**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«БАЛТИЙСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. И. КАНТА»**

**ИНСТИТУТ ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК**

**И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

|  |  |
| --- | --- |
| Рекомендована к защите:  методический руководитель  направления подготовки  к. ф.-м. н., доцент ИФМНиИТ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мищук Б.Р.  "\_\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | Допущена к защите:  первый заместитель директора  ИФМНиИТ  к. ф.-м. н., доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Шпилевой  "\_\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**Тема: «РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ МЕДИЦИНСКОГО УСТРОЙСТВА»**

**Направление подготовки: 01.03.02 «Прикладная математика и информатика»**

Квалификация (степень**)**: **бакалавр**

ВКР защищена на оценку: **Выполнил:** студент 4 курса

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_С.А.Зорко

**Руководитель:** старший преподаватель

ИФМНиИТ

\_\_\_\_\_\_\_\_\_ А.О. Синюхин

Калининград, 2020

Оглавление

[Введение 3](#_Toc40544658)

[Глава 1 Изучение функциональных возможностей устройства и протокола взаимодействия с ним 7](#_Toc40544659)

[1.1 Обзор протоколов Bluetooth 7](#_Toc40544660)

[1.2 Протокол Bluetooth LE 9](#_Toc40544661)

[1.3 Описание используемого миостимулятора 14](#_Toc40544662)

[1.4 Взаимодействие с миостимулятором 15](#_Toc40544663)

[Глава 2 Анализ имеющегося мобильного приложения 19](#_Toc40544664)

[2.1 Технологии используемые в имеющимся мобильном приложении 19](#_Toc40544665)

[2.2 Анализ Android приложения 20](#_Toc40544666)

[Глава 3 Разработка нового Android приложения 26](#_Toc40544667)

[3.1 Обзор платформ для разработки 26](#_Toc40544668)

[3.2 Подготовка к разработке мобильного приложения 31](#_Toc40544669)

[3.3 Доработка функционала взаимодействия с устройством 33](#_Toc40544670)

[3.4 Доработка интерфейса приложения 36](#_Toc40544671)

# Введение

Электронное здравоохранение (e-health англ.) - это использование информационных и коммуникационных технологий (ИКТ) для здравоохранения. За последние 10 лет рынок e-health устройств значительно вырос. Ученые по всему миру совершают тысячи открытий в сфере медицины и компьютерных технологий, которые позволяют реализовать все более функциональные устройства и методы лечения, диагностики пациентов.

Электронное здравоохранение включает в себя такие компоненты как:

* Мобильные медицинские системы (mHealth)
* Информационные системы здравоохранения (HIS)
* Телемедицина
* Обучение на расстоянии (eLearning)

Мобильные медицинские системы – это системы использующие мобильные устройства в медицине и общественном здравоохранении. Этот термин чаще всего используется в отношении использования устройств мобильной связи, таких как мобильные телефоны, планшетные компьютеры и персональные цифровые помощники (КПК), и носимые устройства, такие как умные часы, для упрощения или улучшения качества работы служб здравоохранения, получения информации о состоянии пациентов и сбора данных. Варианты использования мобильных медицинских систем включают использование мобильных устройств для сбора данных о состоянии здоровья населения или пациентов отдельно взятого медицинского учреждения, сбор медицинской информации практикующим врачам, исследователям и пациентам, мониторинг показателей жизнедеятельности пациента в режиме реального времени, прямое предоставление медицинской помощи (с помощью мобильной телемедицины), а также обучение и взаимодействие работников здравоохранения.

Среди большого разнообразия mHelath устройств существует такой класс устройств как – миостимуляторы. Принцип работы миостимулятора основан на воздействии на мышцы тела с помощью электрических импульсов. Электростимуляция активно воздействует на организм за счет импульсов разной длительности - от 0,5 до 300 мс, с силой тока до 5 мА (на лице), до 100 мА (на теле) и частотой 10-150 Гц, которые работают с прерываниями. Во время процедуры этот эффект идентичен нормальной работе мышц во время их деятельности. Ток, проходящий через ткани, возбуждает клетки и стимулирует активную работу мышц, а в перерывах расслабляет их. Благодаря такому ритму работы ток под электродами не раздражает кожу, а эпидермис не повреждается из-за длительной электрической стимуляции. Когда электрический ток воздействует на мышцы и нервы, их биологическая активность изменяется. Импульсы провоцируют сокращение мышц, что укрепляет и активизирует их. Если мышца перегружена, электростимуляция помогает снять такой стресс. Электростимуляция мышц спины помогает пациентам с неподвижным шейным, поясничным, воспаленным или искривленным позвоночником, которым присущи боли, они потеряли чувствительность, мышцы ослаблены и многое другое. Также такая процедура необходима на этапе реабилитации после операции на позвоночнике, она поможет укрепить мышечный каркас на позвоночнике.

Большинству mHealth устройств необходимо наличие интерфейса для удаленной настройки и сбора статистики. Это позволяет сократить затраты на производство устройства. Обычно такие интерфейсы основаны на протоколах:

* USB
* Wi-Fi
* Мобильная связь (2/3/4/5G)
* Bluetooth Low Energy

Основной недостаток USB интерфейсов – необходимость проводного подключения к устройству, но в то же время достоинством является – бесперебойная связь с устройством и быстрая скорость передачи данных между смартфоном и устройством. Wi-Fi позволяет организовать передачу данных между устройствами находящимися в одной Wi-Fi сети. Это удобно для организации связи различных IoT устройств, находящихся в относительной близости друг от друга, но связь теряется при выходе за пределы действия Wi-Fi сети. Мобильная связь позволяет передать данные на любые расстояния при подключении к ней обоих устройств. Главным недостатком мобильной связи считается обязательное наличие SIM карты и не возможность бесплатного подключения к мобильной связи. В настоящее время стандартом протокола для работы с mHealth устройствами является Bluetooth Low Energy подключение. В сравнении с классическим Bluetooth, Bluetooth Low Energy потребляет гораздо меньше энергии. Это позволяет мобильным устройствам обмениваться данными с устройствами, имеющими строгие энергетические потребностями. Недостатками является необходимость передавать данные между телефоном и устройством маленькими порциями, следовательно низкая скорость передачи данных.

**Целью** данной выпускной квалификационной работы (ВКР) является

реализация мобильного приложения для управления миостимулятором с использованием Bluetooth Low Energy протокола, которое будет использовать все возможности устройства, работать без сбоев, а также иметь удобный пользовательский интерфейс.

**Актуальность работы** обусловлена необходимостью в мобильном приложении для управления работой миостимулятора которое будет задавать программу работы, а также собирать различные метрики с устройства.

**Задачи ВКР:**

1. Выбрать платформу для разработки
2. Разработать пользовательский интерфейс
3. Расширить функционал модуля взаимодействия с устройством
4. Наладить бесперебойную работу Bluetooth соединения

**Структура работы.** Выпускная квалификационная работа по теме «Разработка автоматизированной информационной системы учета движения товаров ООО «Европласт» состоит из введения, трех глав, заключения, списка использованной литературы, перечня сокращений и приложений.

Во введении раскрывается актуальность темы, а также цель и задачи.

В первой главе рассматриваются общие вопросы касающиеся взаимодействия с устройством. {…} В заключении главы формулируются выводы.

Во второй главе …

В третьей главе …

В заключении подводятся итоги работы, формулируются окончательные выводы.

В приложении …

**Содержание работы.** Полный объем работы составляет X страниц машинописного текста. Всего работа содержит X рисунков и Y таблиц.

# Глава 1 Изучение функциональных возможностей устройства и протокола взаимодействия с ним

Взаимодействие мобильного приложения с миостимулятором осуществляется по протоколу Bluetooth Low Energy.

## Обзор протоколов Bluetooth

Bluetooth - это стандарт беспроводной технологии, используемый для обмена данными между стационарными и мобильными устройствами на коротких расстояниях с использованием коротковолновых УВЧ-радиоволн в промышленных, научных и медицинских радиодиапазонах, от 2,402 ГГц до 2,480 ГГц, и для создания персональных сетей (PAN). Первоначально он был задуман как беспроводная альтернатива кабелям передачи данных RS-232.

В настоящее время существует пять основных версий стандарта Bluetooth:

* **Bluetooth 1.x -** имел много проблем, и производители испытывали трудности с обеспечением совместимости своих продуктов. Версии 1.0 и 1.0B также включали обязательную передачу адреса аппаратного устройства Bluetooth в процессе подключения (что делало анонимность невозможной на уровне протокола).
* **Bluetooth 2.x** - Эта версия спецификации ядра Bluetooth была выпущена до 2005 года. Основным отличием является увеличение скорости передачи до 3 Мбит / с, хотя максимальная скорость передачи данных (с учетом межпакетного времени и подтверждений) составляет 2,1 Мбит/с.
* **Bluetooth 3.x -** был выпущен в апреле 2009 года. Обеспечивает теоретическую скорость передачи данных до 24 Мбит/с, но не через сам канал Bluetooth. Вместо этого канал Bluetooth используется для согласования и установления соединения, а трафик с высокой скоростью передачи данных передается по каналу связи 802.11.
* **Bluetooth 4.x -** был выпущен в июне 2010 года. Он включает в себя классические протоколы Bluetooth, высокоскоростной Bluetooth и Bluetooth с низким энергопотреблением (Bluetooth LE).
* **Bluetooth 5.x –** был выпущен в декабре 2016 года. Его новые функции в основном ориентированы на новую технологию Интернета вещей. Одним из основных отличий было добавление возможностей для увеличения скорости передачи данных по Bluetooth LE протоколу до 2 Мбит/с за счет уменьшения дальности беспроводной передачи данных или увеличение дальности за счет уменьшения скорости передачи данных.

**1.1.1 Влияние на здоровье**

Bluetooth использует микроволновый радиочастотный спектр в диапазоне 2,402–2480 ГГц, который представляет собой неионизирующее излучение, с полосой частот, аналогичной ширине полосы частот, используемой беспроводными и мобильными телефонами. До настоящего времени не было продемонстрировано никакой конкретной демонстрации вреда, даже если IARC (Международное агенство по изучению рака) включил беспроводную передачу в список возможных канцерогенов. Максимальная выходная мощность радиомодуля Bluetooth составляет 100 мВт для класса 1, 2,5 мВт для класса 2 и 1 мВт для устройств класса 3. Даже максимальная выходная мощность класса 1 ниже, чем у мобильных телефонов с самым низким энергопотреблением. Это доказывает, что Bluetooth протокол абсолютно безопасен для здоровья конечного пользователя.

## Протокол Bluetooth LE

Bluetooth Low Energy, ранее известный как Wibree, является подмножеством Bluetooth v4.0 с совершенно новым стеком протоколов для быстрого создания неспецифических соединений. В качестве альтернативы стандартным протоколам Bluetooth, которые были представлены в Bluetooth v1.0 v3.0, он предназначен для приложений с очень низким энергопотреблением, работающих от небольшого источника питания. Конструкция микросхемы допускает два типа реализации: двухрежимную и однорежимную.

По сравнению с классическим Bluetooth, Bluetooth Low Energy предназначен для обеспечения значительно сниженного энергопотребления и стоимости при сохранении аналогичного диапазона связи. С точки зрения увеличения времени автономной работы устройств Bluetooth, BLE представляет собой значительный прогресс.

В однорежимной реализации реализован только стек протоколов с низким энергопотреблением.

В двухрежимной реализации функциональность Bluetooth LE интегрирована в существующий классический контроллер Bluetooth.

**1.2.1 Радиоинтерфейс протокола Bluetooth LE**

Технология Bluetooth LE работает в том же диапазоне спектра (диапазон ISM 2,400–2,4835 ГГц), что и классическая технология Bluetooth, но использует другой набор каналов. Вместо классических семидесяти девяти каналов по 1 МГц, Bluetooth Low Energy имеет сорок каналов по 2 МГц. Внутри канала данные передаются с использованием гауссовой частотной модуляции, аналогично классической схеме базовой скорости Bluetooth. Скорость передачи составляет 1 Мбит/с (с возможностью 2 Мбит/с в Bluetooth 5), а максимальная мощность передачи составляет 10 мВт (100 мВт в Bluetooth 5).

Bluetooth LE использует скачкообразную перестройку частоты, чтобы противодействовать узкополосным помехам. Классический Bluetooth также использует скачкообразную перестройку частоты, но детали отличаются. В результате, в то время как FCC и ETSI классифицируют технологию Bluetooth как схему FHSS, Bluetooth Low Energy классифицируется как система, использующая методы цифровой модуляции или расширенный спектр прямой последовательности.

**1.2.2 Оповещение и обнаружение устройств**

Устройства BLE обнаруживаются с помощью процедуры, основанной на трансляции оповещающих пакетов. Это делается с использованием 3 отдельных каналов (частот), чтобы уменьшить помехи. Оповещающее устройство отправляет пакет по меньшей мере на один из этих трех каналов с периодом повторения, называемым интервалом оповещения. Для уменьшения вероятности множественных последовательных коллизий к каждому интервалу оповещения добавляется случайная задержка до 10 миллисекунд. Сканер прослушивает канал в течение периода, называемого окном сканирования, которое периодически повторяется каждый интервал сканирования.

Следовательно, задержка обнаружения определяется вероятностным процессом и зависит от трех параметров (а именно, интервала оповещения, интервала сканирования и окна сканирования). Схема обнаружения BLE использует метод периодического интервала, для которого верхние границы задержки обнаружения могут быть выведены для большинства параметризаций. Хотя задержки обнаружения BLE могут быть аппроксимированы моделями, для чисто периодических протоколов, основанных на интервалах, случайная задержка, добавляемая к каждому интервалу оповещения и трехканальному обнаружению, может вызвать отклонения от этих прогнозов или потенциально привести к неограниченным задержкам для некоторые параметризации.

**1.2.3 Безопасность**

Bluetooth LE поставляется с очень низким стандартом безопасности. BTLE или Bluetooth Smart — это новый режим модуляции и форматирования пакетов канального уровня, предназначенный для устройств с низким энергопотреблением. Он встречается в последних моделях смартфонов, спортивных устройствах, сенсорах и скоро появится во многих медицинских устройствах. К сожалению, реализация шифрования данных для протокола Bluetooth LE усложняется тем, что она сильно увеличивает энергопотребление, нивелируя энергосберегающие преимущества данного протокола. Этот недостаток безопасности дает возможность любому устройству поблизости прослушивать Bluetooth LE подключение. Данная уязвимость позволяет перехватывать пакеты для дальнейшей сборки для получения данных, передаваемых между устройствами, а также атаки, подразумевающие “инъекцию” данных.

Bluetooth LE был предложен в качестве основы для приложения по отслеживанию контактов людей во время пандемии COVID-19 2020 года, для бесшумного отслеживания других телефонов, которые находятся в диапазоне Bluetooth LE, позволяя тем, кто находился в постоянной близости с возможно, зараженным человеком быть уведомленным без какой-либо идентификации личности.

**1.2.4 Взаимодействие с программным обеспечением**

Все устройства Bluetooth LE используют профиль общих атрибутов (GATT). Интерфейс прикладного программирования, предлагаемый операционной системой, работающей на устройстве с поддержкой Bluetooth Low Energy, обычно основывается на концепциях GATT. GATT имеет следующую терминологию:

* **Клиент** – устройство, которое отправляет GATT команды и запросы, а также принимает ответы. Например, мобильный телефон пользователя.
* **Сервер** – устройство, которое принимает GATT команды и запросы, а также возвращает ответы на полученные запросы. Например, датчик температуры.
* **Характеристика** - значение данных, передаваемое между клиентом и сервером. Например, текущее напряжение батареи.
* **Сервис** – набор связанных характеристик, которые работают вместе, чтобы выполнить определенную функцию. Например, сервис “Термометр” включает характеристики для значения измерения температуры и временной интервал между измерениями.
* **Дескриптор** - предоставляет дополнительную информацию о характеристике. Например, характеристика значения температуры может иметь указание ее единиц измерения (например, по Цельсию), а также максимальных и минимальных значений, которые может измерять датчик. Дескрипторы являются необязательными - каждая характеристика может иметь любое количество дескрипторов.

Некоторые значения сервиса и характеристики используются в административных целях - например, имя модели и серийный номер можно считать стандартными характеристиками в сервисе общего доступа. Сервисы могут также включать другие сервисы в качестве вспомогательных. Основные функции устройства – это так называемые первичные сервисы, а вспомогательные функции, которые к ним относятся, являются вторичными сервисами.

Сервисы, характеристики и дескрипторы совместно называются атрибутами и идентифицируются UUID. Любой разработчик может выбрать случайный или псевдослучайный UUID для частного использования, но Bluetooth SIG зарезервировал диапазон UUID (в форме xxxxxxxx-0000-1000-8000-00805F9B34FB) для стандартных атрибутов. Для эффективности эти идентификаторы представлены в протоколе в виде 16-битных или 32-битных значений, а не 128 битов, необходимых для полного UUID.

Протокол GATT предоставляет клиенту ряд команд для обнаружения информации о сервере. К ним относятся:

* Обнаружение UUID для всех основных сервисов
* Найти сервис с заданным UUID
* Найти вторичные сервисы для заданного первичного сервиса
* Просмотреть все характеристики для заданного сервиса
* Найти характеристики, соответствующие заданному UUID
* Считать все дескрипторы для определенной характеристики

Также предоставляются команды для чтения (передача данных с сервера на клиент) и записи (от клиента к серверу) значений характеристик:

* Значение может быть прочитано либо путем указания UUID характеристики, либо по значению идентификатора (которое возвращается вышеприведенными командами).
* Операции записи всегда идентифицируют характеристику по идентификатору, но могут выбирать, требуется ли ответ от сервера.
* Операции «длинное чтение» и «длинная запись» могут использоваться, когда длина данных характеристики превышает максимально разрешенный размер пакета для данной радиолинии.

GATT поддерживает уведомления и указания. Клиент может запросить уведомление о конкретной характеристике с сервера. Сервер может затем отправить значение клиенту, когда оно станет доступным. Например, сервер датчика температуры может уведомлять своего клиента каждый раз, когда он проводит измерение. Это избавляет клиента от необходимости опрашивать сервер, что требует постоянной работы радиосхемы сервера. Указание аналогично уведомлению, за исключением того, что оно требует ответа от клиента в качестве подтверждения того, что он получил сообщение.

**1.2.5 Воздействие на батарею**

Bluetooth Low Energy разработан для того, чтобы устройства имели низкое энергопотребление. Устройства с периферийными и центральными функциями имеют разные требования к питанию. Исследование, проведенное компанией Aislelabs , занимающейся разработкой программного обеспечения для маяков, показало, что такие периферийные устройства, как бесконтактные маяки, обычно работают в течение 1-2 лет, питаясь от батарейки типа «таблетка» емкостью 1000 мАч. Это возможно из-за энергоэффективности протокола Bluetooth Low Energy, который передает только небольшие пакеты по сравнению с Bluetooth Classic, который также подходит для аудио и данных с высокой пропускной способностью.

Напротив, непрерывное сканирование для тех же маяков в центральной роли может потреблять 1000 мАч в течение нескольких часов. Устройства Android и iOS также сильно влияют на заряд батареи в зависимости от типа сканирования и количества находящихся рядом устройств с низким энергопотреблением Bluetooth. Благодаря новым чипсетам и достижениям в программном обеспечении, к 2014 году телефоны на базе Android и iOS имели незначительное энергопотребление при использовании Bluetooth LE.

## Описание используемого миостимулятора

Миостимулятор представляет собой носимое устройство, от которого отходят провода к спине пациента. Передача данных осуществляется при помощи встроенного Bluetooth LE модуля. Устройство имеет разъем для SD/MMC-карты c Flash памятью.

* Максимальный ток стимуляции мышечной ткани для данного устройства - до 5мА на канал
* Время непрерывной работы в режиме стимуляции до 5 часов;
* Время работы в режиме ожидания 6 суток;
* Максимальное кол-во различных запрограммированных сеансов — до 40;
* Максимальное кол-во событий включения по расписанию — до 10 в сутки;
* Интерфейсы: Buetooth 4.0, USB
* Частота стимуляции: 30-4000Hz
* Формы импульсов стимуляции: меандр, синусоидальный, трапечевидный, пилообразный



## Взаимодействие с миостимулятором

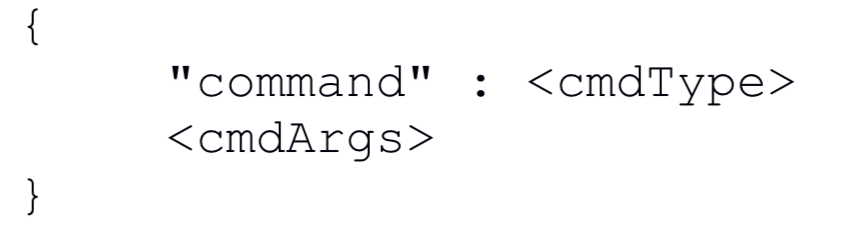
Прикладной уровень стека протоколов взаимодействия устройства с мобильным приложением основан на формате данных JSON. Обмен данными осуществляется по принципу "запрос-ответ", минимальной единицей данных является JSON-документ. Мобильное приложение посылает *документ*, описывающий операцию (команду), которую должно выполнить устройство; получив и обработав её, устройство посылает ответное сообщение, содержащее код выполнения и (для некоторых команд) полезные данные.

Устройство поддерживает следующие операции:

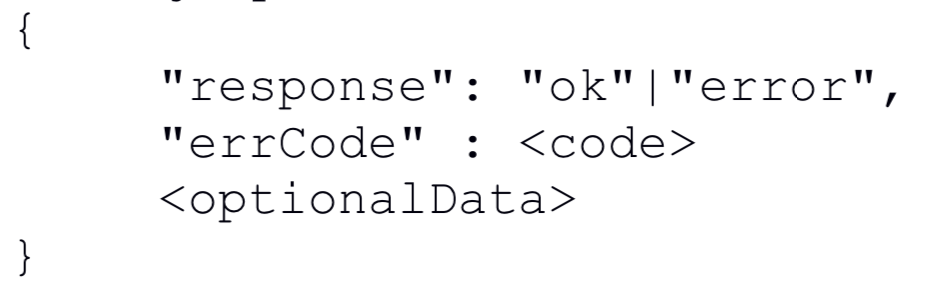
* Синхронизация (установка) времени на устройстве;
* Программирование плана лечения (по расписанию или в ручном режиме);
* Запуск и остановка сеанса лечения вручную;
* Запрос данных о батарее питания (напряжение);

Если операция по той или иной причине не выполнена, возвращается код ошибки, указывающий на причину её возникновения.

Общий вид запроса, посылаемого мобильным приложением:



Ответ устройства:

******

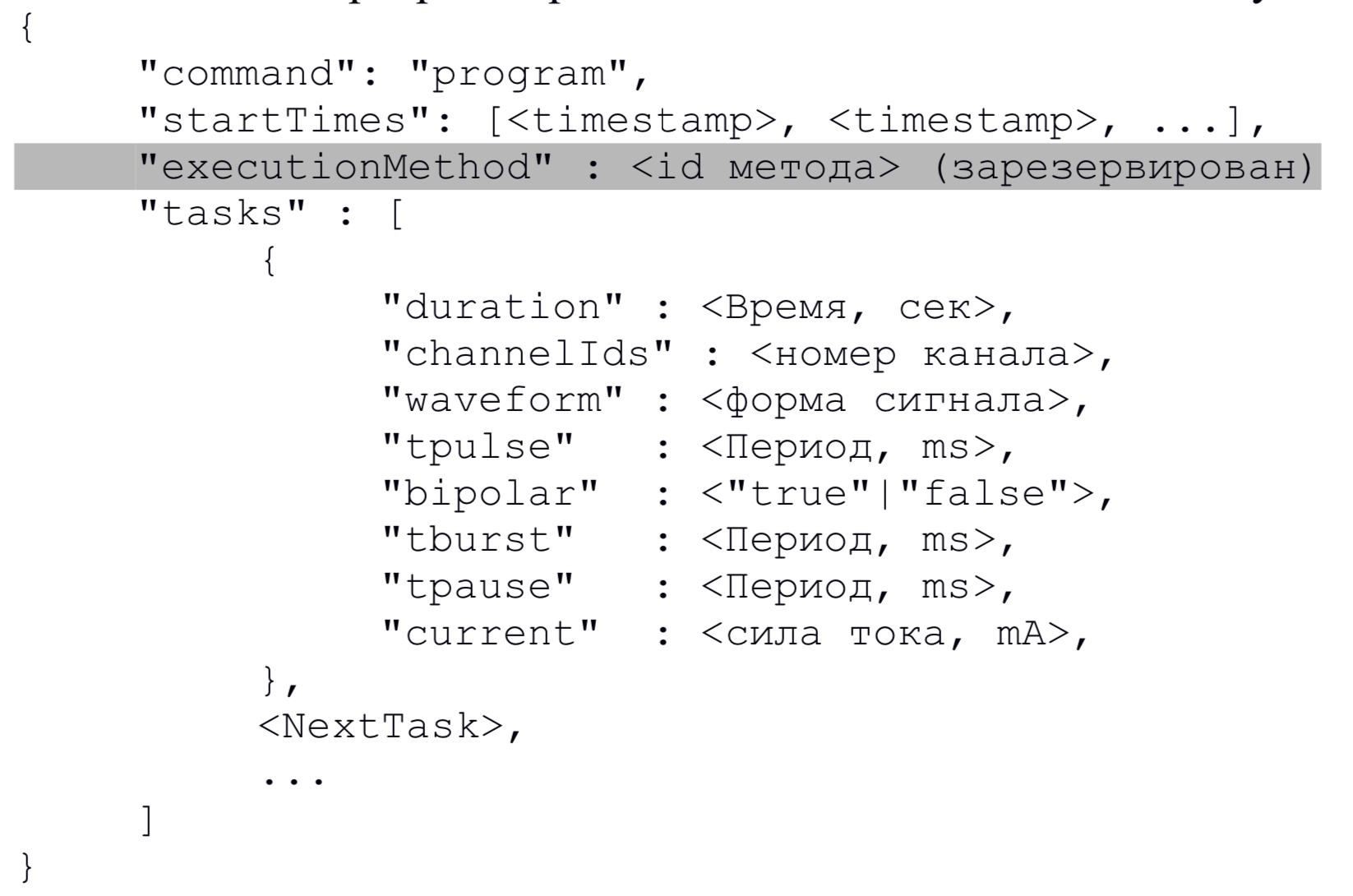
**1.4.1 Программирование плана лечения**

План лечения (называемый также программой лечения) представляет собой список сеансов, в каждом из которых описываются все необходимые параметры:

* Время начала электромиостимуляции
* Сила тока
* Форма импульсов
* Частота импульса
* Длительность серии импульсов и длительность паузы между ними
* Двуполярный или однополярный сигнал.

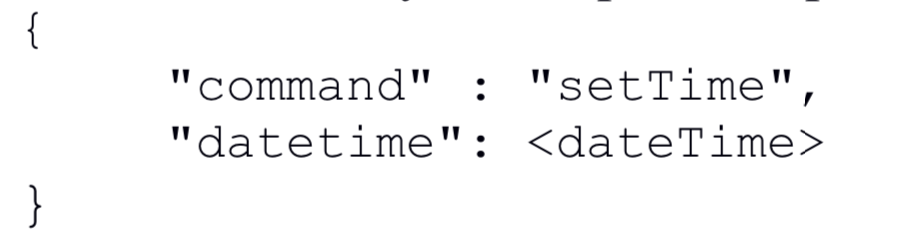
Максимальное количество сеансов в плане лечения равно 40, при этом план лечения может иметь до 10 меток времени, т.е. быть запущен до 10 раз в сутки.

Команда программирования плана лечения имеет следующий формат:

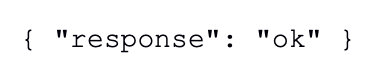
******

**1.4.2 Синхонизация времени на устройстве**

Синхронизация времени необходима для того, чтобы устройство включалось в нужное время по расписанию. Команда имеет формат:



В случае успеха возвращается:



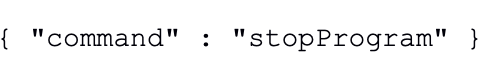
**1.4.3 Ручной запуск/остановка сеанса лечения**

Любая программа лечения может быть запущена принудительно командой с мобильного устройтва, либо кнопкой "Старт/Стоп" на корпусе устройства.

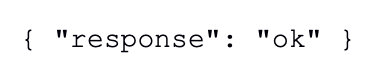
Если в данный момент времени устройтво исполняет план лечения по расписанию, ручной запуск прервёт текущий сеанс. По выходу из ручного режима, устройство продолжит работать в режиме по расписанию.

Формат команды запуска:

  
Формат команды останова:



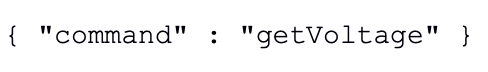
При нормальных условиях функциорирования устройства, всегда возвращается:



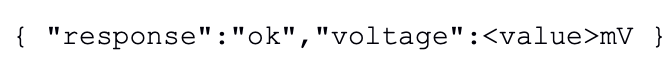
Возможна задержка до 1*сек* между моментом окончания обработки команды и моментом начала/окончания электромиостимуляции.

**1.4.4 Запрос напряжения батареи питания**

Команда может быть использована для мониторинга заряда батареи. Формат команды:



Формат ответа:



# Глава 2 Анализ имеющегося мобильного приложения

## 2.1 Технологии используемые в имеющимся мобильном приложении

В первую очередь был проведен анализ мобильного приложения. Целью было ознакомиться с его архитектурой, типами объектов и используемыми сторонними библиотеками.

При старте приложение инициализирует библиотеки Crashlitics - отвечающую за сбор информации об ошибках, Timber – отвечающую за форматирование логов, Toothpick – отвечающую за “инъекцию зависимостей” в приложение, Threeten – отвечающую за работу со временем. Разберем основные технологии, применяемые в данном приложении.

**2.1.1 Внедрение зависимости**

Внедрение зависимости (англ. Dependency injection, DI) — процесс предоставления внешней зависимости программному компоненту. Является специфичной формой «инверсии управления» (англ. Inversion of control, IoC), когда она применяется к управлению зависимостями. В полном соответствии с принципом единственной обязанности объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму.

Реализация внедрения зависимостей дает следующие преимущества:

* Повторное использование кода
* Легкость рефакторинга
* Простота тестирования

В данном приложении за реализацию данного паттерна отвечает библиотека Toothpick, основанная на дереве областей данных. По мере выхода из области данных, дерево сокращается, тем самым освобождая память.

**2.1.2 Методы реактивного программирования**

Реактивное программирование — парадигма программирования, ориентированная на потоки данных и распространение изменений. Это означает, что должна существовать возможность легко выражать статические и динамические потоки данных, а также то, что нижележащая модель исполнения должна автоматически распространять изменения благодаря потоку данных.

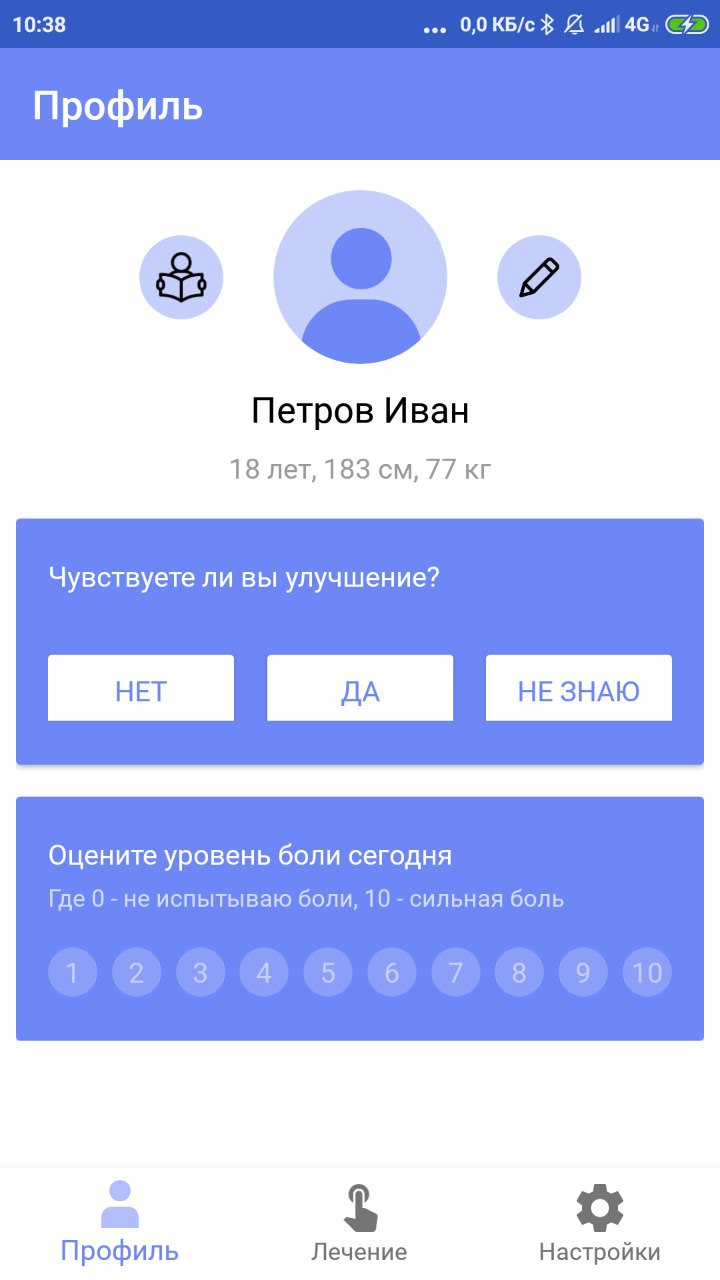
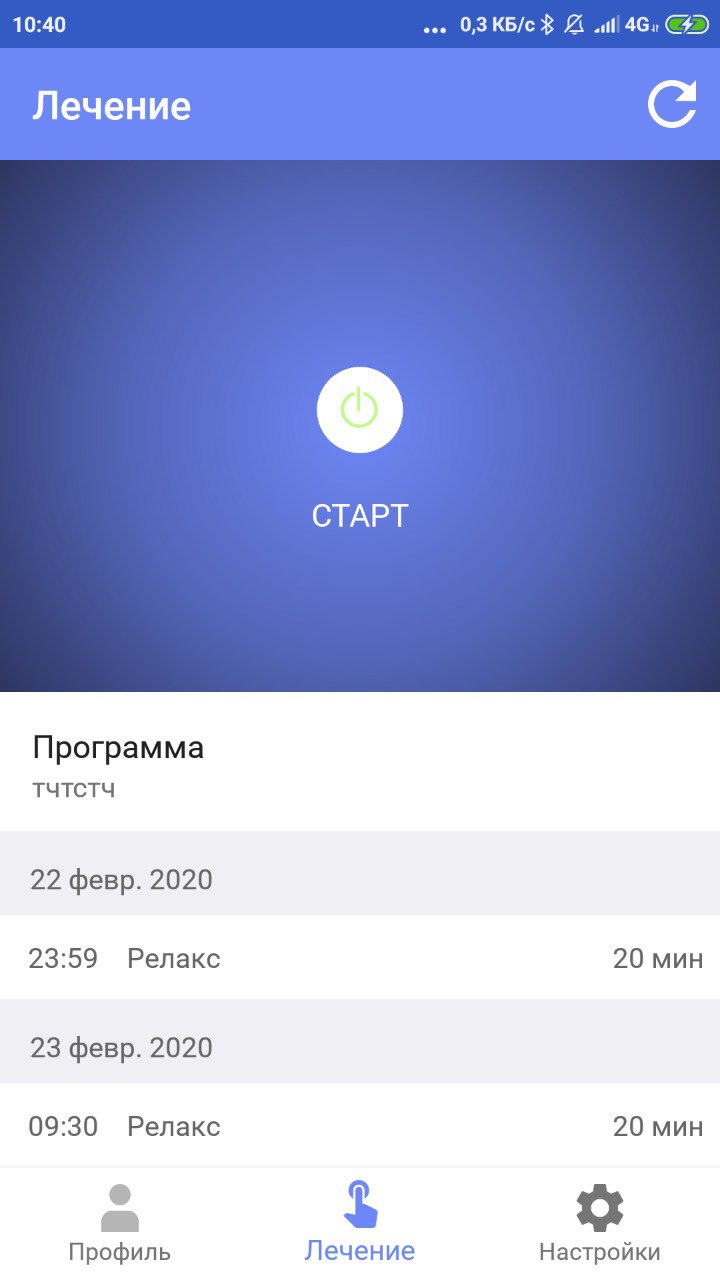
Данная парадигма предполагает введение нового типа данных – Обозреватель и Обозреваемый. Это позволяет создавать асинхронные потоки данных и получать данные из таких потоков в различных компонентах приложения.

В данном приложении методы реактивного программирования реализует библиотека AndroidX созданная на основе популярной библиотеки ReactiveX.

ReactiveX — это библиотека для составления асинхронных программ с использованием обозреваемых последовательностей. Она расширяет паттерн Обозревателя для поддержки последовательностей данных и событий, а также добавляет операторы, которые позволяют декларативно составлять последовательности команд, возлагая на себя работу с многопоточностью, синхронизацией, безопасностью потоков, параллельные структуры данных и блокировку ввода / вывода.

## 2.2 Анализ Android приложения

Данный миостимулятор имеет частично работающее мобильное приложение, написанное для операционной системе Android на языке Kotlin. К сожалению, из-за истощения бюджета разработки, работа над ним была прекращена. Требовалось написать мобильное приложение использующие все возможности устройства, и не имеющее недостатков старого приложения.

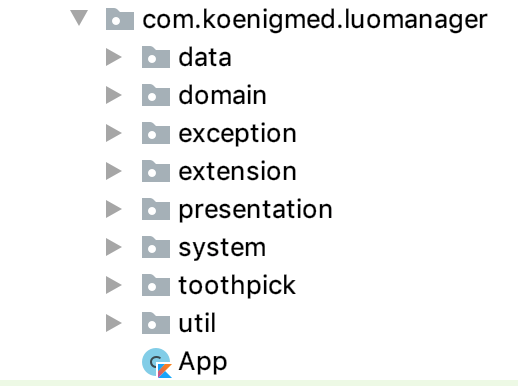
**2.2.1 Структура Android приложения**

Приложение написано на платформе Android на языке Kotlin. В приложении используется MVP (Model View Presenter) паттерн работы с интерфейсом, (DI) (Dependency injection) паттерн для внедрения зависимостей. Интерфейс приложения построен на принципе – “Одна активность — много фрагментов”. Для хранения данных приложение использует базу данных SQLite, также приложение обменивается данными с сервером через REST API в формате JSON.

Типы классов, используемых в приложении:

1. Mapper – преобразует одни объекты в другие
2. Model – модель данных
3. Repository – объект для работы с данными полученными из памяти или из сети. Объект такого класса возвращает данные в виде обозреваемого потока данных.
4. TypeConverter – конвертирует одни типы данных в другие. Используется для конвертации объектов для последующей работы с базой данных
5. Dao – интерфейс для общения с базой данных
6. Database – объект ответственный за общение с базой данных
7. Deserializer – преобразователь полученных данных в объекты
8. Serializer – преобразователь объектов в данные для последующей отправки или хранения
9. TypeAdapter – адаптер для преобразования одних типов в другие
10. Interceptor – объект перехватывающий поток данных с возможностью его модификации
11. Api – объект отвечающий за обмен данными с программным интерфейсом – сервером или устройством.
12. Interactor – объект представляющий собой, высокоуровневую прослойку для работы с программным интерфейсом, обычно использует методы Api объекта.
13. Extension – объект описывающий методы дополняющие функционал существующим объектам
14. Handler – объект отвечающий за работу с определенным типом данных
15. Presenter – объект отвечающий за работу с данными графического интерфейса приложения
16. View – интерфейс для работы с графическим интерфейсом приложения
17. State – модель описывающая состояние объекта
18. Presentation – модель хранящая данные с графического интерфейса приложения
19. Delegate – класс берущий на себе некоторые функции другого класса
20. Adapter (Списка) – объект отвечающий за вывод данных списка на экран
21. Fragment – объект работающий на прямую с графическим интерфейсом, обязан быть привязан к активности
22. Activity – объект работающий напрямую с графическим интерфейсом приложения.
23. Callback – объект функции
24. Util – объект представляющий собой утилиту для работы с данными
25. Formatter – объект отвечающий за форматирование данных
26. Widget – объект интерфейса с обособленной логикой
27. Module – объект выполняющий ‘биндинг’ для последующего внедрения зависимостей (DI)
28. Provider – объект создающий различные объекты для последующей передачи в компоненты (DI)
29. Wrapper – объект представляющий собой оболочку для работы с другим объектом
30. Helper – объект содержащий в себе вспомогательные методы

Так выглядит директория проекта



App – представляет собой главный объект приложения. В нем инициализируются основные компоненты приложения.

В директории ‘data’ содержаться объекты, отвечающие за взаимодействие с моделями данных приложения.

* В ее поддиректории ‘decice’ содержаться модели данных, отвечающие за взаимодействие с устройством и объекты, отвечающий за обмен данными с устройством.
* В поддиректории ‘mapper’ содержаться объекты, преобразовывающие одни модели данных в другие.
* В поддиректории ‘model’ храниться сами модели данных.
* В поддиректории ‘repository’ хранятся объекты для работы с данными, получаемыми из памяти устройства или из сети
* В поддиректории ‘room’ хранятся объекты, отвечающие за работу с базой данных
* В поддиректории ‘server’ хранятся объекты, отвечающие за обработку данных, приходящих с веб сервера, а также объект, отвечающий за интерфейс взаимодействия с ним

В директории ‘domain’ находятся модели данных необходимые для работы с веб сервером. А также объекты, создающие высокоуровневую прослойку для работы с устройством, а также различными сервисами веб сервера.

Объекты, отвечающие за отображение данных и взаимодействие с пользовательским интерфейсом, находятся в директории ‘presentation’. Как мы видим компоненты интерфейса отделены от данных, которые демонстрируют. Для этого в приложении реализуется паттерн MVP (Model View Presenter), при помощи библиотеки Moxy. В Android приложениях объектом, реализующим View, является Activity или Fragment, в данном приложении они находятся в поддиректории ‘ui’. В свою очередь в поддиректории ‘mvp’ хранятся объекты типа Presenter и Model. Также в ней хранятся модели описывающие данные конкретного экрана интерфейса. Польза такого подхода заключается в том, что вся работа с данными и взаимодействия с различными программными интерфейсами выносится в Presenter, тем самым облегчая процессы тестирования, а также поиска и устранения ошибок.

В директории ‘system’ находятся объекты управляющие системными ресурсами.

В директории ‘toothpick’ находятся объекты, взаимодействующие с библиотекой Toothpick, реализующей паттерн DI (Dependency Injection).

* В поддиректории ‘module’ содержаться различные группы зависимостей, которые затем могут быть использованы в различных компонентах приложения.
* В поддиректории ‘provider’ содержаться объекты, создающие другие объекты для последующего использования компонентами приложения при помощи методов DI

**2.2.2 Анализ функциональности имеющегося приложения**

Данное приложение имеет существенный ряд недостатков. В процессе тестирования было замечены:

* Перебои в работе Bluetooth соединения с устройством
* Недоработки графического интерфейса приложения
* Отсутствие реализации некоторых функций устройства

# Глава 3 Разработка нового Android приложения

## 3.1 Обзор платформ для разработки

На этапе разработки требований к мобильному приложению всегда остро стоит выбор технологии разработки. В данный момент на рынке существует две основные мобильные операционные системы – Android от Google и iOS от Apple. В связи с таким делением рынка технологии делятся на нативные и кроссплатформенные.

Нативные технологии включают в себя средства разработки от компаний, разработавших мобильные операционные системы. Например, Android SDK от Google для разработки под Android или Software Development Kit от Apple для разработки под iOS. Главными преимуществами такого подхода являются:

* Высокая оптимизация работы с ресурсами устройства – разработчики операционной системы всегда предоставляют оптимизированные библиотеки и компоненты, заточенные под работу с конкретной платформой.
* Удобство работы с аппаратными компонентами устройства.
* Возможность использовать все возможности конкретной операционной системы

Такой подход имеет и ряд недостатков:

* Необходимость поддержки нескольких программных продуктов одновременно для покрытия всего рынка
* Увеличение трудозатрат разработки в следствие использования двух различных кодовых баз

Кроссплатформенные технологии были созданы чтобы устранить главные недостатки нативных технологий и дать разработчикам возможность работать с одним инструментов с последующим созданием Android и iOS проектов из одной кодовой базы. Преимуществами такого подхода считаются:

* Работа и поддержка единой кодовой базы
* Использование единого стэка технологий для создания Android и iOS приложений

Недостатками данного подхода считаются:

* Низкий уровень оптимизации работы с ресурсами устройства – чаще всего у разработчика использующего такую технологию нету прямого доступа к исходному коду приложения для конкретной платформы. Для выполнения команд используются встроенные интерпретаторы вызывающие системные методы для конкретной платформы, что неизбежно влечет к задержке их выполнения.
* Зависимость от разработчиков кроссплатформенной технологии. При выборе технологии, существует вероятность прекращения ее поддержки. Например, в августе 2019 года Dropbox прекратил разработку своего инструмента кроссплатформенной разработки, а в декабре 2019 года прекратил свою поддержку Corona SDK – популярный инструмент разработки, насчитывавший более 500 тыс. разработчиков на момент закрытия.
* Некоторые кроссплатформенные технологии не используют нативные компоненты при работе с интерфейсом, вместо них используется OpenGL рендеринг собственных компонентов. В случаях, когда такой рендеринг не оптимизирован, можно наблюдать существенное замедление производительности интерфейса, а иногда и ухудшения качества отрисовки компонентов.
* Потребность в подключении нативных модулей для работы с нестандартными возможностями операционной системы или аппаратными возможностями устройства

Даже с учетом всех вышеперечисленных недостатков, многие клиенты нуждающиеся в мобильном приложений, предпочитают кроссплатформенную разработку. В первую очередь она позволяет им значительно снизить стоимость и увеличить скорость разработки и поддержки проекта качеству.

По состоянию на май 2020 года существует множество кроссплатформенных

технологий для разработки мобильных приложений, самыми распространенными являются:

* React Native – Фреймворк для разработки мобильных приложений на языке JavaScript с использованием нативных компонентов на базе фреймворка для разработки веб приложений React от компании Facebook.
* Flutter – Фреймворк для разработки мобильных приложений на языке Dart с использованием собственного рендеринга интерфейса и собственных компонентов похожих на компоненты Android SDK. Был создан компанией Google.

**3.1.1 Обзор фреймворка React Native**

React Native представляет собой фреймворк для разработки мобильных приложений с открытым исходным кодом, созданная Facebook. Он используется для разработки приложений для Android, iOS, Web и UWP, позволяя разработчикам использовать как кроссплатформенный код, так и писать свои нативные модули для каждой платформы под нужды своего проекта.

В 2012 году Марк Цукерберг прокомментировал: «Самая большая ошибка, которую мы сделали как компания, заключалась в том, что мы ставили слишком большие ставки на HTML, игнорируя мобильные устройства». Он пообещал, что Facebook начнет переносить свои приложения на мобильные телефоны. Затем один из разработчиков компании нашел способ создавать элементы пользовательского интерфейса для iOS из фонового потока JavaScript. В Facebook решили организовать внутренний хакатон и усовершенствовать этот прототип, чтобы иметь возможность создавать собственные приложения с этой технологией. После нескольких месяцев разработки Facebook выпустил первую версию React Native.

Принципы работы React Native практически идентичны веб фреймворку React, за исключением того, что React Native не управляет DOM через Virtual DOM. Он работает в фоновом режиме (который интерпретирует JavaScript, написанный разработчиками) непосредственно на конечном устройстве и связывается с нативным проектом через сериализованный, асинхронный и пакетный мост.

Компоненты React создают прослойку для существующего нативного кода и взаимодействуют с API операционной системы через декларативную парадигму пользовательского интерфейса React и JavaScript.

React Native не использует HTML или CSS. Вместо этого сообщения из потока JavaScript используются для управления нативными компонентами. React Native также позволяет разработчикам писать собственные нативные модули на таких языках, как Java для Android и Objective-C или Swift для iOS, что делает его еще более гибким.

Приложения написанные c использованием React Native:

* Facebook – приложение социальной сети Facebook насчитывающие миллионы пользователей.
* Instagram – приложение социальной сети Instagram.
* SoundCloud Pulse — это приложение для управления аккаунтом от ведущей аудио-платформы.
* Tesla - приложение для управления электрокаром.

**3.1.2 Обзор технологии Flutter**

Flutter - это технология для разработки программного обеспечения с открытым исходным кодом, созданный Google. Он используется для разработки приложений для Android, iOS, Windows, Mac, Linux, Google Fuchsia и веб приложений. Первая версия Flutter была известна под кодовым названием «Sky» и работала в операционной системе Android. Это было обнародовано на саммите разработчиков Dart в 2015 году, с заявленным намерением иметь возможность отображать последовательно со скоростью 120 кадров в секунду.

Основные компоненты Flutter включают в себя:

* Платформу Dart
* Flutter фреймворк
* Библиотеку Foundation
* Виджеты Flutter

Приложения Flutter написаны на языке Dart и используют наиболее продвинутые функций данного языка.

При написании и отладке приложения Flutter использует Just In Time компиляцию, что позволяет выполнить «горячую перезагрузку», с помощью которой изменения в исходных файлах могут быть загружены в работающее приложение. Flutter поддерживает горячую перезагрузку с сохранением состояния, где в большинстве случаев изменения в исходном коде могут быть немедленно отражены в работающем приложении, не требуя перезапуска или потери состояния. Эта функция, реализованная во Flutter, получила широкое признание. Приложения написанные на Flutter компилируются с опережающей (AOT) компиляцией как на Android, так и на iOS, что дает Flutter высокую производительность на мобильных устройствах.

Программный код Flutter, написан на C ++, что обеспечивает низкоуровневую поддержку рендеринга с использованием графической библиотеки Skia от Google. Кроме того, он взаимодействует со специфичными для каждой платформы SDK. Фреймворк Flutter — это портативная среда для размещения приложений Flutter. Он реализует основные библиотеки Flutter, включая анимацию и графику, файловый и сетевой ввод / вывод, поддержку специальных возможностей, архитектуру плагинов, а также среду выполнения Dart и набор инструментов для компиляции.

Приложения написанные c использованием Flutter:

* Утилита Google Ads
* Приложение Alibaba
* Приложение Birch Finance

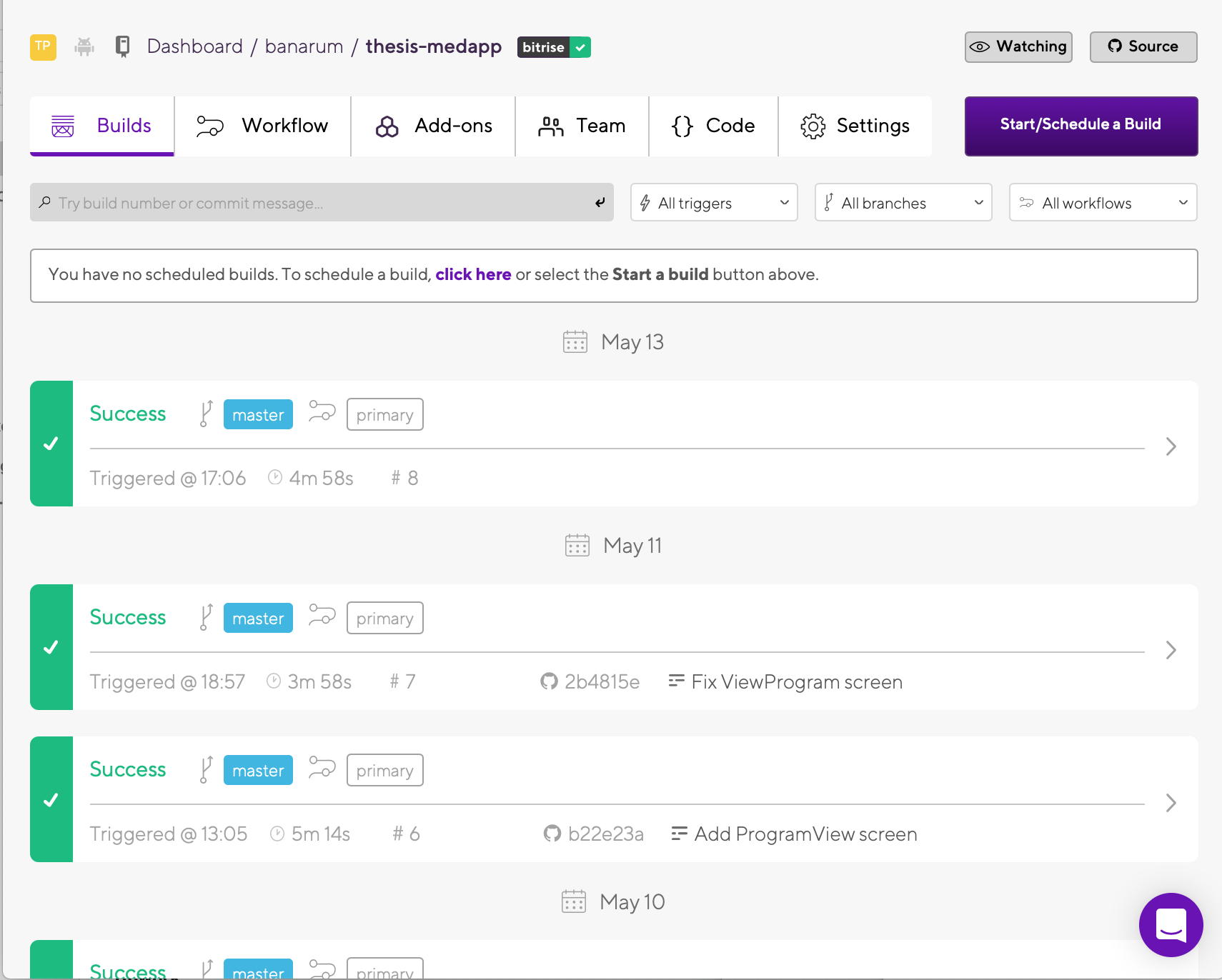
## 3.2 Подготовка к разработке мобильного приложения

В первую очередь было решено отказаться от использования кроссплатформенного фреймворка в связи с необходимостью работы с нативным модулем Bluetooth LE. Было решено использовать тот же стек технологий, что и в старом приложении.

**3.2.1 Работа с системой непрерывной интеграции**

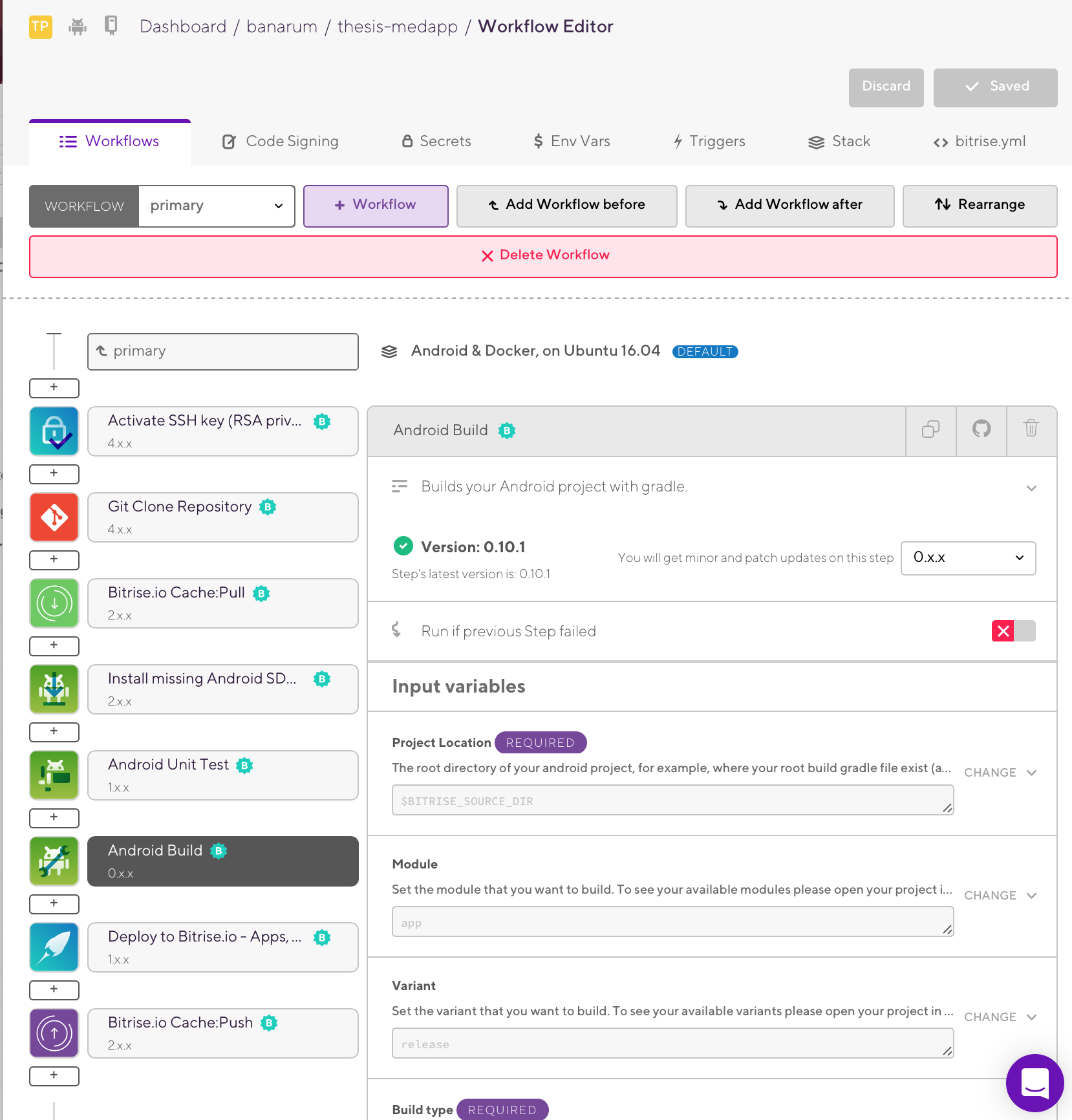
Непрерывная интеграция — практика разработки программного обеспечения, которая заключается в постоянном слиянии рабочих копий в общую основную ветвь разработки (до нескольких раз в день) и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем. В проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счёт наиболее раннего обнаружения и устранения ошибок и противоречий, но основным преимуществом является сокращение стоимости исправления дефекта, за счёт раннего его выявления.

В качестве платформы предоставляющей инструменты непрерывной интеграции была выбрана платформа – Bitrise. Главными ее преимуществами являются простота настройки цепей сборки и автоматизация многих процессов отправки приложения в магазин приложений для каждой платформы. Bitrise CI подключается к репозиторию приложения, затем требуется задать ‘ветку’, в изменения в которой будут запускать процесс сборки мобильного приложения.



В случае успешного завершения сборки приложения, каждому разработчику будет отправлено письмо по электронной почте со ссылкой, перейдя по которой можно скачать приложение. В случае, когда сборка приложения не завершилась успешно, каждому разработчику будет отправлено письмо по электронной почте с указанием ошибки в приложении.

Для сборки мобильных приложений Bitrise CI использует индивидуальные контейнеры при каждом запуске сборочного процесса, запускаемые на сервере, это гарантирует чистоту сборки и независимость от предыдущего состояния машины. Так как достаточно часто существует необходимость программирования собственных процессов сборки, в Bitrise CI существует свой конфигуратор сборки с множеством утилит для автоматизации процессов, а также возможность запускать свои скрипты.



После окончания сборки, Bitrise CI позволяет загрузить готовое приложение на свой сервер для получения ссылки, которая будет отправлена всем членам команды разработки для дальнейшего тестирования.

## 3.3 Доработка функционала взаимодействия с устройством

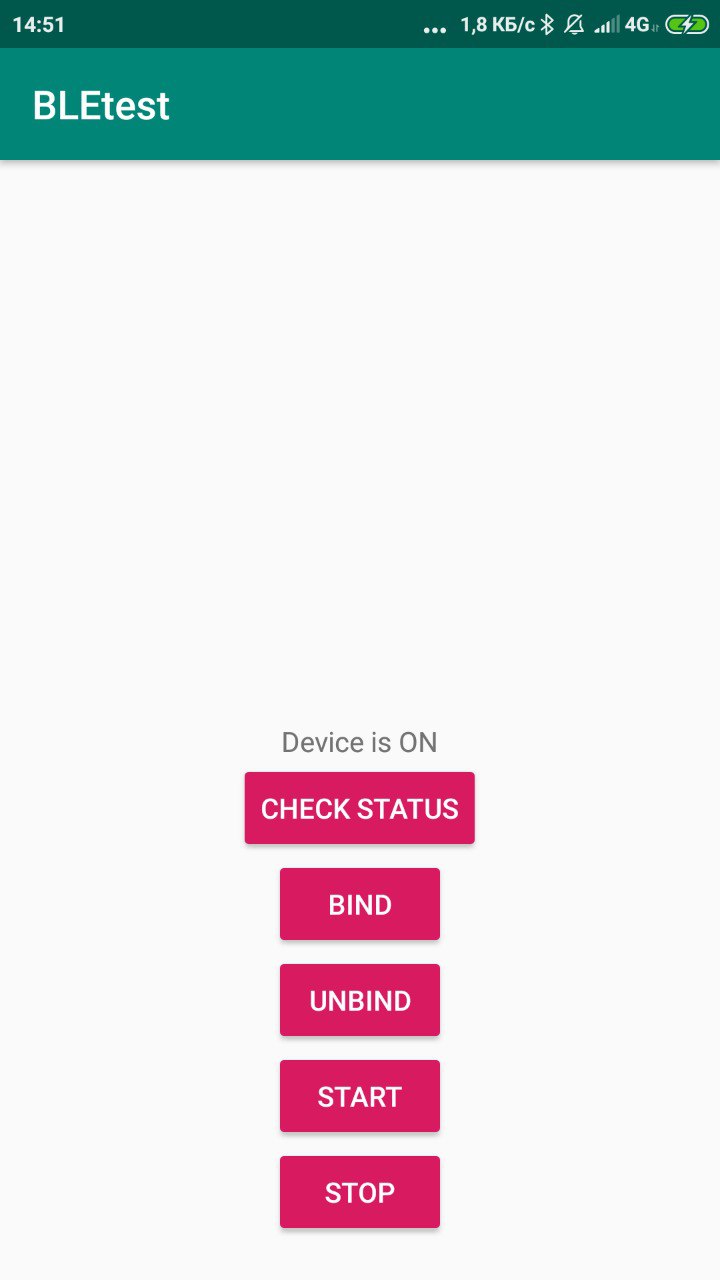
**3.3.1 Исправление ошибок обрыва соединения**

В результате тестирования было выявлено, что компонент ответственный за взаимодействие с устройством имеет неисправности.

После исследования исходного кода приложения, было выявлено что за обмен данными с устройством отвечают объекты:

* domain/interactor/device/BtInteractor.kt – отвечает за работу BLE соединения, поиск и подключение к устройству. Данный объект использует шаблон проектирования – Singleton для того, чтобы инициалироваться только один раз за время работы приложения.
* data/decice/DeviceApi.kt – отвечает за взаимодействие приложения с устройством. При создании получает в себя объект, отвечающий за GATT протокол соединения, который использует для взаимодействия с устройством.
* domain/interactor/device/DeviceInteractor.kt – отвечает за вызов некоторых функций устройства, таких как синхронизация времени и проверка доступности устройства.

Для выявления ошибок данной системы было проведено модульное тестирование вышеперечисленных компонентов. Данные компоненты были перенесены в пустое приложение.



При нажатии на кнопку ‘Bind’ приложение устанавливает связь с устройством, в то время как при нажатии ‘Unbind’ – разрывает эту связь. Кнопка ‘Start’ запускает программу лечения на устройстве, в то время как кнопка ‘Stop’ останавливает программу лечения.

Далее было проведено нагрузочное тестирование. Приложение подключилось к устройству, и было оставлено в таком состоянии на десять часов. Через шесть часов произошел разрыв соединения с устройством. Просмотрев журнал событий, была найдена ошибка в функции объекта DeviceApi:

fun onCharacteristicChanged(chunk: String) {  
 responseString += chunk  
 if (responseString.isValidJson()) {  
 val response = gson.fromJson(responseString, JsonDeviceResponse::class.java)  
 if (response.isOk()) {  
 progressCallback.invoke(Progress(100))  
 } else {  
 isError = true  
 }  
 responseString = ""  
 }  
}

Данная функция получает данные от устройства, и записывает их в строку – responseString, каждый раз проверяя пришел ли полный Json объект используя функцию .isValidJson(). Если функция возвращает ‘true’, создается Json объект типа JsonDeviceResponse. Ошибка заключалась в вызове функции .isOk() при несуществующем объекте – response. Причиной ошибки является неполноценность функции .isValidJson().

fun String.isValidJson(): Boolean {  
 if (*isEmpty*()) return false  
 var openCount = 0  
 var closeCount = 0  
 for (c in this.*toCharArray*()) {  
 if (c in "{") {  
 ++openCount  
 } else if (c in "}") {  
 ++closeCount  
 }  
 }  
 return openCount == closeCount  
}

Данная функция проверяет количество открывающихся и закрывающихся скобок, в случае если их количество совпадает Json считается корректным. Такой подход удобен в случае когда надо часто проверять корректность Json, но иногда он может давать сбой. Для полноценной проверки Json, в таких случаях была добавлена проверка на наличие объекта в функции onCharacteristicChanged.

…

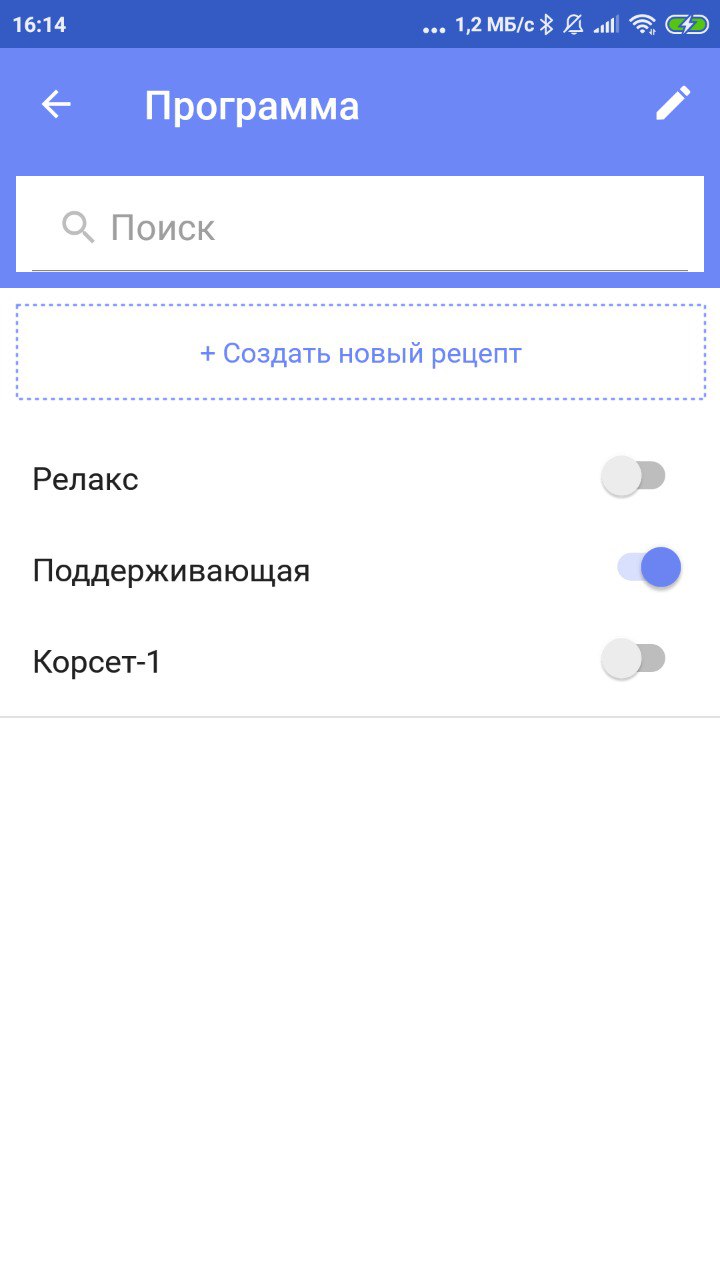
if (response != null && response.isOk()) {  
 …  
} else {  
 …  
}

После исправления данной ошибки было проведено повторное нагрузочное тестирование, в результате которого не было найдено других ошибок, связанных с обрывом соединения.

## 3.4 Доработка интерфейса приложения

**3.4.1 Добавление возможности редактировать существующую программу лечения**

Одним из недостатков интерфейса мобильного приложения было отсутствие возможности изменять уже существующие программы лечения. Также одним из неудобств является интерфейс записи программы на устройство. Пользователю нужно нажать на один из переключателей, чтобы запустить одну из программ.

****

Было решено создать новый экран, открывающийся по нажатию на программу из списка. На данном экране пользователь сможет запустить, редактировать или удалить существующую программу.

Для начала был видоизменен вид элемента списка, таким образом что названия программы было смещено в центр, а индикатор работы программы, был удален. Затем были созданы объекты: ViewProgramPresenter, ViewReceiptView, ViewProgramFragment. Элемент списка был запрограммирован на то, чтобы по нажатию запускать ViewProgramFragment и передавать id программы.

Для вывода на экран графического представления программы лечения, требовалось получить модель ReceiptPresentation. После анализа исходного кода было обнаружено, что за доступ к данным существующих программ отвечает компонент ProgramInteractor. Данный объект выдает модели типа MyoProgram. Требовалось написать функцию конвертирующую модель MyoProgram в ProgramInteractor. Для конвертации моделей программ одного типа в другие используется объект ProgramMaper. В нем была написана функция конвертации MyoProgram в ReceiptPresentation.

fun mapToReceipt(program: MyoProgram): ReceiptPresentation {  
 val result = ReceiptPresentation()  
 result.name = program.name  
  
 result.channel1Data = mapToChannelData(1, program.myoProgramMyoTaskList!!)  
 result.channel2Data = mapToChannelData(2, program.myoProgramMyoTaskList)  
  
 result.executionTimeS = program.executionTimeS  
 result.programType = program.programType  
 if (program.startTimes!=null) {  
 if (program.startTimes.size == 1) {  
 result.startTime = program.startTimes[0]  
 result.endTime = program.startTimes[0]  
 } else {  
 val duration = Duration.between(program.startTimes[0], program.startTimes[1]).toMinutes()  
 result.startTime = program.startTimes[0].minusMinutes(duration)  
 result.endTime = program.startTimes.*last*()  
 }  
 }  
 return result  
}

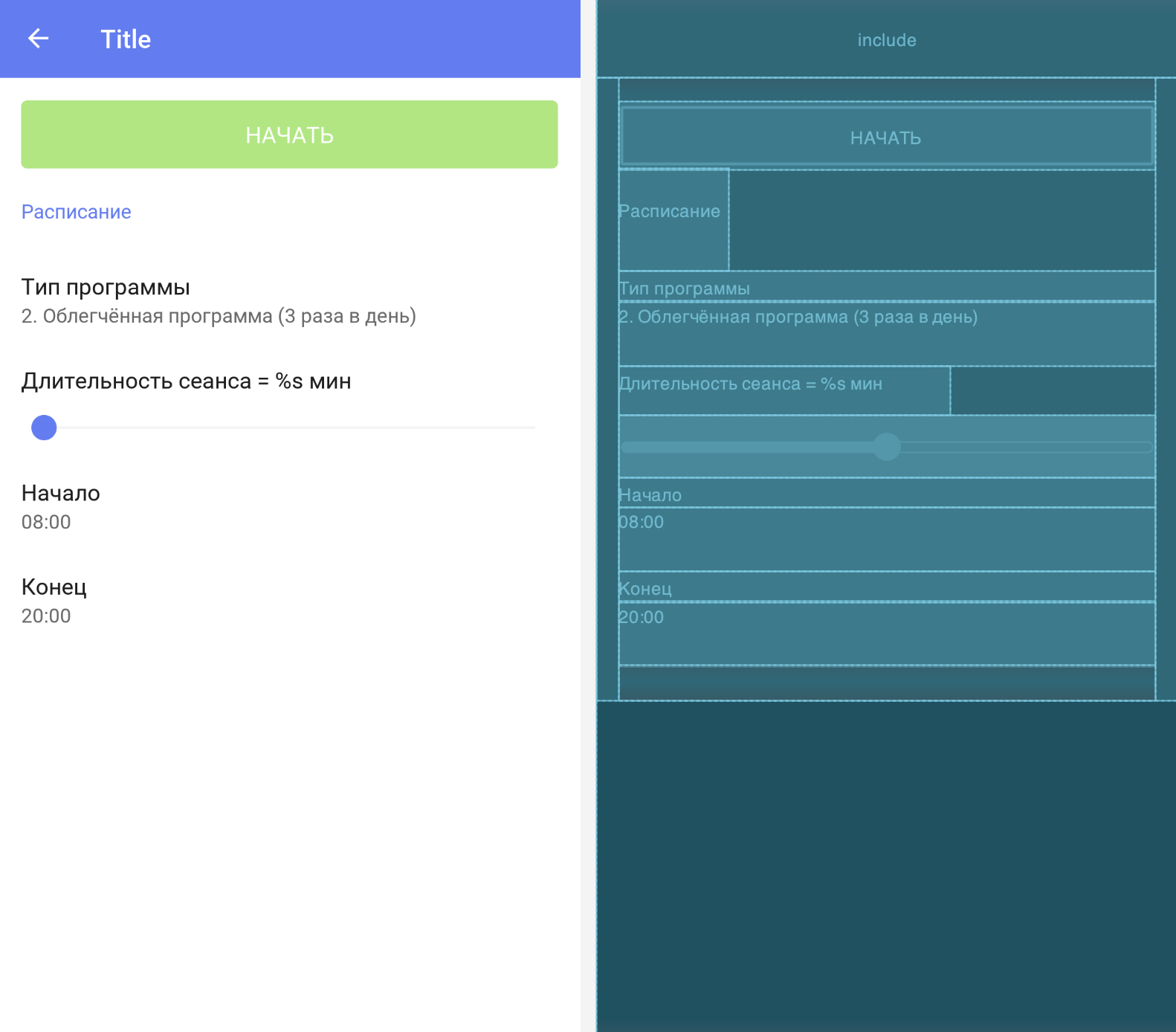
Затем была создана функция внутри объекта ProgramInteractor, получающая программу из базы данных по ее идентификатору, конвертирующая и выдающая модель ReceiptPresentation в виде обозреваемого потока.

fun getReceiptPresentation(programId: String) = programRepository.getProgram(programId)  
 .subscribeOn(schedulers.io())  
 .observeOn(schedulers.ui())  
 .map {programMapper.mapToReceipt(it)}

Затем внутри ViewProgramPresenter была написана функция получающая модель ReceiptPresentation из ProgramInteractor и передающая все нужные данные в ViewProgramFragment используя интерфейс viewState типа ViewReceiptView.

fun setProgramPresentation(programId: String) {  
 this.programId = programId  
 programInteractor.getReceiptPresentation(programId)  
 .subscribe { receiptPresentation ->  
 this.receiptPresentation = receiptPresentation  
 this.setInfo(receiptPresentation)  
 this.setType(receiptPresentation.programType)  
 setChannels(receiptPresentation)  
 }  
}

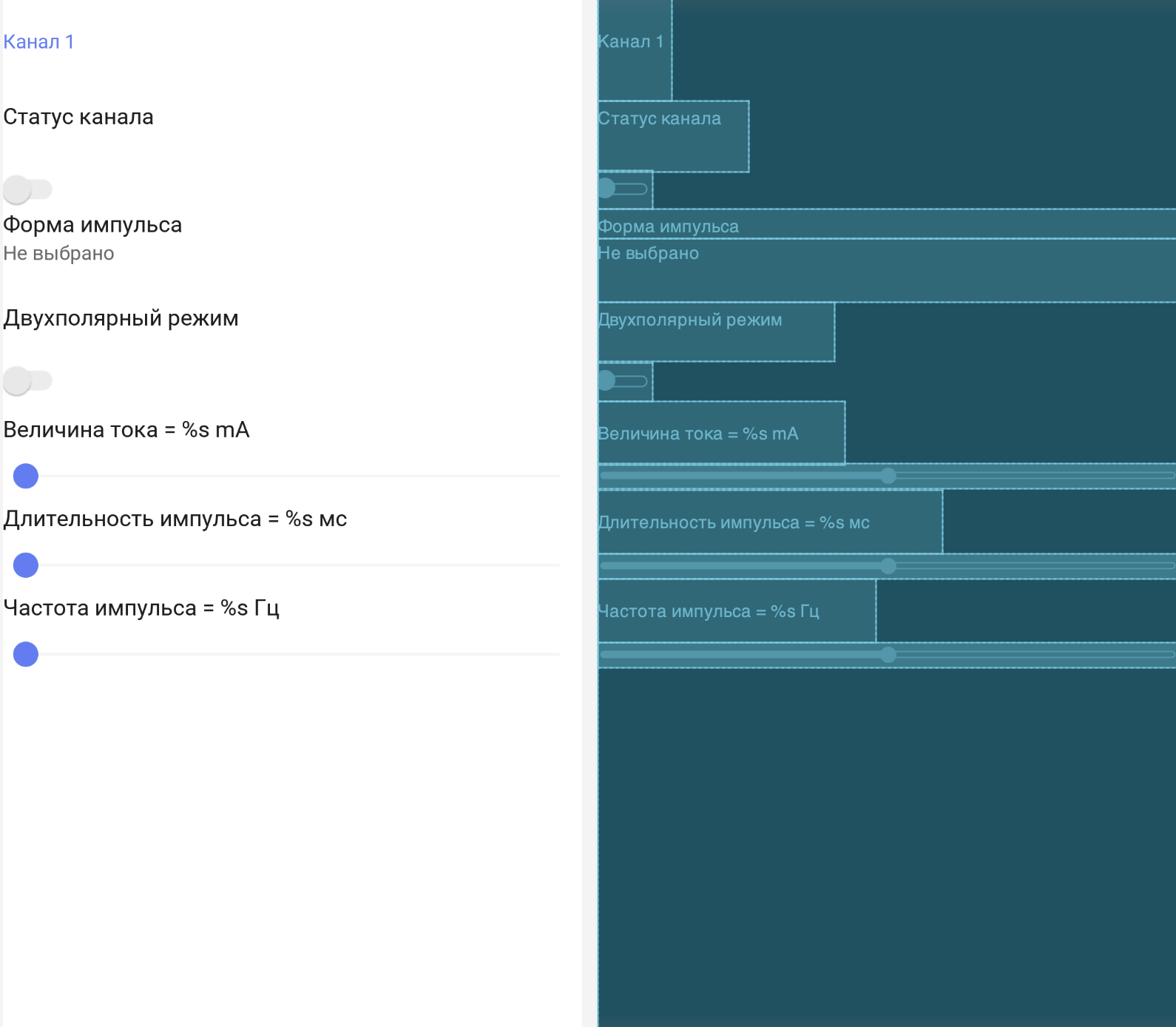
Для отображения интерфейса был создан файл разметки – fragment\_view\_program.xml



По нажатию кнопки – “Начать”, приложение программирует полученную программу на устройство. Для этого используется функция, переходящая на экран записи программы, и передающая туда значение идентификатора программы:

fun onStartProgram() {  
 router.navigateTo(Screens.SYNC\_SCREEN, this.programId)  
}

На данном экране пользователь может изменить тип программы, длительность сеанса, а также время начала и конца сеанса терапии. Также пользователь может просмотреть форму импульса, величину тока, частоту, а также длительность импульсов различных каналов, без возможности их изменить.



Еще одной функцией этого экрана является возможность удалить программу из памяти. Это действие осуществляется функцией, вызывающей метод объекта ProgramInteractor, позволяющий удалить одну или несколько программ из памяти зная их идентификаторы:

fun deleteProgram(){

programInteractor.deletePrograms(

*mutableSetOf*(this.programId!!)

).subscribe({  
 router.exitWithResult(

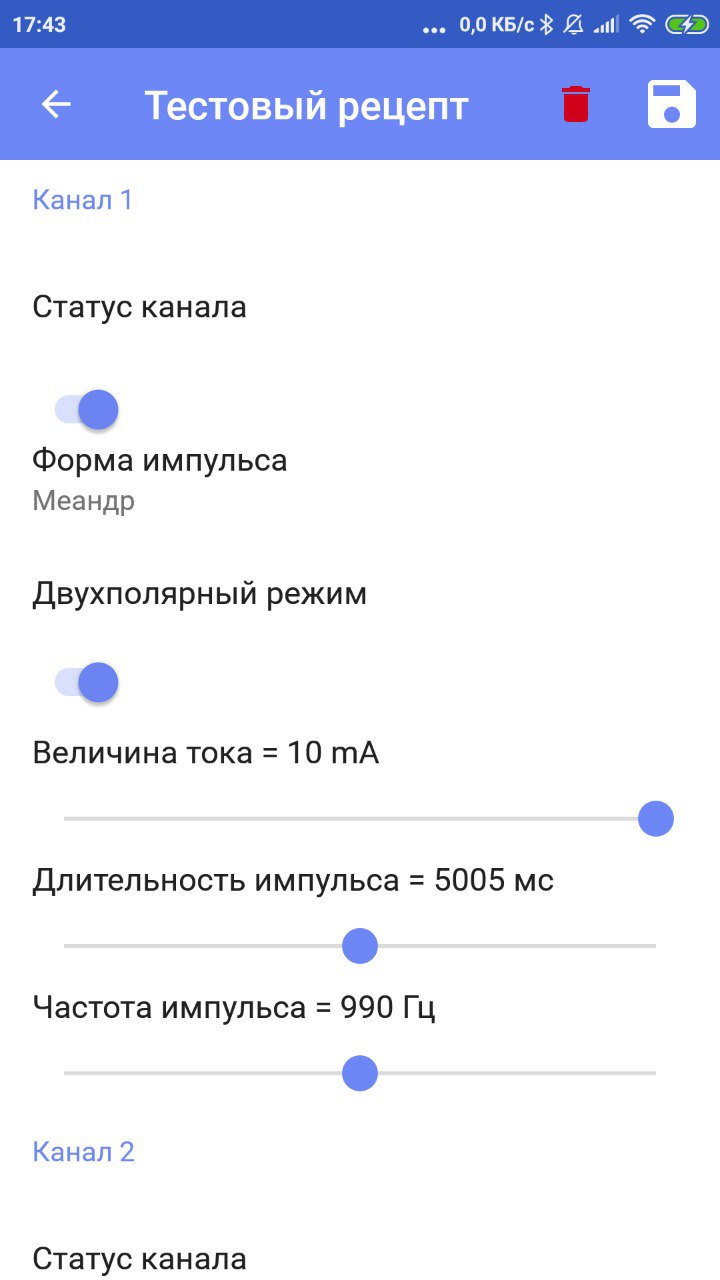
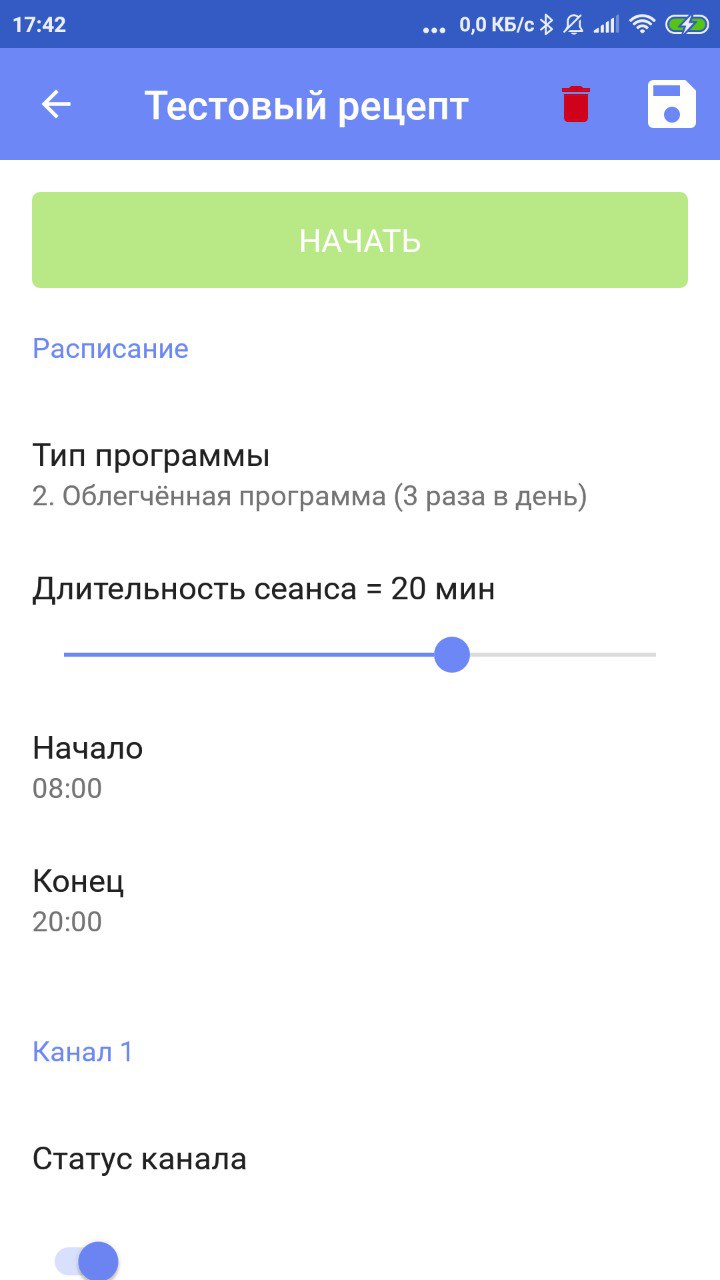
Screens.RESULT\_CODE\_PROGRAM\_ADDED, null

)  
 },{Timber.e(it)})

}

В случае успеха, пользователь перейдет к списку программ, в противном случае, будет отображено сообщение об ошибке.

После реализации всего вышеперечисленного функционала итоговый экран выглядит так:



**3.3.1 Добавление индикатора подключения к устройству**

Одним из главных недостатков интерфейса является отсутствие индикатора подключения к устройству. Так как большая часть функционала приложения недоступна при отсутствии подключения к устройству, пользователю необходимо знать статус подключения.

В начале работы был проведен анализ объекта BtInteractor, отвечающего за подключение к устройству. Данный объект имеет паттерн Singleton. При создании объект использует BluetoothAdapter для запуска сканирования сети на наличие устройств. Для нахождения устройства используется фильтрация по UUID сервиса. После нахождения устройства, производится подключение по GATT интерфейсу. При создании GATT интерфейса, мы передаем на вход объект gattListener, который получает данные о состоянии соединения через функции:

* onConnectionStateChange – получает данные о состоянии GATT соединения
* onServicesDiscovered – используется для получения информации о доступных сервисах устройства
* onCharacteristicRead – вызывается при чтении характеристики
* onCharacteristicWrite – вызывается при записи характеристики
* onCharacteristicChanged – вызывается при изменении характеристики

Так как данные о состоянии соединения приходят в функцию onConnectionStateChange, следовательно именно в ней должна быть написана логика изменения состояния соединения.

Для передачи информации о состоянии соединения с устройством было решено воспользоваться паттерном Наблюдатель. При использовании такого паттерна проектирования в классе источника изменений создается объект типа Observable. Объекты, которым требуется получать данные о происходящих изменениях, имеют возможность ‘подписаться’ на изменения используя объект типа Observer.

Для реализации паттерна Наблюдатель, была использована библиотека RxAndroid. В начале был создан объект stateEmmiter типа ObservableEmmitor, который используется для изменения состояния объекта, затем был создан объект stateObservable типа ConnectableObservable. Данный тип используется для создания объекта за изменением состояния которого могут следить сразу несколько наблюдателей.

private lateinit var stateEmmiter: ObservableEmitter<Int>  
var stateObservable: ConnectableObservable<Int> = Observable.create<Int> { emitter -> stateEmmiter = emitter}.publish()

Для изменения состояния объекта используется функция

stateEmmiter.onNext(newState)

После реализации системы оповещения об изменении состояния соединения с устройством, требовалось реализовать графический интерфейс для него. Для его отображения был выбран объект Toolbar, находящийся вверху экрана.

При попытке подключения к устройству, должен показываться анимированный индикатор загрузки. При успешном или неуспешном подключении к устройству, должна показываться соответствующая иконка.

Для реализации анимированного индикатора загрузки, был использван стандартный компонент – ProgressBar. Для отображения иконки используется стандартный компонент – ImageView.

Следующим этапом стало создание объекта Observer, который будет ‘подписываться’ на объект типа Observable, для получения данных об изменении состояния соединения и последующего вывода на экран. Внутри класса TreatmentPresenter, в функции, вызываемой при подключении объекта типа Presenter к объекту типа Fragment, через интерфейс типа TreatmentView.

btInteractor.stateObservable.observeOn(schedulers.ui()).subscribe {  
 when (it) {  
 BtInteractor.BT\_CONNECTION\_PROGRESS ->  
 *viewState*.setLoading(true)  
 BtInteractor.BT\_CONNECTION\_ACTIVE ->  
 *viewState*.setLoading(false, true)  
 BtInteractor.BT\_CONNECTION\_INACTIVE ->  
 *viewState*.setLoading(false, false)  
 }  
}

Далее были внесены изменения в интерфейс TreatmentView:

fun setLoading(isLoading: Boolean, isSuccess: Boolean = false)

И затем была написана реализация этой функции в объекте TreatmentFragment:

override fun setLoading(isLoading: Boolean, isSuccess: Boolean) {  
 toolbar\_progress\_bar.*visibility* = if (isLoading) View.*VISIBLE* else View.*GONE* toolbar\_device\_fail.*visibility* = if (isLoading || isSuccess) View.*GONE* else View.*VISIBLE* toolbar\_device\_success.*visibility* = if (isLoading || !isSuccess) View.*GONE* else View.*VISIBLE*}

После всех перечисленных манипуляций, функционал отображения индикатора состояния соединения был успешно реализован.

