, как полностью сконфигурируется и запустится его *Android* оболочка.

## 2.2 Разработка информационной модели системы, обоснование и проектирование базы данных

В данном дипломном проекте для хранения всех пользовательских данных используется база данных *PostgreSQL*. Никакие пользовательские данные не должны храниться локально на мобильном устройстве, кроме тех токенов и ключей, которые необходимы для обеспечения более удобного входа и авторизации в приложении. Такое решение обусловлено тем, что необходимо обеспечить наименьшую вероятность потери данных пользователем при удалении, переустановке или обновлении приложения.

*PostgreSQL* – это объектно-реляционная система управления базами данных (ОРСУБД). В *PostgreSQL* появилось множество новшеств, которые были реализованы в некоторых коммерческих СУБД гораздо позднее.

*PostgreSQL* – СУБД с открытым исходным кодом, основой которого был код, написанный в Беркли. Она поддерживает большую часть стандарта *SQL* и предлагает множество современных функций:

- сложные запросы;
- внешние ключи;
- триггеры;
- изменяемые представления;
- транзакционная целостность;
- многоверсионность.

Кроме того, пользователи могут всячески расширять возможности *PostgreSQL*, например создавая свои

- типы данных;
- функции;
- операторы;
- агрегатные функции;
- методы индексирования;
- процедурные языки.

А благодаря свободной лицензии, *PostgreSQL* разрешается бесплатно использовать, изменять и распространять всем и для любых целей – личных, коммерческих или учебных [17].

Для взаимодействия между мобильным приложением и базой данных необходимо создать серверное приложение, которое будет принимать *REST*запросы от клиентских приложений, совершать запросы к базе данных и отправлять полученную информацию обратно.

Так как выбранное решение для организации базы данных основано на *SQL* таблицах, то необходимо тщательно продумать структуру и организацию таблиц данных для того, чтобы избежать совершения неэффективных запросов к базе данных и уменьшения производительности информационной системы в целом.

Для работы приложения был создан набор таблиц, которые покрывают имеющийся функционал и позволяют использовать их для реализации новых функций. Рассмотрим созданные таблицы и дадим краткую характеристику каждой из них.

Таблица *«users»* предназначена для хранения списка пользователей приложения. При окончании регистрации в приложении в эту таблицу вносится новая запись с данными, которые внес пользователь в специальной регистрационной форме. На рисунке 2.5 можно видеть, какие поля входят в состав данной таблицы.

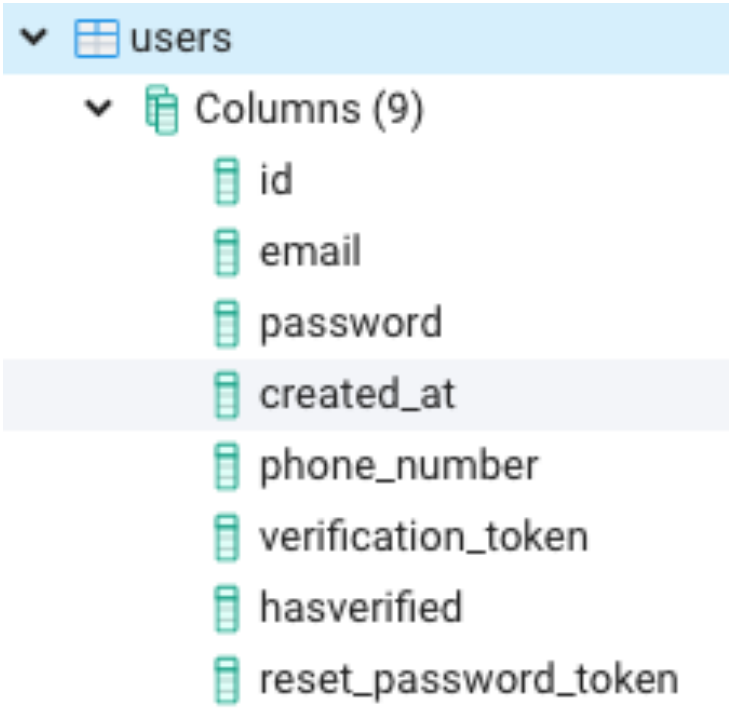



Рисунок 2.5 – Структура таблицы «*users*»

Таблица *«metrics»* предназначена для хранения вводимых пользовательских данных для учета и предоставления последующей статистики. Сохраненные в таблице данные ассоциируются с определенным пользователем посредством поля *user_id*, в котором хранится идентификационный номер пользователя из таблицы *users*. Данная таблица хранит информацию о количестве потребленных пользователем углеводов, приемов пищи, лекарственных средств и уровне глюкозы в крови, а также о количестве совершенной физической активности.

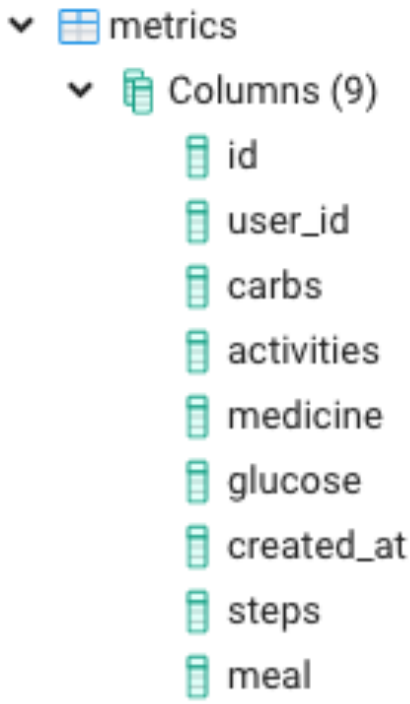



Рисунок 2.6 – Структура таблицы «*metrics*»

Таблица *«achievements_info»* содержит данные о достижениях пользователя. Так как в приложении должна быть реализована система вознаграждения и достижений, то именно эта таблица будет хранить соответствующие данные. Особенность данной таблицы в том, что она хранит промежуточные данные о прогрессе пользователя.

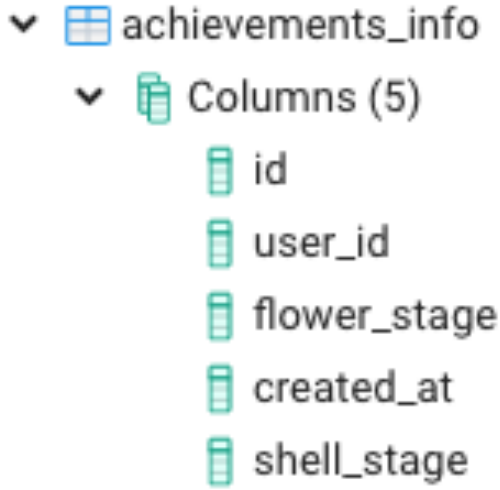



Рисунок 2.7 – Структура таблицы «*achievements_info*»

Также для слежения за «успеваемостью» пользователя необходимо создать таблицу, которая хранит данные о «наградах» пользователя. В приложении необходимо организовать по крайней мере два уровня. Для каждого уровня достижений характерен свой тип призов и наград пользователя. Например, находясь на первом уровне пользователь

награждается цветками за ввод своих данных. При переходе на второй уровень в качестве награды будет использоваться морская ракушка. Ниже на рисунке 2.8 приведены структуры двух типов наград пользователя.

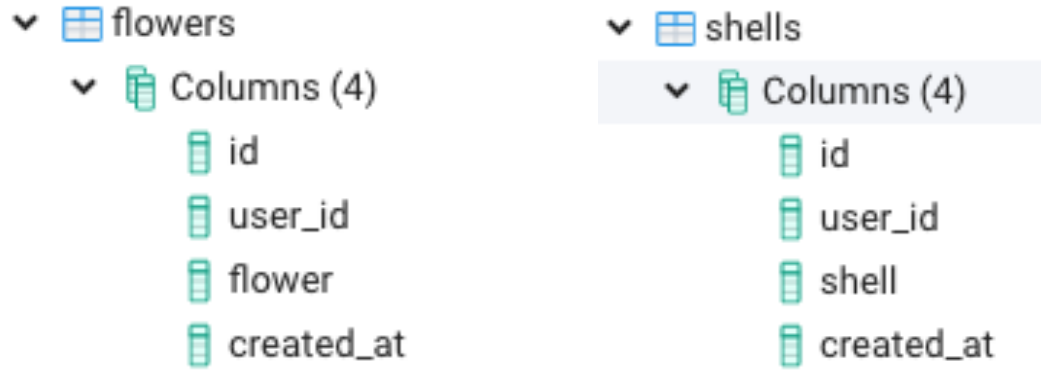



Рисунок 2.8 – Структура таблиц «*flowers*» и «*shells*»

Рассмотренные выше таблицы и их связи и отношения отображены в одном из плакатов графического материала на *ER*-диаграмме базы данных.

### 2.3 Разработка алгоритмов функционирования программного средства

Алгоритм – это точно установленное предписание (инструкция) о выполнении в определённом порядке некоторой последовательности операций, однозначно ведущих к решению той или иной конкретной задачи [19].

Предписание алгоритма представляет собой конечный набор правил, который задаёт потенциально осуществимый вычислительный процесс (процесс исполнения алгоритма), ведущий от варьирующих в определённых пределах исходных данных к получению результата, однозначно определяемого допустимыми исходными данными. Последнее подразумевает, что результат выполнения алгоритма напрямую зависит от исходных данных: то есть один и тот же алгоритм при разных исходных данных даст разные результаты; с другой стороны, если одному и тому же алгоритму передать несколько раз одни и те же данные, он должен столько же раз выдать один и тот же результат. Простейшими примерами алгоритмов являются арифметические правила сложения, вычитания, умножения, деления и тому подобные.

Предписание алгоритма, как правило, фиксируется в виде текста на некотором формализованном языке, называемого программой. Понятие программы формулируется в чисто структурных терминах синтаксиса этого языка, без какого-либо обращения к смысловым категориям. Точно такой же характер носит и описание процедуры выполнения программы. Поэтому в роли исполнителя алгоритмов, записанных на формализованных языках,

может выступать не только человек, но и наделённое соответствующими возможностями автоматическое устройство, машина.

Универсальным исполнителем алгоритмов является компьютер. С его помощью возможно выполнять все основные виды алгоритмов. Возможность машинного осуществления алгоритмических процедур, и, соответственно, машинного решения задач, стимулировала развитие вычислительной техники и создание математической теории алгоритмов.

В современной науке теория алгоритмов является основой конструктивного направления в математике и логике, а также выступает одной из базисных дисциплин в области вычислительной техники и программирования, машинного решения разнообразных задач, моделирования различных процессов и других областях.

Различают следующие виды алгоритмов [19]:

1 Линейный – список команд (указаний), выполняемых последовательно друг за другом;

2 Разветвляющийся – алгоритм, содержащий хотя бы одну проверку условия, в результате которой обеспечивается переход на один из возможных вариантов решения;

3 Циклический – алгоритм, предусматривающий многократное повторение одной и той же последовательности действий. Количество повторений обусловливается исходными данными или условием задачи.

В приложении чаще всего используется разветвляющийся вид алгоритмов, для наглядности его общая схема приведена на рисунке 2.9.

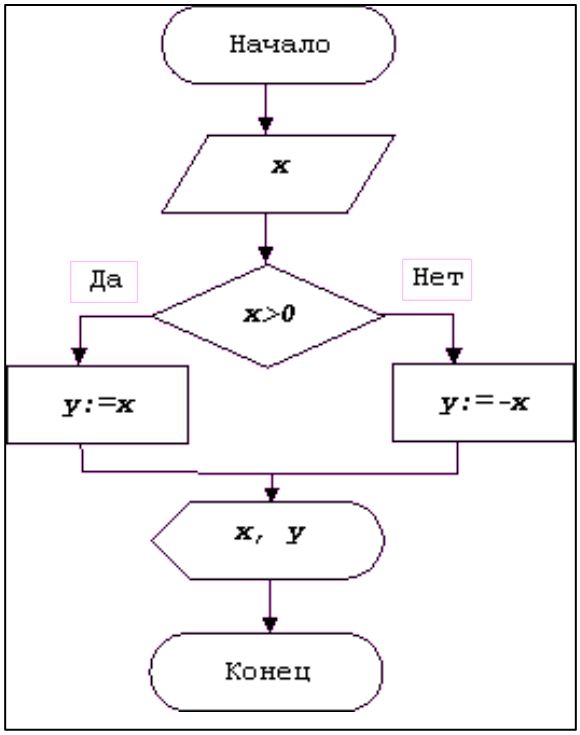



Рисунок 2.9 – Схема разветвляющегося алгоритма

На схеме наглядно проявляется важное свойство ветвящихся алгоритмов: их исполнение всегда проходит только по одному из возможных путей, который определяется конкретными текущими условиями, причем в каждом случае от начала алгоритма (входа) до его конца (выхода). Это свойство присуще всякому логически правильно составленному алгоритму и является признаком правильной организации ветвлений.

Существует широкий круг задач, при решении которых необходимо сделать определенный выбор в зависимости от выполнения некоторых условий. В разветвляющихся алгоритмах принцип линейного автоматического перехода от команды к команде не является всеобщим. Ветвящиеся алгоритмы допускают два способа представления – графический и словесный.

Система разрабатываемого программного средства будет состоять из нескольких составных частей: веб-сервер, мобильное приложение и база данных. Для лучшего понимания работы приложения необходимо рассмотреть его работу в виде отдельных шагов.

Рассмотрим алгоритм аутентификации пользователя для получения доступа входа в мобильное приложение. Схема данного алгоритма представлена в перечне графического материала.

Для получения доступа к функциям мобильного приложения пользователю необходимо зарегистрироваться и создать учетную запись либо войти в уже существующую. После запуска приложения пользователь видит экран с логотипом приложения. Пока отображается этот экран, внутри приложения происходит проверка наличия локально сохраненного в мобильном устройстве ключа, необходимого для доступа к приложению и данным пользователя. Если такой ключ присутствует, то пользователь считается прошедшим аутентификацию и перенаправляется на главный экран приложения. В случае, если ключ доступа отсутствует на мобильном устройстве, то пользователь перенаправляется на экран входа в приложение, где необходимо ввести адрес электронной почты и пароль либо воспользоваться номером мобильного телефона для входа. После отправки введенных данных на сервере происходит проверка на корректность и процесс аутентификации пользователя. Далее в качестве ответа мобильное приложение получает ключ доступа и сохраняет его локально на устройстве для организации последующего входа в приложение после его перезапуска. В конце данного процесса пользователь перенаправляется на главный экран приложения.

Далее рассмотрим алгоритм ввода пользовательских данных о здоровье, которые в последствии будут использованы для составления статистики. Схема данного алгоритма представлена в перечне графического материала.

Для ввода данных здоровья для учета пользователю необходимо открыть специально предназначенный экран, содержащий форму для ввода. Во время того, как пользователь заполняет данными форму происходит параллельный процесс анимации роста игрового предмета награды пользователя. После подтверждения отправки данных они подвергаются процессу валидации. Если введенные данные корректны, то происходит

разветвление алгоритма. В случае, если пользователь заполнил все поля ввода, предмет игровой награды считается полностью выросшим. Поэтому он добавляется в коллекцию наград, расположенную на главном экране. Также увеличивается уровень пользовательского игрового прогресса. В случае, когда после добавления новой награды и увеличения прогресса шкала энергии заполнена полностью, пользователю открывается новая игровая локация, происходит уведомление о достижении нового игрового уровня и перенаправление на главный экран приложения. В случае, когда заполняя форму ввода данных, пользователь заполнил не все поля ввода, происходит сохранение текущей стадии роста предмета игровой награды, отправка данных на сервер и закрытие экрана ввода данных.

## 2.4 Разработка и обоснование пользовательского интерфейса программного средства

Пользовательский интерфейс является принципиально важной частью продукта. Неудобный пользовательский интерфейс может оттолкнуть пользователя, снизить эффективность использования веб-сервиса. Для разработки пользовательского интерфейса обычно привлекают дизайнера.

В связи с особенностями программного продукта пользовательский интерфейс также должен быть прост в использовании – расположение некоторых элементов, цветовая гамма и т.п. должны соответствовать сценария использования приложения через сенсорный экран мобильного устройства.

На сегодняшний день существует множество рекомендаций по проектированию пользовательского интерфейса, а также готовых решений, которые можно просто настроить под определенное приложение. Например, *Android* предоставляет следующие элементы для построения приложений в соответствии с концепцией *Material Design* [18]:

- новую тему;
- новые виджеты для сложных представлений;
- новые *API*-интерфейсы для нестандартных теней и анимаций.

*Material Design* представляет собой комплексную концепцию создания визуальных, движущихся и интерактивных элементов для различных платформ и устройств. Теперь *Android* включает в себя поддержку приложений с элементами *Material Design*. Чтобы использовать элементы *Material Design* в своих приложениях под *Android*, руководствуйтесь инструкциями в спецификации *Material Design*, а также воспользуйтесь новыми компонентами и функциями, доступными в *Android* 5.0 (уровень *API* 21) и выше.

Однако такие решения подходят для типового мобильного приложения, которое использует элементы интерфейса, предоставляемые операционной системой. В случае какого-либо приложения с игровой логикой подобные решения далеко не всегда будут являться актуальными. Именно в таком положении и находится приложение, разрабатываемое в данном дипломном проекте. Поэтому в таком случае необходимо прибегнуть к помощи дизайнера.

Интерфейс программного средства должен удовлетворять следующим требованиям:

– интерфейс должен быть интуитивно понятен и не вызывать замешательства;

– в интерфейсе должен присутствовать доступ к наиболее необходимым для пользователя частям программного средства;

– интерфейс должен располагать цветовой гаммой, не раздражающей глаз и не вызывающей чувство отвращения, цвета должны быть спокойных тонов;

– интерфейс должен максимально использовать свободное пространство экрана мобильного устройства, но не должно быть нагромождения;

– использование программного средства должно быть понятно пользователю без специальной подготовки и не должно требовать прохождения предварительного обучения в использовании программы;

– расположение элементов, стиль иконок и поведение интерфейса должно следовать устоявшимся стандартам разработки программ для системы *Android*, для того чтобы программное средство не выбивалось из визуального стиля системы.

Для создания пользовательского интерфейса в приложениях на платформе *React Native* используется такой же подход, как и в разработке с библиотекой *React*. Этот подход предполагает написание кода разметки компонентов и последующее добавление стилей по принципу, схожему с веб-разработкой. Ниже будут рассмотрены результаты проектирования пользовательского интерфейса основных экранов приложения.

При первом открытии приложения пользователю необходимо создать учетную запись и зарегистрировать ее, используя адрес электронной почты и номер мобильного телефона. Номер телефона пользователь может не указывать, а использовать только адрес почты. Также необходимо создать пароль для защиты учетной записи от взлома.

На рисунке 2.10 слева изображен интерфейс экрана регистрации нового пользователя. Так как использование номера мобильного устройства является опциональным, то необходимо было реализовать возможность отмены его использования. Кроме кнопки подтверждения формы на экране можно видеть две дополнительные кнопки, отвечающие за переход к экрану входа, в случае, когда пользователь уже имеет учетную запись, и переход к экрану сброса пароля. Интерфейс данных экранов практически аналогичен экрану регистрации, поэтому их скриншоты не приведены в данном описании.

После прохождения пользователем процесса аутентификации он перенаправляется к главному экрану приложения. Изображение данного экрана можно видеть на рисунке 2.10 справа.

Так как в приложении реализована игровая логика, то и интерфейс соответственно будет не стандартным. На рисунке 2.10 справа изображена игровая локация первого уровня – парк. Также на рисунке присутствует игровой персонаж, сидящий на ветке дерева. Вверху экрана находится панель

прогресса пользователя, кнопка переключения локаций и кнопка входа в настройки приложения. Внизу данного экрана находится кнопка перехода в меню навигации. Также внизу экрана располагается круговая диаграмма, которая отображает текущую статистику уровня глюкозы в крови пользователя на основе введенных им данных. При нажатии на данную диаграмму происходит навигация приложения к экрану с более подробной статистикой. А при нажатии на иконку редактирования пользователю будет отображена форма ввода новых данных для учета здоровья.

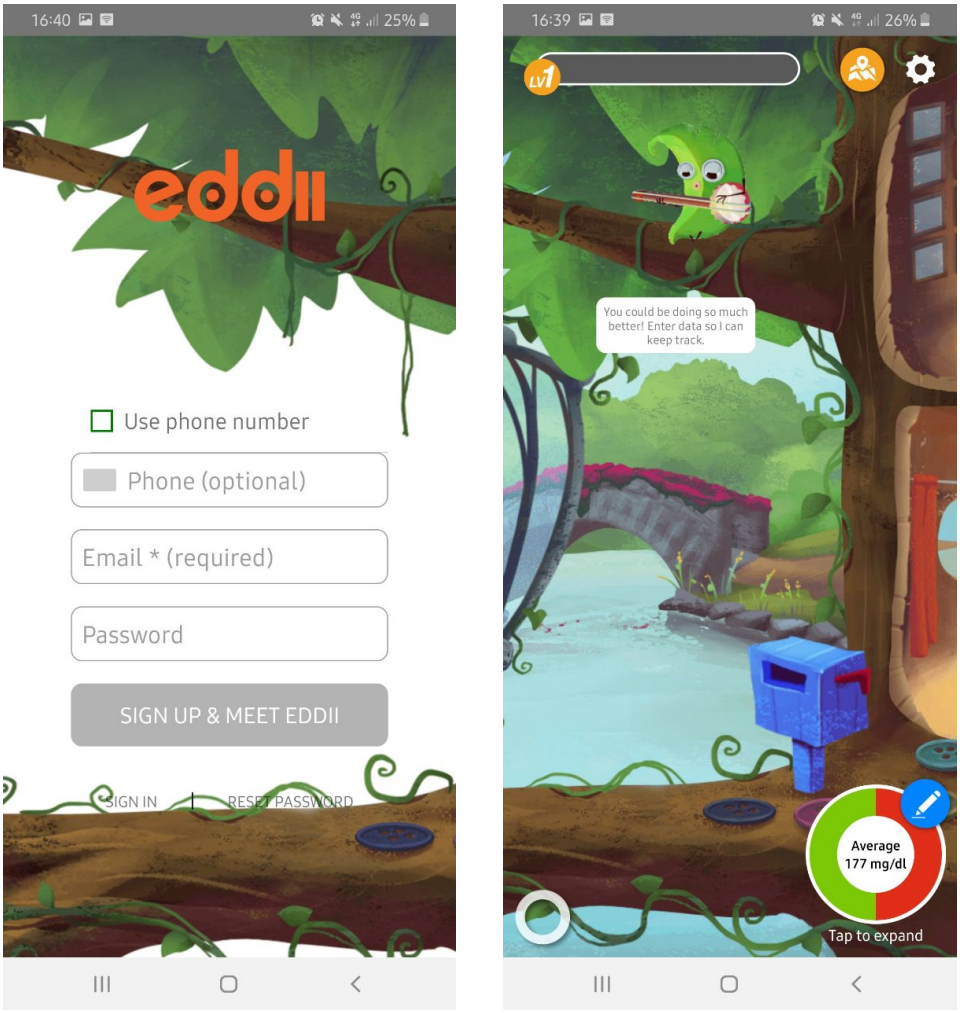



Рисунок 2.10 – Экраны регистрации и главный экран приложения

На рисунке 2.11 слева представлено окно переключения игровых локаций. Новые локации становятся доступными по мере игрового прогресса пользователя, а точнее по мере количества введенных данных. При переключении локации произойдет перенаправление на главный экран приложения. Отличие будет заключаться в том, что тема оформления главного экрана будет перерисована в соответствии с новой локацией.

Как уже было описано выше, при нажатии на круговой график пользователь перенаправляется на подробную статистику своего здоровья. На открывшемся экране отображаются все данные, которые пользователь когда-

либо вносил. Данные отображаются в отсортированном по дням недели формате. Здесь пользователь может ознакомиться с информацией о его уровне глюкозы в крови, принятом количестве лекарственных средств, времени физической активности и количестве потребленных калорий.

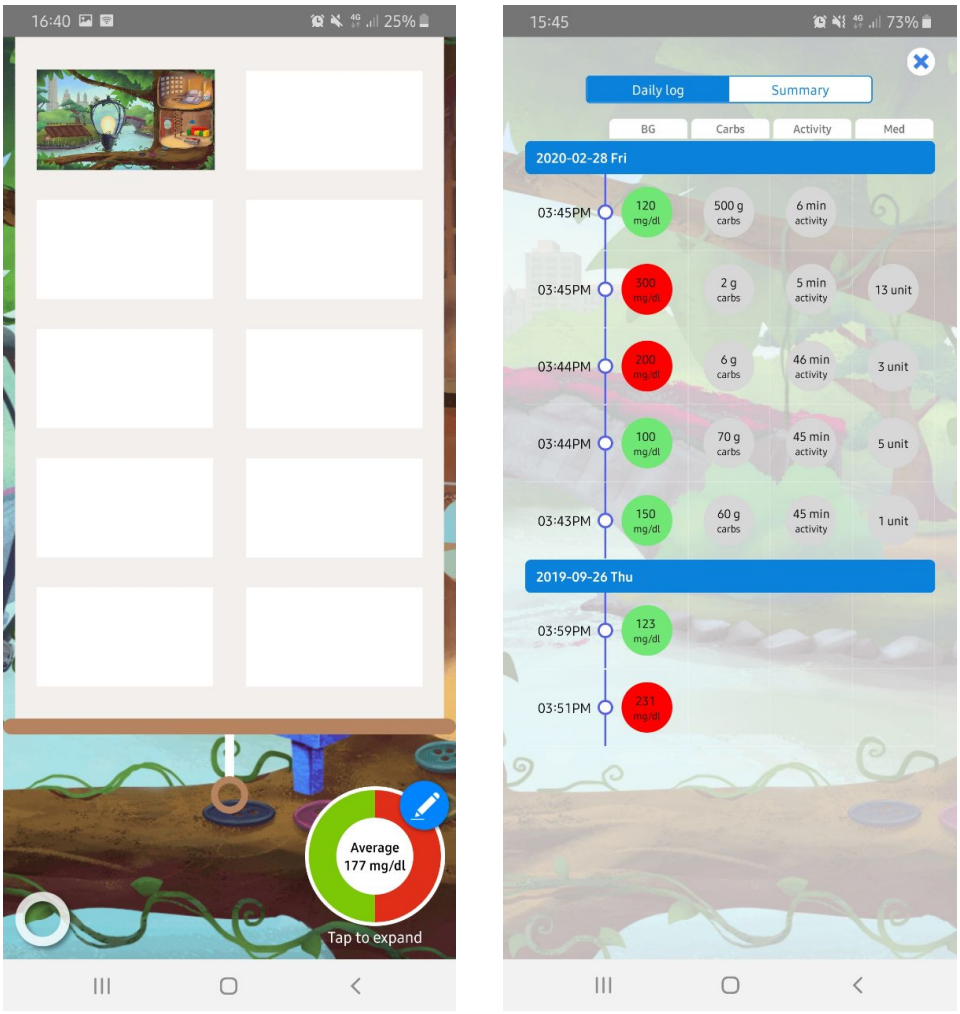



Рисунок 2.11 – Экраны переключения игровой локации и подробной статистики

На рисунке 2.12 слева изображен экран ввода новых данных о здоровье пользователя.

На экране ввода данных находится простая форма для внесения информации. Однако особенностью данного экрана является то, что на нем также присутствует цветок. Цветок является игровой наградой пользователя за ввод данных. Логика заключается в том, что чем больше полей формы заполнит пользователь, тем быстрее вырастет цветок. Полностью выросший цветок отправляется на «клумбу» главного экрана. Если же пользователь уже завершил первый уровень, то в качестве награды будет выращиваться уже другой тип предмета.

На рисунке 2.12 справа изображен экран чата. Он предназначен для имитации общения пользователя с игровым персонажем посредством онлайн-переписки.

Вверху данного экрана также отображен текущий прогресс пользователя. Пользователю предоставляется выбрать ответное сообщение посредством нажатия кнопки. Игровой персонаж, в свою очередь, будет отвечать на сообщения пользователя.

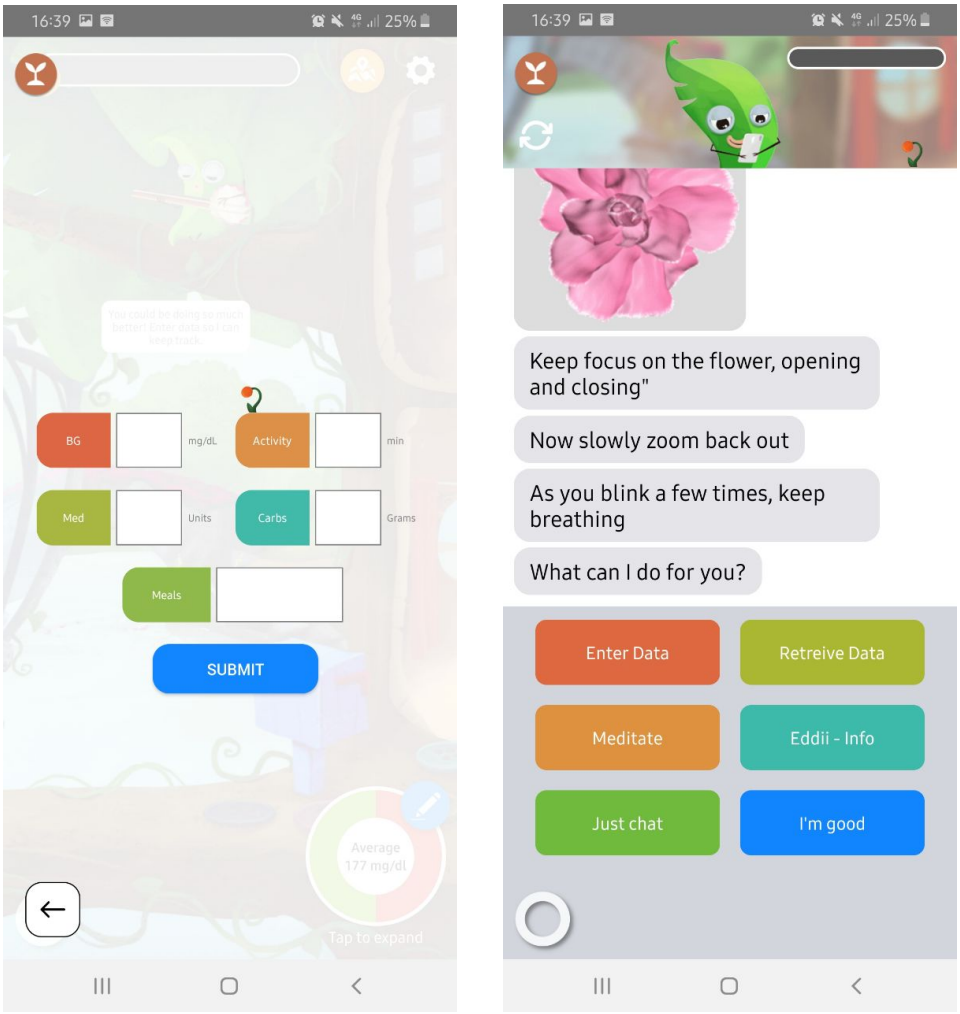



Рисунок 2.12 – Экраны ввода данных и чата

Интерфейс проектировался таким образом, чтобы на экране устройства отображалась самая необходимая для пользователя информация и функции. Также получились интуитивно понятное меню и навигация в приложении. Реализация доступа к широким функциональным возможностям и обеспечение простоты работы противоречат друг другу. Разработка эффективного интерфейса призвана сбалансировать эти цели [25]. Таким образом был разработан пользовательский интерфейс, отвечающий требованиям, описанным в начале подраздела.

# 3 ИНЖЕНЕРНЫЕ РАСЧЕТЫ, ИСПОЛЬЗУЕМЫЕ В ПРОГРАММНОМ СРЕДСТВЕ

## 3.1 Расчет среднестатистического значения уровня глюкозы в крови пользователя на основе введенных данных

Расчет среднестатистического значения уровня глюкозы в крови пользователя в приложении может рассчитываться для различных промежутков времени. Например, для выдачи статистики за последнюю неделю программное средство выбирает только те данные из базы, которые были введены за последнюю неделю. Специально для этого при любом вводе данных заполняется соответствующее поле таблицы, содержащее дату и время введения данных. В некоторых случаях расчет среднестатистического значения уровня глюкозы в крови может производиться для конкретного запрошенного дня. Однако, каким бы ни был промежуток времени для расчета, ход вычислений остается тот же. Сперва производится выборка данных, соответствующих нужной дате, а затем для расчета среднестатистического значения уровня глюкозы в крови используется следующая формула:

$$G_{\text{с}}=\frac{\sum_{i=1}^{n} G_i}{n}, \tag{3.1}$$

где $G_{\text{с}}$ – среднее значение уровня глюкозы в крови;

$G_i$ – определенное значение уровня глюкозы из выборки;

$n$ – количество значений в выборке данных.

Для наглядности произведем вычисления на примере и сравним их с результатом, отображаемым в приложении.

На рисунке 3.1 изображены снимки двух экранов. Слева располагается статистика введенных данных по дням недели, а справа – среднестатистические данные за неделю. Для проверки данных, рассчитанных приложением и предоставленных справа на рисунке, проведем вычисления по формуле 3.1. На момент проведения вычислений данными, введенными за последнюю неделю, являются последние пять записей таблицы на рисунке слева. Это значения 120, 300, 200, 100, 150. Проведем вычисления:

$$G_{\text{с}}=\frac{120+300+200+100+150}{5}$$

Из вышеприведенного уравнения получаем результат

$$G_{\text{с}} = 174$$

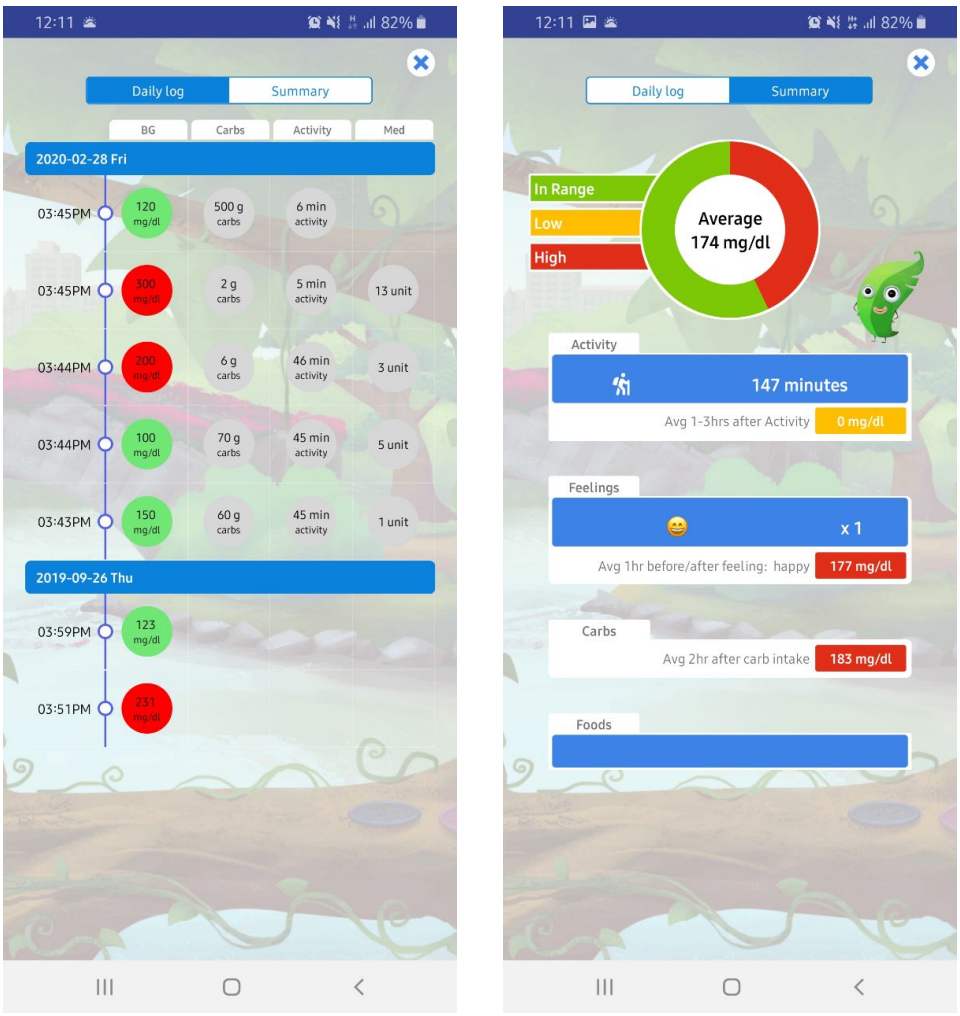



Рисунок 3.1 – Пример результатов вычислений в приложении

Полученный в результате проделанных вычислений результат соответствует ожидаемому, что позволяет сделать вывод о корректности расчета среднестатистических данных уровня глюкозы в крови в приложении.

### 3.2 Анализ памяти, используемой программным средством

Так как разрабатываемое в данном дипломном проекте программное средство имеет в своей основе фреймворк для кроссплатформенной разработки, то одним из важных показателей приложения является количество памяти устройства, занимаемое данным приложением. Зачастую программные средства для мобильных устройств, основанные на кроссплатформенных технологиях, занимают в конечном результате чрезвычайно большой объем памяти устройства, что накладывает ограничения на количество потенциальных установок мобильного приложения.

В данном анализе будут рассмотрены три устройства, работающие на разных версиях ОС *Android*, которые наиболее распространены на сегодняшний день, а именно 7, 8 и 10 версии операционной системы.

На рисунках 3.2, 3.3 и 3.4 изображены скриншоты сводки информации об объеме памяти, занимаемой приложением на седьмой, восьмой и десятой версиях ОС *Android* соответственно.

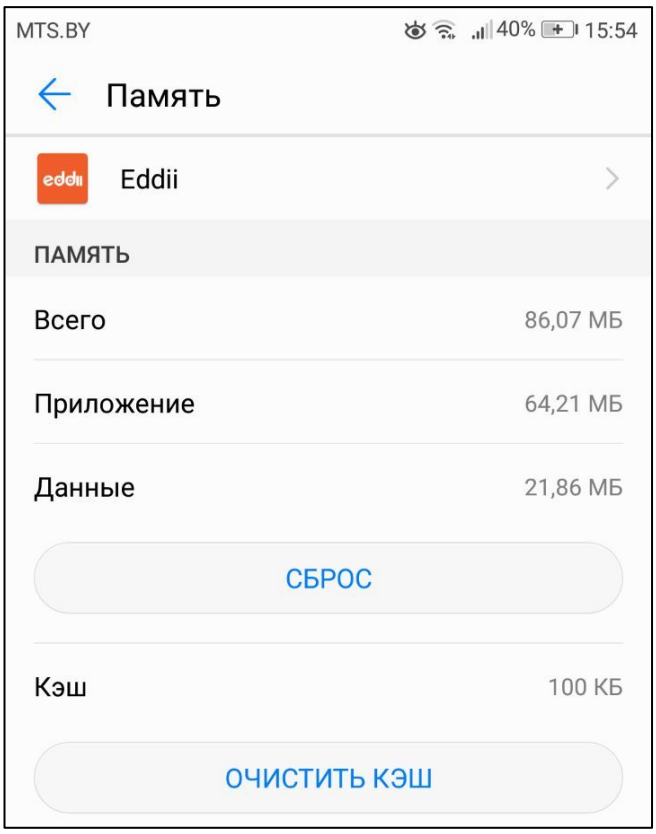



Рисунок 3.2 – Сводка информации об объеме памяти на *Android* 7


Рисунок 3.3 – Сводка информации об объеме памяти на *Android* 8

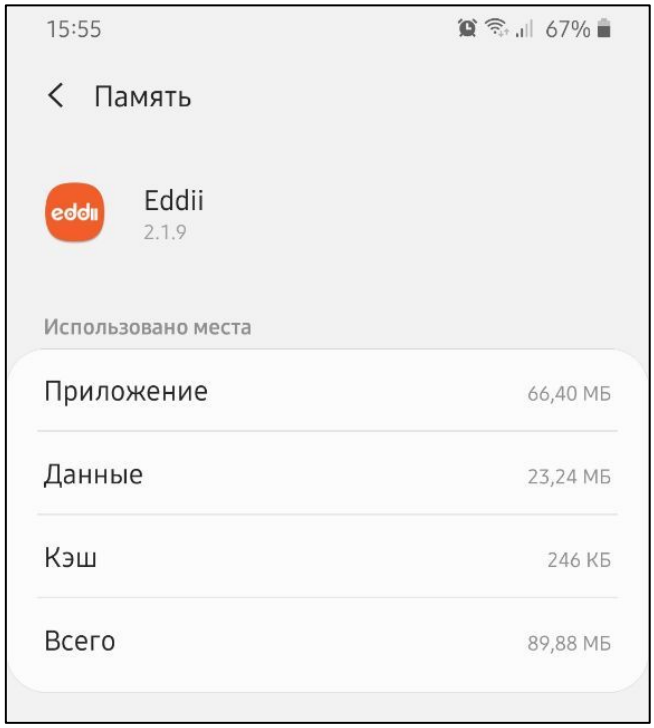



Рисунок 3.4 – Сводка информации об объеме памяти на *Android* 10

Как видно из приведенных выше изображений, объем памяти мобильного устройства, занимаемый приложением, колеблется от 86 до 90 мегабайт. Такое количество памяти не является идеальным для приложения с достаточно простыми функциональными возможностями. Однако эти объемы все же приемлемы для пользователя, так как на сегодняшний день смартфоны обладают мощностями и объемами памяти, сопоставимыми с характеристиками настольных компьютеров двухлетней давности. Также не стоит забывать о возможностях, которые предоставляет кроссплатформенная технология, а именно реализация приложения для второй мобильной операционной системы – *iOS*. Такая реализация будет проходить практически без дополнительных усилий и при минимальных временных затратах.

Анализ памяти, используемой программным средством, показал, что проектируемое программное средство использует, хотя и достаточно большой, но приемлемый для потенциальных пользователей объем памяти. Данные показатели характерны для большинства приложений, присутствующих на сегодняшний день на рынке.

# 4 ЭКСПЛУАТАЦИЯ ПРОГРАММНОГО СРЕДСТВА

## 4.1 Ввод в эксплуатацию программного средства

Программное средство уже можно считать полностью готовым к эксплуатации. Однако в течение первого месяца приложение должно быть использовано ограниченным кругом пользователей для проведения альфа-тестирования в качестве внутреннего приёмочного тестирования. Затем необходимо организовать бета-тестирование с привлечением добровольцев из числа обычных будущих пользователей продукта, которым будет доступна предварительная версия продукта (так называемая бета-версия).

После окончания упомянутых стадий тестирования и исправления ошибок приложение можно публиковать в сервисе *Google Play*.

Основным способом распространения *Android* приложений является публикация в таких магазинах как:

- *Google Play*;
- *Amazon Appstore*;
- *Yandex market*;
- *Galaxy Market*;
- *Badoo Shop*;
- *Opera Mobile Store.*

Разработчик выкладывает программное средство в общий доступ, после чего потенциальный пользователь может бесплатно или за некоторую плату скачать приложения. На сегодняшний день лидирующую позицию занимает магазин *Google Play* и *Amazon Appstore*.

*Amazon Appstore* – альтернативный магазин приложений для *Android*. В нем гораздо меньше контента, чем в маркете *Google Play*, но зато есть большое количество эксклюзивов. Кроме того, *Amazon* очень часто проводит промоакции и раздает десятки платных приложений и игр совершенно бесплатно.

Изначально Amazon попыталась выложить свой магазин в *Google Play*, но встретила непонимание со стороны *Google*. Приложение было удалено из официального маркета *Android*, но его можно установить с сайта *Amazon* [20].

*Google Play* (предыдущее название – *Android Market*) – магазин приложений, игр, книг, музыки и фильмов компании *Google* и других компаний, позволяющий владельцам устройств с операционной системой *Android* устанавливать и приобретать различные приложения (владельцам *Android*устройств из Соединённых Штатов и России также доступно приобретение на *Google Play* книжных изданий, музыки, фильмов и периодики). Учётная запись разработчика, которая даёт возможность публиковать приложения, стоит $25. Платные приложения могут публиковать разработчики не из всех стран [21].

Для получения возможности размещать приложения в *Google Play*, необходим профиль разработчика.

С помощью промостраницы разработчика вы можете продвигать свой бренд и приложения в *Google Play*.

Если вы опубликовали в *Google Play* хотя бы одно приложение, то можете создать промостраницу разработчика. Пользователи будут заходить на нее в *Play* Маркете или по специальному *URL*, которым вы поделитесь с ними. На промостранице будет размещена информация о вашем бренде и приложениях, которые вы опубликовали в *Google Play* [22].

Любое приложение, публикуемое в магазине *Google Play*, должно иметь подписанный сертификат. Сертификат позволяет идентифицировать вас как автора программы. И если кто-то попытается выложить программу с таким же именем как у вас, то ему будет отказано из-за конфликта имён. Под именем приложения имеется в виду полное название пакета.

Чтобы подписать приложение уникальным сертификатом необходимо в *Android Studio* выбрать меню *Build – Generate Signed Bundle / APK*. Появится диалоговое окно мастера, как на рисунке 4.1, которое необходимо заполнить данными.

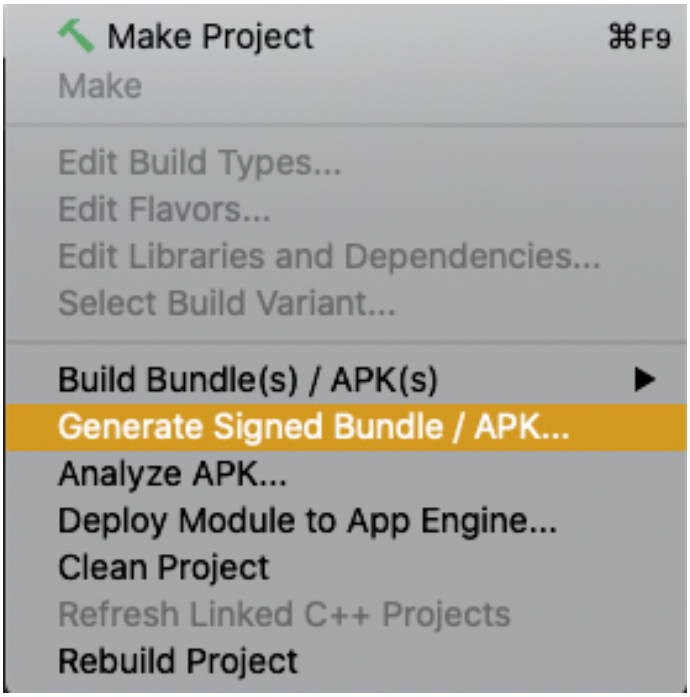



Рисунок 4.1 – Генерация подписанного приложения

Так как в 2018 году сервис *Google Play* перешел от публикации через файлы с расширением *«.apk»* к *Android App Bundle* с расширением *«.aab»*, то необходимо выбрать первый пункт в меню, отображенном на рисунке 4.2.

После выбора способа генерации сборки приложения необходимо произвести подпись этой сборки. Для этого генерируется специальный файл в бинарном формате, содержащий подпись и пароли к этой подписи. Этот этап изображен на рисунке 4.3.

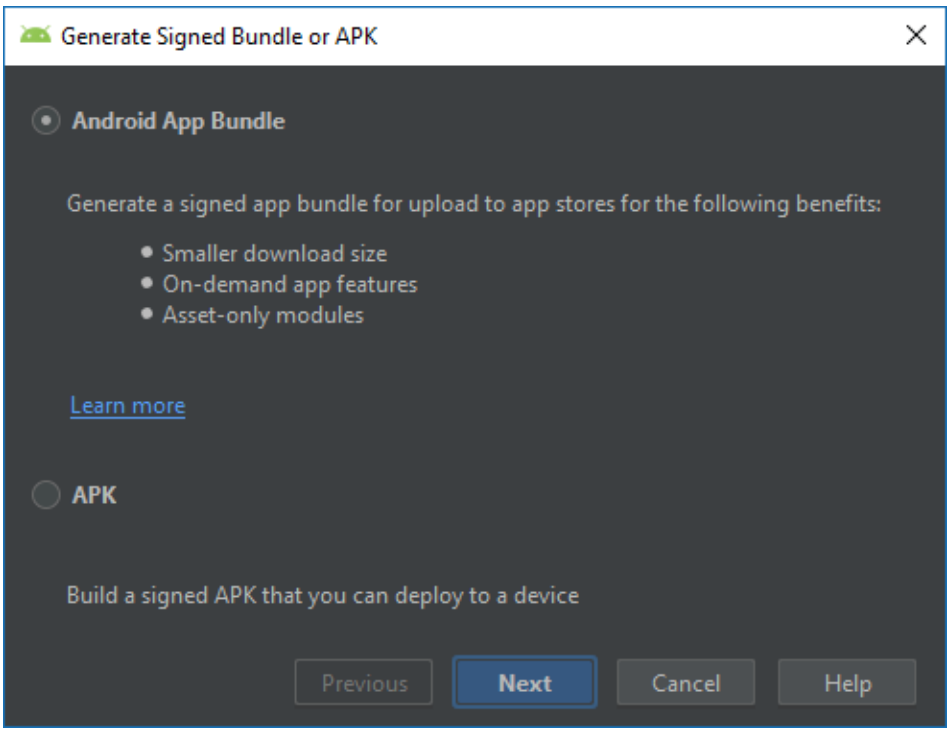



Рисунок 4.2 – Выбор генерации сборки приложения

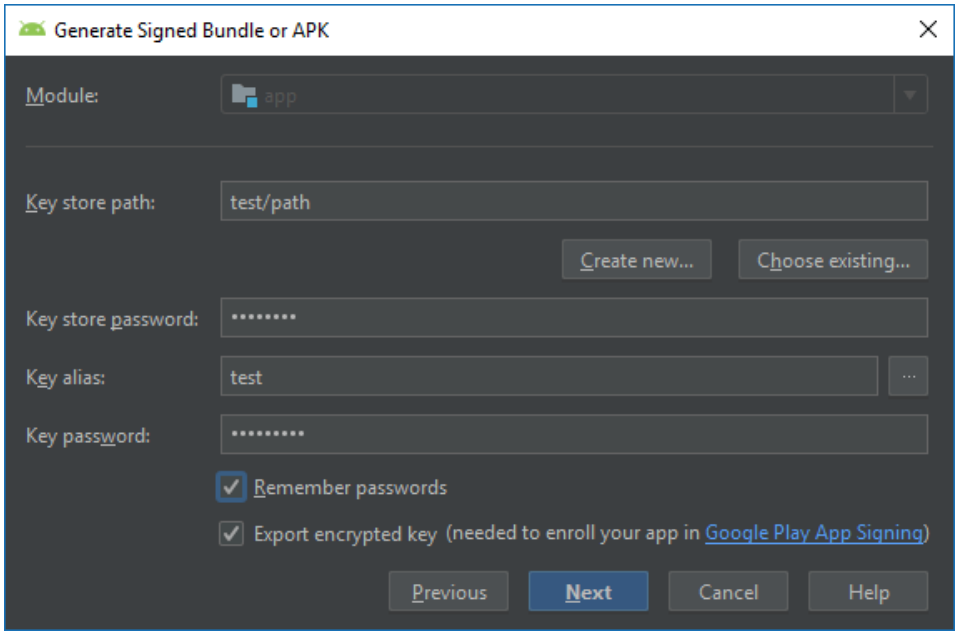



Рисунок 4.3 – Подпись сборки приложения

После подписи приложения разработчик должен указать, что необходимо сгенерировать версию приложения для публикации и начать процесс генерации сборки приложения.

В результате описанного выше процесса разработчик получает файл сборки приложения, который готов к тому, чтобы быть опубликованным в сервисе *Google Play*.

### 4.2 Руководство к пользованию разработанным программным средством

В руководстве пользователя будет объяснено, как пользоваться программным средством и какие функции оно имеет. Перед началом использования программы пользователю необходимо установить её на свой смартфон, который работает под системой *Android* версии не ниже 5.0.

При первом запуске в приложении не будет сохранено никаких данных о пользователе, поэтому пользователь сразу будет перенаправлен на экран входа и регистрации. Для того, чтобы получить полный доступ к функциям приложения, пользователю необходимо пройти процесс знакомства с приложением и процесс аутентификации.

Как только пользователь впервые откроет приложение, ему будет предложено пройти процесс знакомства с приложением. Данный экран приложения изображен на рисунке 4.4 слева

Посредством общения с чат-ботом пользователь вносит данные о себе, такие как имя, тип диабета, которым он болеет, возраст, тип лекарственных средств и время их принятия. Окно установки времени напоминаний о принятии лекарственных средств изображено на рисунке 4.4 справа.

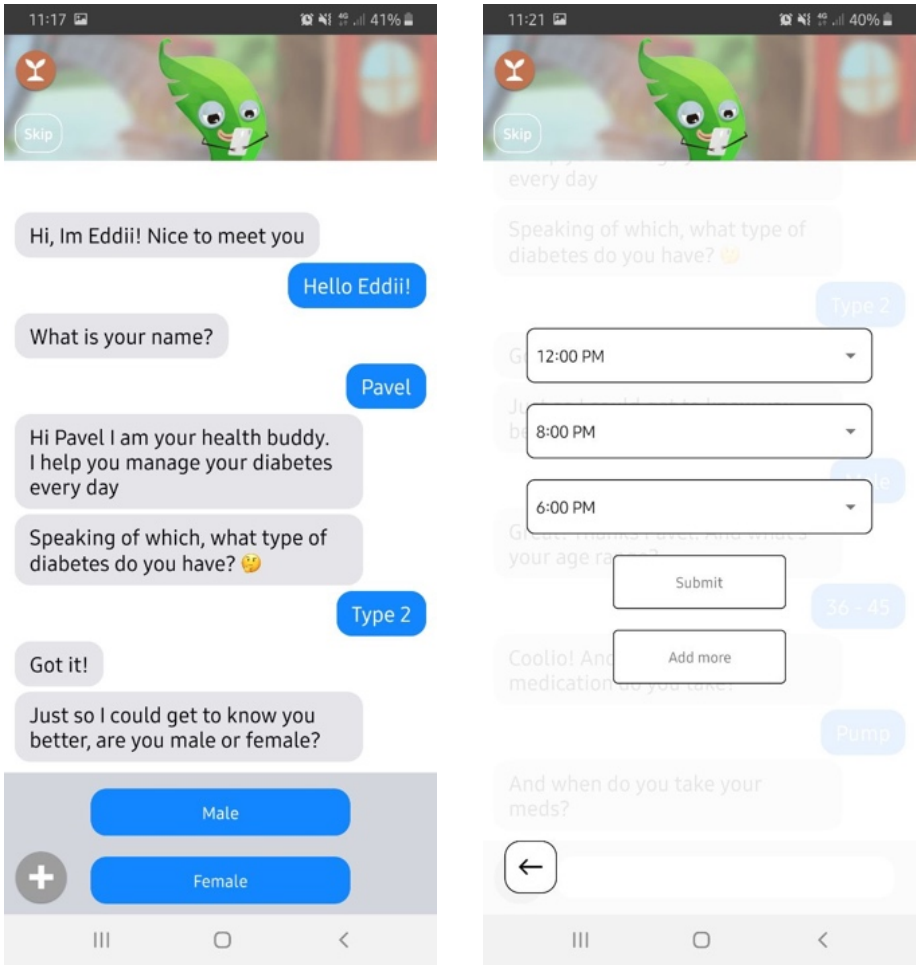



Рисунок 4.4 – Экраны знакомства с приложением и установки уведомлений

Далее пользователь вводит промежуток значений уровня глюкозы в крови, который является нормальным для него. Также пользователь отвечает на вопрос, каким способом он замерят уровень глюкозы в крови. После этого чат-бот дает краткие пояснения о работе приложения и предлагает создать учетную запись пользователя на основе уже указанных данных. Изображение этого экрана расположено на рисунке 4.5 слева.

После прохождения обучения приложения и создания учетной записи пользователь перенаправляется на главный экран программного средства.

Пользователь имеет возможность пропустить прохождение процесса ознакомления с приложением и сразу создать учетную запись или войти.

Для того, чтобы зарегистрироваться в приложении, пользователю необходимо ввести свой адрес электронной почты и придумать пароль длиной не менее шести символов. Также опционально при регистрации можно внести номер своего мобильного телефона, что в последующем позволит войти в приложение без использования электронного адреса.

В случае, если пользователь уже имеет учетную запись в приложении, необходимо перейти на экран входа. Для этого нужно нажать кнопку *«SIGN IN»*. Экран входа в приложение изображен на рисунке 4.5 справа. Пользователь может использовать электронный адрес и пароль, либо посредством нажатия кнопки *«Use phone number»* перейти на экран входа с использованием номера мобильного телефона. В этом случае пароль не понадобится – пользователь получит смс-сообщение с одноразовым кодом для доступа к приложению.

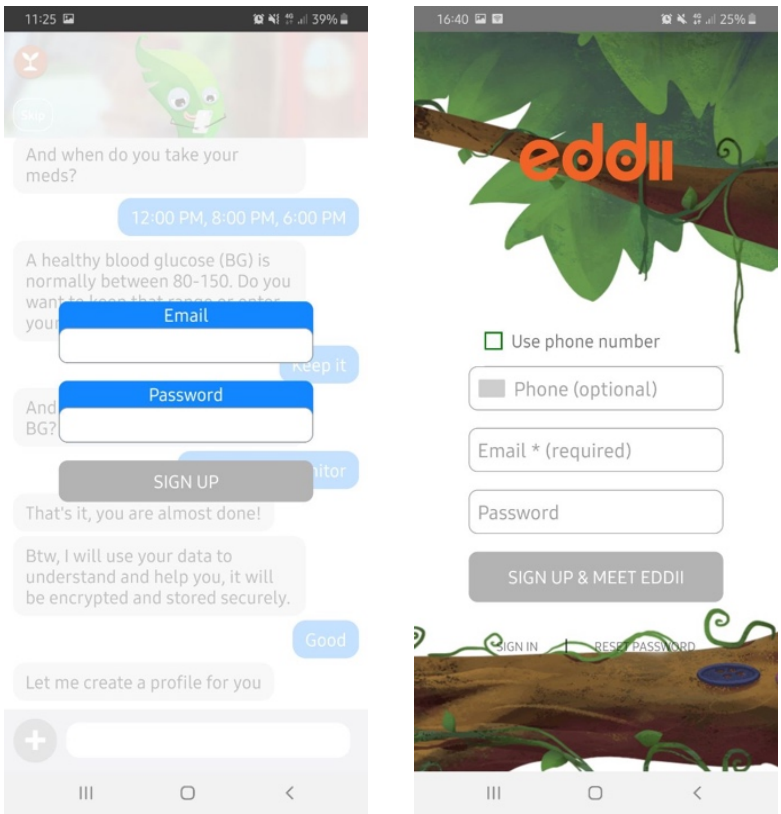



Рисунок 4.5 – Экраны создания учетной записи и регистрации

После входа в приложение пользователь направляется на главный экран, откуда имеет возможность воспользоваться основными функциями приложения.

В правом нижнем углу находится синяя кнопка, нажав которую пользователь увидит форму для внесения учетных данных о показателях здоровья. Она изображена на рисунке 4.6 справа.

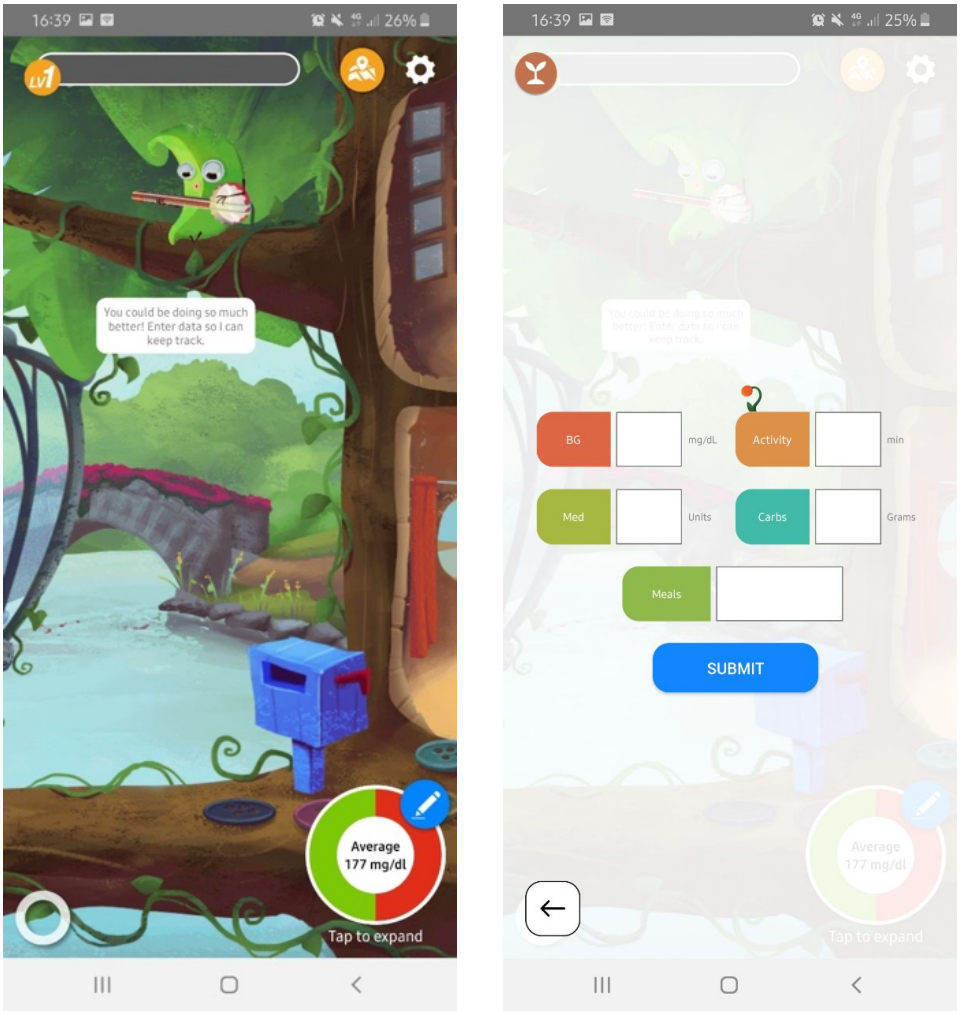



Рисунок 4.6 – Главный экран приложения и экран ввода данных

Чем больше данных для отслеживания вносит пользователь, тем быстрее вырастает его игровое вознаграждение. После того, как предмет вознаграждения полностью вырос, он отправляется в коллекцию, расположенную на главном экране.

В левом нижнем углу любого экрана приложения находится меню, откуда можно перейти к любому экрану в программном средстве. Оттуда можно перейти к экрану чата с игровым персонажем, где посредством нажатия на кнопки предлагаемых ответов можно осуществлять общение с чат-ботом.

В правом верхнем углу главного экрана располагается кнопка перехода к настройкам приложения, личной учетной записи и другим.

На рисунке 4.7 слева изображен экран редактирования персональных данных пользователя.

На рисунке 4.7 справа изображен экран настроек приложения, где можно контролировать звуковые настройки и настройки принятия уведомлений.

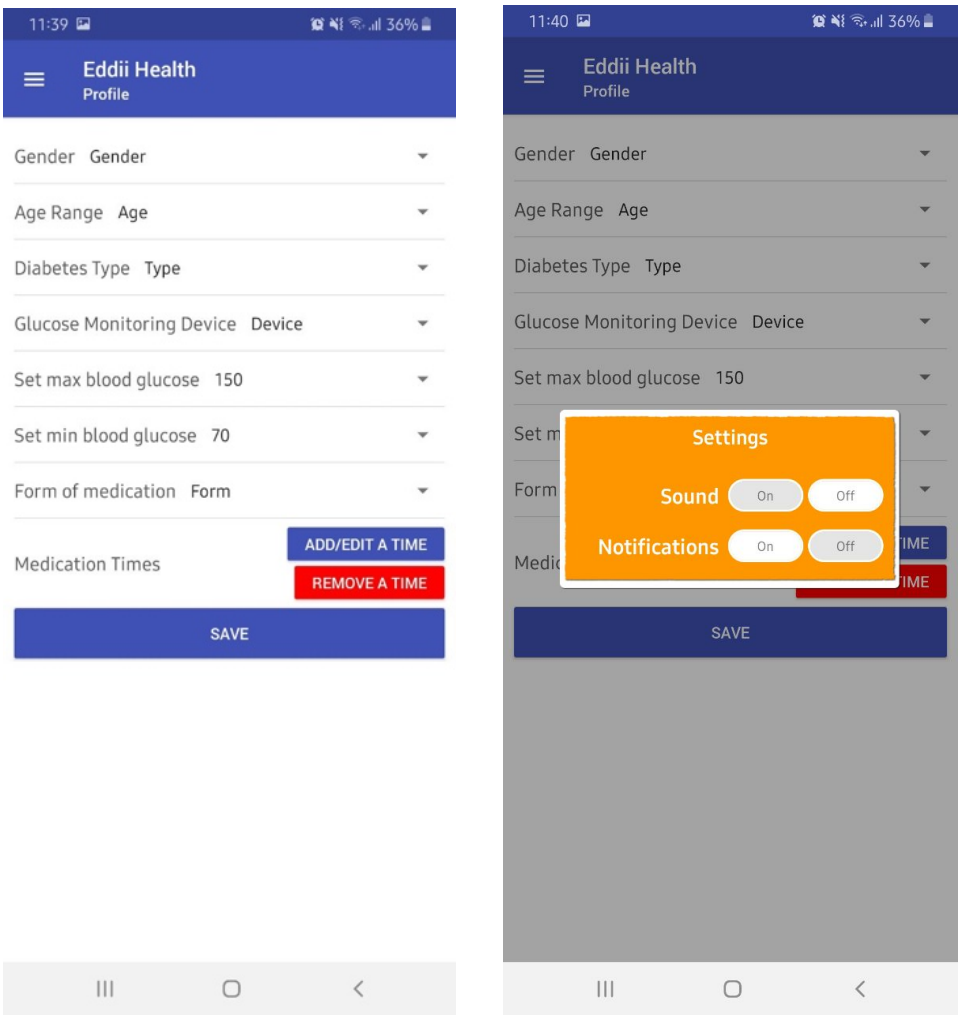



Рисунок 4.7 – Редактирование персональных данных и настройки приложения

Для выхода из приложения необходимо перейти к меню настроек и выбрать соответствующий пункт.

# 5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА ДЛЯ ОТСЛЕЖИВАНИЯ ДАННЫХ О СОСТОЯНИИ ЗДОРОВЬЯ БОЛЬНЫХ САХАРНЫМ ДИАБЕТОМ ЛЮДЕЙ

## 5.1 Краткая характеристика программного средства

Разработанное приложение представляет собой мобильное *Android* приложение, помогающее людям, больным сахарным диабетом, организовать учет своего состояния здоровья для поддержания здорового образа жизни и слежения за своим заболеванием.

Программное средство позволяет пользователю:

– вводить данные об уровне глюкозы в крови и других характеристиках своего здоровья;

– получать статистику различной степени подробности на основе введенных данных;

– назначать получение напоминаний посредством уведомлений о необходимости принятия лекарственных средств;

– взаимодействовать с игровым персонажем приложения посредством чата;

– оставаться вовлеченным в процесс отслеживания своего здоровья благодаря системе внутри-игровых достижений.

Разработанное программное средство подлежит свободной реализации на рынке информационных технологий [24]. Основное его предназначение – свободная продажа пользователям.

Программный продукт разработан сотрудниками отдела разработки программного обеспечения предприятия ООО «Техартгруп». Предприятие является компанией-резидентом ПВТ Республики Беларусь.

## 5.2 Расчет затрат на основную заработную плату команды разработчиков

Для расчёта основной заработной платы исполнителей данного проекта применяется следующая формула [24]:

$$З_о = К_{пр} \sum_{i=1}^{n} З_{ч\,i} \cdot t_i, \quad (5.1)$$

где $К_{пр}$ – коэффициент премий, установленный за выполнение, перевыполнение плановых показателей ($К_{пр} = 1{,}5$);

$n$ – количество исполнителей, занятых разработкой ПО;

$З_{ч\,i}$ – часовая заработная плата $i$-го исполнителя, руб.;

$t_i$ – трудоемкость работ, выполняемых $i$м исполнителем, ч.

Месячная заработная плата каждого из исполнителей определена по фактическим данным предприятия ООО «Техартгруп», на котором проходилась преддипломная практика. Трудоемкость работ была определена в соответствии с сроком разработки проекта и количеством работы того или иного специалиста. Срок разработки проекта равен 4 месяцам, количество рабочих часов в месяце было принято равным 168 часам. Размер премии определен исходя из практики, сложившейся на предприятии ООО «Техартгруп», и равен 50%.

Таблица 5.1 – Исполнители разрабатываемого проекта

| Исполнитель | Трудоемкость работ, ч | Месячная заработная плата, р. |
|---|---|---|
| Старший программист руководитель | 672 | 2800 |
| Программист | 672 | 2100 |
| Дизайнер | 336 | 1600 |
| Тестировщик | 260 | 1800 |

Часовая заработная плата рассчитывается путем деления месячной заработной платы каждого исполнителя на количество рабочих часов в месяце.

$$З_{\text{ч ст.прогр}} = \frac{2800}{168} = 16{,}7 \text{ руб./ч},$$

$$З_{\text{ч прогр}} = \frac{2100}{168} = 12{,}5 \text{ руб./ч},$$

$$З_{\text{ч дизайнер}} = \frac{1600}{168} = 9{,}5 \text{ руб./ч},$$

$$З_{\text{ч тест}} = \frac{1800}{168} = 10{,}7 \text{ руб./ч}.$$

Далее по формуле 5.1 рассчитываем основную заработную плату исполнителей:

$$З_{\text{о}} = 1{,}5 \cdot (16{,}7 \cdot 672 + 12{,}5 \cdot 672 + 9{,}5 \cdot 336 + 10{,}7 \cdot 260) = 38394{,}6 \text{ руб.}$$

В результате проведенных вычислений затраты на основную заработную плату команды разработчиков составили 38394,6 рублей.

### 5.3 Расчет дополнительной заработной платы исполнителей

В данном разделе рассматриваются выплаты, предусмотренные законодательством Республики Беларусь о труде за неотработанное время. К

ним от носятся: оплата трудовых отпусков, льготных часов, выполнение государственных обязанностей и прочие выплаты, не связанные с основной деятельностью исполнителей.

Указанные выплаты распределяются на себестоимость отдельных объектов по нормативу (в процентах к основной заработной плате работников) и рассчитываются по формуле [24]:

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (5.2)$$

где $Н_д$ – норматив дополнительной заработной платы рабочих, равный 20%;

$З_о$ – затраты на основную заработную плату, руб.

Рассчитаем дополнительную заработную плату исполнителей

$$З_д = \frac{38394{,}6 \cdot 20}{100} = 7678{,}9 \text{ руб.}$$

В результате проведенных вычислений затраты на дополнительную заработную плату исполнителей составили 7678,9 рублей.

### 5.4 Расчет отчислений на социальные нужды

В данном разделе рассматриваются отчисления в фонд социальной защиты населения и на обязательное страхование, предусмотренные законодательством Республики Беларусь.

Расчёт осуществляется в соответствии с действующими законодательными актами по формуле [24]:

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \quad (5.3)$$

где $Н_{соц}$ – ставка отчислений в фонд социальной защиты населения (34%) и на обязательное страхование (0,6%), %;

$З_о$ – затраты на основную заработную плату, руб.;

$З_д$ – затраты на дополнительную заработную плату, руб.

Ставка отчислений в фонд социальной защиты населения, согласно действующему законодательству, составляет 34%. Так как предприятие, разрабатывающее данное программное средство, является резидентом ПВТ Республики Беларусь, то отчислениями в фонд социальной защиты населения не облагается часть доходов работника, превышающая однократный размер средней заработной платы работников в стране за месяц [25]. Номинальная начисленная средняя заработная плата за месяц в Республике Беларусь на январь 2020 года составляет 1238,7 рублей. А так как заработная плата каждого из работников данного проекта превышает среднюю по стране, то расчет отчислений на социальные нужды производится по формуле:

$$Р_{соц} = \frac{З_{ср} \cdot n \cdot Н_{соц}}{100}, \quad (5.4)$$

где $З_{ср}$ – средняя заработная плата в Республике Беларусь, руб.;

$n$ – количество месяцев на разработку проекта.

Рассчитаем отчисления в фонд социальной защиты населения

$$Р_{соц} = \frac{1238{,}7 \cdot 4 \cdot 34{,}6}{100} = 1714{,}4 \text{ руб.}$$

В результате проведенных вычислений затраты на отчисления в фонд социальной защиты населения составили 1714,4 рублей.

### 5.5 Расчет прочих затрат

Прочие затраты включаются в себестоимость разработки ПО в процентах от затрат на основную заработную плату команды разработчиков по формуле [24]:

$$З_{пз} = \frac{З_{о} \cdot Н_{пз}}{100}, \quad (5.5)$$

где $Н_{пз}$ – норматив прочих затрат (100%);

$З_{о}$ – затраты на основную заработную плату, руб.

Проведем расчет прочих затрат на разработку программного средства

$$З_{пз} = \frac{38394{,}6 \cdot 100}{100} = 38394{,}6 \text{ руб.}$$

В результате проведенных вычислений прочие затраты на разработку программного средства составили 38394,6 рублей.

### 5.6 Расчет полной суммы затрат на разработку ПО

Полная сумма затрат на разработку программного обеспечения находится путем суммирования всех рассчитанных статей затрат и вычисляется по формуле [24]:

$$С_{р} = З_{о} + З_{д} + Р_{соц} + З_{пз} \quad (5.6)$$

Полная сумма затрат на разработку проекта составит

$$С_{р} = 38394{,}6 + 7678{,}9 + 1714{,}4 + 38394{,}6 = 86182{,}5 \text{ руб.}$$

Таблица 5.2 – Затраты на разработку программного обеспечения

| Статья затрат | Сумма, руб. |
|---|---|
| Основная заработная плата команды | 38394,6 |
| Дополнительная заработная плата команды | 7678,9 |
| Отчисления на социальные нужды | 1714,4 |
| Прочие затраты | 38394,6 |
| Общая сумма затрат на разработку | 86182,5 |

В результате проведенных расчетов общая сумма затрат на разработку составила 86182,5 рублей, большая часть из которых – суммы затрат на заработную плату команды разработчиков и прочих затрат.

### 5.7 Расчет экономического эффекта от реализации программного средства на рынке информационных технологий

Программное средство планируется разместить в магазине мобильных приложений, пользователи смогут приобрести и загрузить приложение.

Предполагаемые объемы продаж составляют около 25 тысяч проданных копий программного средства. Такие объемы обосновываются тем, что в Республике Беларусь по данным на 1 января 2019 года на диспансерном учете находилось более 335 тысяч человек, из которых более 2 тысяч составляли дети [26]. При чем было отмечено, что ежегодно происходит прирост количества диабетиков в размере 58%. Предполагаемые объемы продаж составляют примерно тринадцатую часть от общего количества целевой аудитории программного продукта. Однако ожидается, что предполагаемые объемы продаж достигнут запланированного уровня в течение года после публикации программного средства в магазине приложений.

Анализ рынка существующих программных средств, проведенный в пункте 1.2, показал, что реализованные программные средства, хотя и являются бесплатными, обладают весьма схожими функциями, не обеспечивают должного уровня вовлечения пользователя в процесс отслеживания собственного состояния здоровья и совершенно не предусмотрены для пользователей детского и подросткового возраста. Элементарный анализ цен на ПО различной тематики в магазинах приложений показывает, что цены варьируются от 1 до 30 рублей. Поэтому цена за приобретение разработанного приложения устанавливается в размере 5 рублей, что является практически символическим вознаграждением организации-разработчику программного продукта и не вызывает у потенциального пользователя каких-либо мыслей о чрезмерной дороговизне продукта.

Если учесть, что компания-разработчик, выполняющая разработку, является членом ПВТ, то прибыль, полученная от распространения приложения, не облагается налогами на добавленную стоимость и на прибыль.

Тогда чистая прибыль от реализации программного средства на рынке будет рассчитываться по формуле [24]:

$$П = Ц \cdot N - С_р , \quad (5.7)$$

где Ц – доход от продажи одной копии программного средства, руб.;

$N$ – количество проданных копий программного средства;

$С_р$ – затраты на разработку программного средства, руб.

Тогда оценочная чистая прибыль составит

$$П = 5 \cdot 25000 - 86182{,}5 = 38817{,}5 \text{ руб.}$$

Таким образом, сумма инвестиций в разработку оказалась меньше, чем сумма годового эффекта от реализации программного средства. Это свидетельствует о том, что инвестиции окупятся менее, чем за год.

Для оценки эффективности затрат на разработку ПО в данном случае необходимо рассчитать простую норму прибыли (уровень рентабельности инвестиций) по формуле [24]:

$$У_р = \frac{П}{С_р} \cdot 100\% , \quad (5.8)$$

Тогда общая рентабельность разрабатываемого в данном дипломном проекте программного средства составит

$$У_р = \frac{38817{,}5}{86152{,}7} \cdot 100\% = 45\% .$$

В результате проведенных вычислений в ходе технико-экономического обоснования разработки программного средства для отслеживания данных о состоянии здоровья больных сахарным диабетом людей, были получены оценочная чистая прибыль в размере 38817,5 рублей и рентабельность инвестиций на уровне 45%. Данные экономические показатели свидетельствуют об экономической целесообразности и эффективности разработки программного средства.

# ЗАКЛЮЧЕНИЕ

В ходе данного дипломного проектирования было спроектировано и разработано программное средство, позволяющее пользователю, имеющему заболевание сахарным диабетом, вести учет состояния своего здоровья и пользоваться рядом других функций. При разработке были учтены основные требования к его функциональности:

- возможность авторизации в системе;
- возможность просмотра профиля пользователя;
- возможность внесения данных о потребленных пользователем калориях, уровне глюкозы в крови, времени физической активности;
- возможность устанавливать получение уведомлений с напоминанием о принятии лекарственных средств;
- возможность получения пользователем статистики своих данных;
- реализация игровой логики и системы вознаграждения пользователя;
- реализация чат-бота.

Перед началом разработки был проведен анализ аналогичных приложений и выполнено технико-экономическое обоснование проекта с целью определения экономической целесообразности разработки, внедрения и использования данного программного средства, относительно его аналогов. Были рассмотрены существующие на рынке программные средства, имеющие похожий функционал. Анализ показал, что все существующие приложения были бы сложны в использовании аудиторией подросткового и детского возраста. Именно это и стало отличительной чертой в разработке данного программного средства.

Для разработки программного средства использовались две интегрированные среды разработки: *Android Studio* и *Web Storm*. Программное средство предназначено для смартфонов, работающих под операционной системой *Android* от версии 5.0 и выше.

В процессе выполнения дипломной работы использовались научно-техническая литература и материалы с сайтов, посвящённых программированию мобильных устройств с операционной системой *Android*.

Приложение было протестировано и показало свою работоспособность как на стандартных эмуляторах, взятых из *AVD Manager*, так и на реальных устройствах на платформе *Android.*

В результате разработки цель была достигнута – было создано приложение, позволяющее вести учет здоровья, понятное для более юной аудитории и детей. Процесс разработки и сама готовая система были подробно описаны с использованием различных схем и диаграмм, а также было создано удобное руководство пользователя, которое позволит любому разобраться в системе без особых затруднений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


[1] Диабет [Электронный ресурс]. – Режим доступа: *https://www.who.int/ru/newsroom/factsheets/detail/diabetes*

[2] Мобильное приложение [Электронный ресурс]. – Режим доступа: *https://indicator.ru/label/mobilnoeprilozhenie*

[3] Мобильное приложение [Электронный ресурс]. – Режим доступа: *https://www.mk.ru/social/health/2016/06/01/bolshinstvodeteyssakharnymdiabetomchuvstvuyutsebyaneschastnymi.html*

[4] *Mobile OS market share worldwide* [Электронный ресурс]. – Режим доступа: *https://gs.statcounter.com/osmarketshare/mobile/worldwide*

[5] *React Native* [Электронный ресурс]. – Режим доступа: *https://reactnative.dev*

[6] *React Native*: год спустя [Электронный ресурс]. – Режим доступа: *https://engineering.fb.com/android/reactnativeayearinreview/*

[7] Архитектура *React Native* приложений [Электронный ресурс]. – Режим доступа: *https://2018.holyjspiter.ru/talks/3wy7ejpjymg6cqu0igqywa/*

[8] *Flutter* для мобильных разработчиков [Электронный ресурс]. – Режим доступа: *https://flutter.dev/docs/getstarted/flutterfor/androiddevs*

[9] *React Native, Flutter* [Электронный ресурс]. – Режим доступа: *https://trends.google.ru/trends/explore?date=today%205y&q=flutter,reactnative*

[10] Мобильная разработка: критерии выбора [Электронный ресурс]. – Режим доступа: *https://habr.com/ru/company/simbirsoft/blog/460030/*

[11] *WebStorm* [Электронный ресурс]. – Режим доступа: *https://www.jetbrains.com/webstorm/promo*

[12] *Meet Android Studio* [Электронный ресурс]. – Режим доступа: *https://developer.android.com/studio/intro*

[13] Что такое *Activity* [Электронный ресурс]. – Режим доступа: *https://developer.android.com/guide/components/activities/?hl=ru*

[14] Жизненный цикл на *Android* [Электронный ресурс]. Режим доступа: *http://developer.alexanderklimov.ru/android/theory/lifecycle.php*

[15] *Material Design* для *Android* [Электронный ресурс]. Режим доступа: *https://developer.android.com/design/material?hl=ru*

[16] Система сборки *Gradle* [Электронный ресурс]. – Режим доступа: *https://ru.wikipedia.org/wiki/Gradle*

[17] *PostgreSQL*: документация [Электронный ресурс]. – Режим доступа: *https://postgrespro.ru/docs/postgresql/9.6/introwhatis*

[18] Макеты [Электронный ресурс]. – Режим доступа: *https://webhamster.ru/mytetrashare/index/mtb189/14906146443cuohwb6fa*

[19] Алгоритм [Электронный ресурс]. – Режим доступа: *https://ktonanovenkogo.ru/voprosyiotvety/algoritmchtoehtotakoevidyalgoritmov.html*

[20] Для кого *Amazon Store* [Электронный ресурс]. – Режим доступа: *https://www.iguides.ru/main/gadgets/google/how_to_install_amazon_appstore/*

[21] Начало работы с *Google Play* [Электронный ресурс]. – Режим доступа: *https://support.google.com/googleplay/answer/topic=3364260*

[22] Страница разработчика [Электронный ресурс]. – Режим доступа: *https://support.google.com/googleplay/androiddeveloper/answer/6226441*

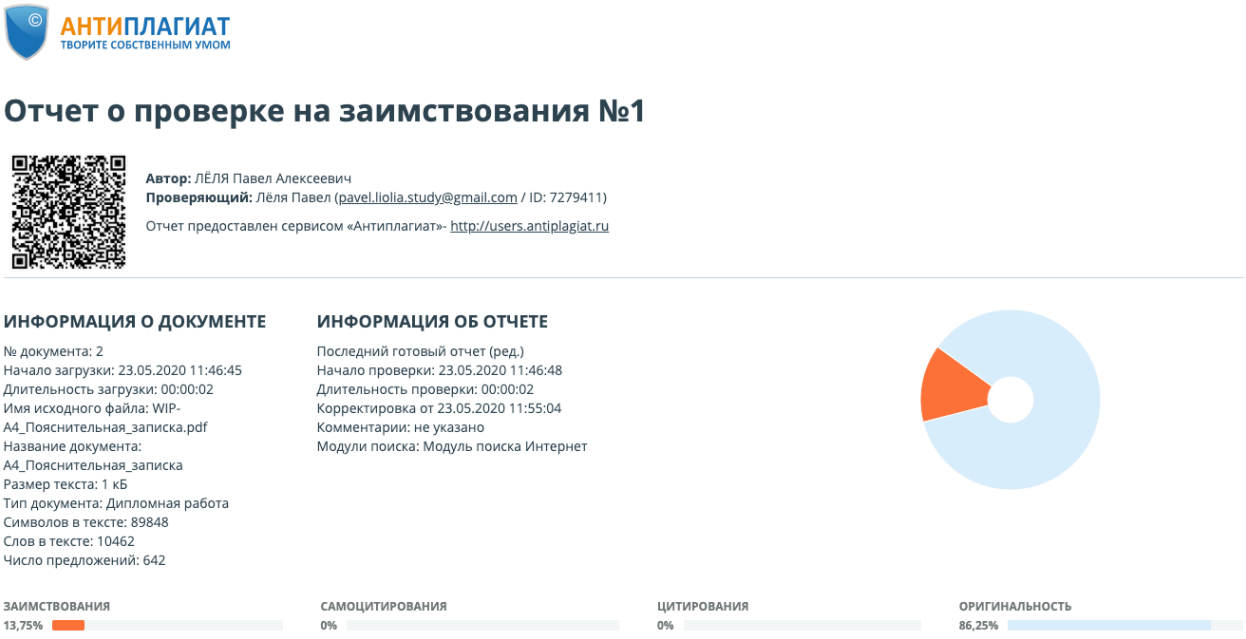
[23] Подписанный *APK* в *Android* [Электронный ресурс]. – Режим доступа: *http://javahelp.ru/androidstudiocreatesignedapk/*

[24] Экономическое обоснование проекта по разработке ПО [Электронный ресурс]. – Режим доступа: *https://www.bsuir.by/m/12_100229_1_124459.pdf*

[25] Декрет Президента РБ о ПВТ [Электронный ресурс]. – Режим доступа: *http://www.pravo.by/document/?guid=3871&p0=Pd0500012*

[26] Статистика больных сахарным диабетом в Беларуси [Электронный ресурс]. – Режим доступа: *https://interfax.by/news/zdorove/istoriya_bolezni/1267214/*

# ПРИЛОЖЕНИЕ А
# (обязательное)
# Отчёт по анализу заимствования материала



АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Отчет о проверке на заимствования №1

**Автор:** ЛЁЛЯ Павел Алексеевич
**Проверяющий:** Лёля Павел (pavel.liolia.study@gmail.com / ID: 7279411)
Отчет предоставлен сервисом «Антиплагиат»- http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 2
Начало загрузки: 23.05.2020 11:46:45
Длительность загрузки: 00:00:02
Имя исходного файла: WIP-A4_Пояснительная_записка.pdf
Название документа: A4_Пояснительная_записка
Размер текста: 1 кБ
Тип документа: Дипломная работа
Символов в тексте: 89848
Слов в тексте: 10462
Число предложений: 642

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 23.05.2020 11:46:48
Длительность проверки: 00:00:02
Корректировка от 23.05.2020 11:55:04
Комментарии: не указано
Модули поиска: Модуль поиска Интернет

ЗАИМСТВОВАНИЯ 13,75%
САМОЦИТИРОВАНИЯ 0%
ЦИТИРОВАНИЯ 0%
ОРИГИНАЛЬНОСТЬ 86,25%

Рисунок А.1 – Скриншот отчета о проверке дипломного проекта в системе «Антиплагиат»

# ПРИЛОЖЕНИЕ Б
## (обязательное)
## Листинги программного кода

```
const AppContainer = createAppContainer(
  createSwitchNavigator(
    {
      Main: MainNav,
      Login: LoginNav,
      Loading: SplashScreen,
      Onboarding: OnBoardingTutorial,
    },
    { initialRouteName: 'Loading' }
  )
);

export default class App extends React.PureComponent {
  state = {
    isBackgroundSoundPlayed: false,
    isSettingsModalOpen: false,

    isNotificationsAvailable: null,
  };

  changeBackgroundMusicPlayStatus = isBackgroundSoundPlayed => {
    Sound.setSound(isBackgroundSoundPlayed);

    this.setState({
      isBackgroundSoundPlayed,
    });
  };

  openSettingsModal = () => {
    MixpanelService.track('click:opensettings');
    this.setState({
      isSettingsModalOpen: true,
    });
  };

  closeSettingsModal = () => {
    MixpanelService.track('click:closesettings');
    this.setState({
      isSettingsModalOpen: false,
    });
  };

  changeNotificationsStatus = async isNotificationsAvailable => {
    if (isNotificationsAvailable !== this.state.isNotificationsAvailable) {
      this.setNotificationsStatus(isNotificationsAvailable);

      if (isNotificationsAvailable) {
        await NotificationService.requestPermissions();
        await NotificationService.registerNotifications();
      }

      MixpanelService.trackWithProperties('click:settingsnotifications', {
        isNotificationsAvailable,
      });

      const status = await
NotificationService.changeNotificationStatus(isNotificationsAvailable);
```

```
      if (isNotificationsAvailable !== status) {
        this.setNotificationsStatus(status);
      }
    }
  };

  setNotificationsStatus = isNotificationsAvailable => {
    this.setState({
      isNotificationsAvailable,
    });
  };

  render() {
    const {
      isBackgroundSoundPlayed,
      isSettingsModalOpen,
      isNotificationsAvailable,
    } = this.state;

    return (
      <KeyboardAvoidingView
        enabled={Platform.OS === 'ios'}
        behavior="padding"
        style={{ flex: 1 }}
      >
        <Root>
          <LocationProvider>
            <SettingsContext.Provider
              value={{
                isSettingsModalOpen,
                openSettingsModal: this.openSettingsModal,
                closeSettingsModal: this.closeSettingsModal,

                isNotificationsAvailable,
                changeNotificationsStatus: this.changeNotificationsStatus,
                setNotificationsStatus: this.setNotificationsStatus,

                isBackgroundSoundPlayed,
                changeBackgroundMusicPlayStatus:
this.changeBackgroundMusicPlayStatus,
              }}
            >
              <AppContainer
                ref={navigatorRef => {
                  NavigationTool.setTopLevelNavigator(navigatorRef);
                }}
              />
              <Video
                repeat
                paused={!isBackgroundSoundPlayed}
                source={BackgroundSound}
                style={{ position: 'absolute', top: 0 }}
              />

              <Modal
                animationType="fade"
                transparent
                visible={isSettingsModalOpen}
                onRequestClose={this.closeSettingsModal}
              >
                <SettingsModal onClose={this.closeSettingsModal} />
              </Modal>
              <LevelCongratulation />
            </SettingsContext.Provider>
```

```
          </LocationProvider>
        </Root>
      </KeyboardAvoidingView>
    );
  }
}

class DrawerWithLogout extends React.PureComponent {
  static contextType = SettingsContext;

  constructor(props) {
    super(props);
  }

  logout = async () => {
    this.context.changeBackgroundMusicPlayStatus(false);
    await LastLocation.setLevel(1);
    resetGuide();
    await AuthService.logout();
    this.props.navigation.navigate('Login');
  };

  goHome = () => {
    this.props.navigation.navigate('Home');
  };

  goToProfile = () => {
    this.props.navigation.navigate('Profile');
  };

  goToFeedbackScreen = () => {
    this.props.navigation.navigate('Feedback');
  }

  render() {
    return (
      <SettingsContext.Consumer>
        {settings => {
          return (
            <ScrollView>
              <SafeAreaView forceInset={{ top: 'always', horizontal: 'never'
}}>
                <View style={[styles.item, this.props.activeItemKey ===
'Home' ? styles.activeItem : {}]}>
                  <TouchableOpacity onPress={this.goHome}>
                    <Text style={[styles.itemLabel, this.props.activeItemKey
=== 'Home' ? styles.activeItemLabel : {}]}>

{LocaleTranslator.translate(LocaleTranslator.localeConstants().settings.HOME)
}
                    </Text>
                  </TouchableOpacity>
                </View>
                <View style={[styles.item, this.props.activeItemKey ===
'Profile' ? styles.activeItem : {}]}>
                  <TouchableOpacity onPress={this.goToProfile}>
                    <Text
                      style={[styles.itemLabel, this.props.activeItemKey ===
'Profile' ? styles.activeItemLabel : {}]}
                    >

{LocaleTranslator.translate(LocaleTranslator.localeConstants().settings.PROFI
LE)}
                    </Text>
```

```
                  </TouchableOpacity>
                </View>
                <View style={styles.item}>
                  <TouchableOpacity
                    onPress={() => {

this.props.navigation.dispatch(DrawerActions.closeDrawer());
                      settings.openSettingsModal();
                    }}
                  >
                    <Text style={styles.itemLabel}>

{LocaleTranslator.translate(LocaleTranslator.localeConstants().settings.SETTI
NGS)}
                    </Text>
                  </TouchableOpacity>
                </View>
                <View style={[styles.item, this.props.activeItemKey ===
'Feedback' ? styles.activeItem : {}]}>
                  <TouchableOpacity onPress={this.goToFeedbackScreen}>
                    <Text
                      style={[styles.itemLabel, this.props.activeItemKey ===
'Feedback' ? styles.activeItemLabel : {}]}
                    >

{LocaleTranslator.translate(LocaleTranslator.localeConstants().settings.FEEDB
ACK)}
                    </Text>
                  </TouchableOpacity>
                </View>
                <TouchableOpacity style={styles.lastItem}
onPress={this.logout}>
                  <Text style={styles.itemLabel}>

{LocaleTranslator.translate(LocaleTranslator.localeConstants().settings.LOGOU
T)}
                  </Text>
                </TouchableOpacity>
              </SafeAreaView>
            </ScrollView>
          );
        }}
      </SettingsContext.Consumer>
    );
  }
}

const styles = StyleSheet.create({
  activeItem: {
    backgroundColor: '#f5f5f5',
  },
  activeItemLabel: {
    color: '#2399ef',
  },
  item: {
    marginTop: 0,
    marginBottom: 0,
    paddingTop: 8,
    paddingBottom: 8,
    paddingLeft: 15,
    borderBottomWidth: StyleSheet.hairlineWidth,
    borderBottomColor: 'grey',
  },
  lastItem: {
```

```
    marginTop: 0,
    marginBottom: 0,
    paddingTop: 8,
    paddingBottom: 8,
    marginLeft: 15,
  },
  itemLabel: {
    marginLeft: 15,
    marginTop: 0,
    marginBottom: 0,
    paddingTop: 8,
    paddingBottom: 8,
    fontWeight: 'bold',
    fontSize: 18,
  },
});

export default createDrawerNavigator(
  {
    Home: { screen: HomeScreen },
    Profile: { screen: ProfileScreen },
    Chat: { screen: ChatScreen },
    PostCards: { screen: PostCardScreen },
    WeeklyChart: { screen: WeeklyChart },
    DailyChart: { screen: DailyChart },
    Feedback: { screen: FeedbackScreen },
  },
  {
    initialRouteName: 'Home',
    contentOptions: {
      itemStyle: styles.item,
      labelStyle: styles.itemLabel,
    },
    contentComponent: props => <DrawerWithLogout {...props} />,
  }
);
```

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