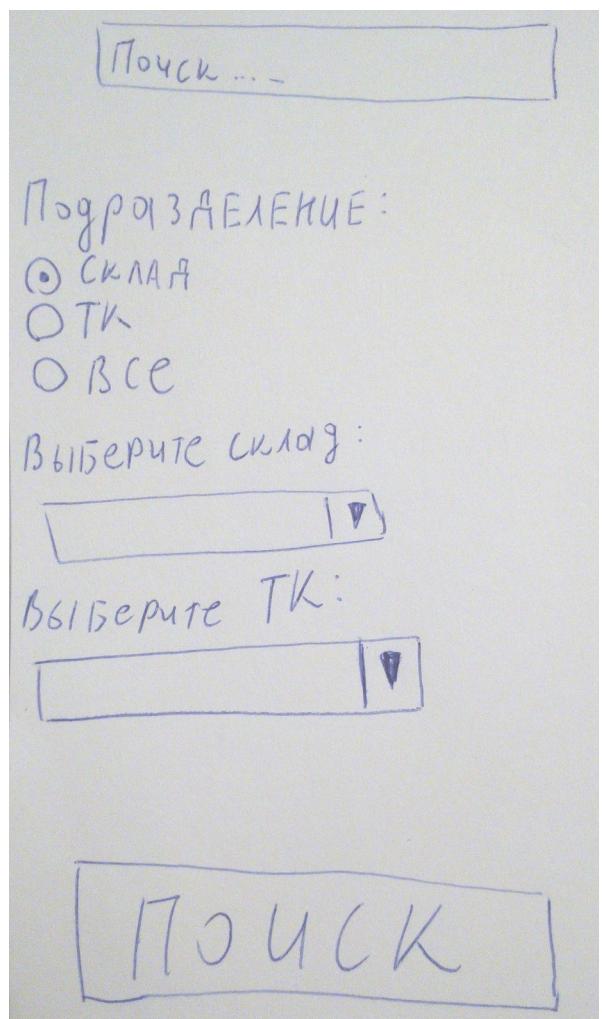
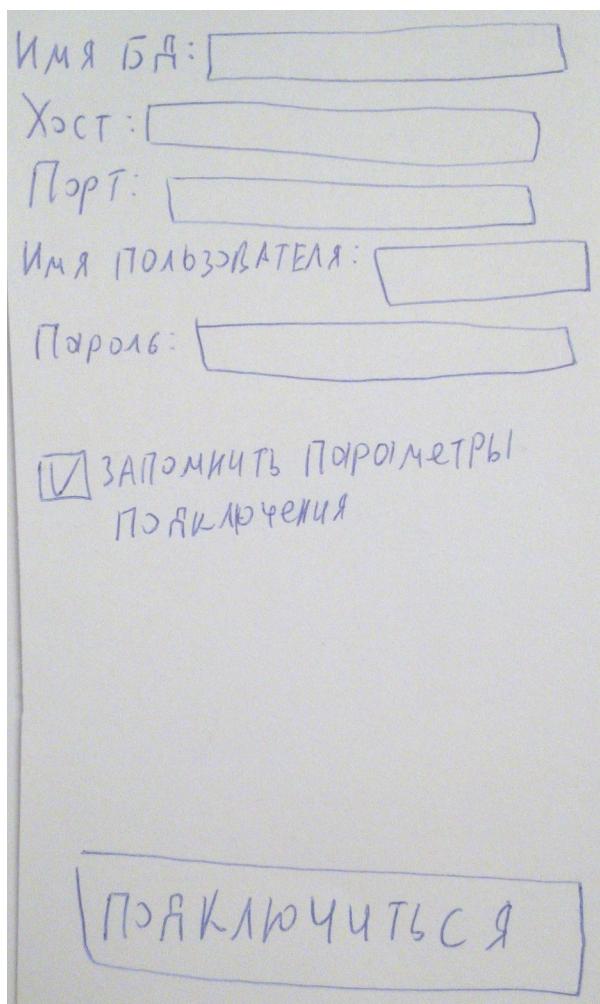


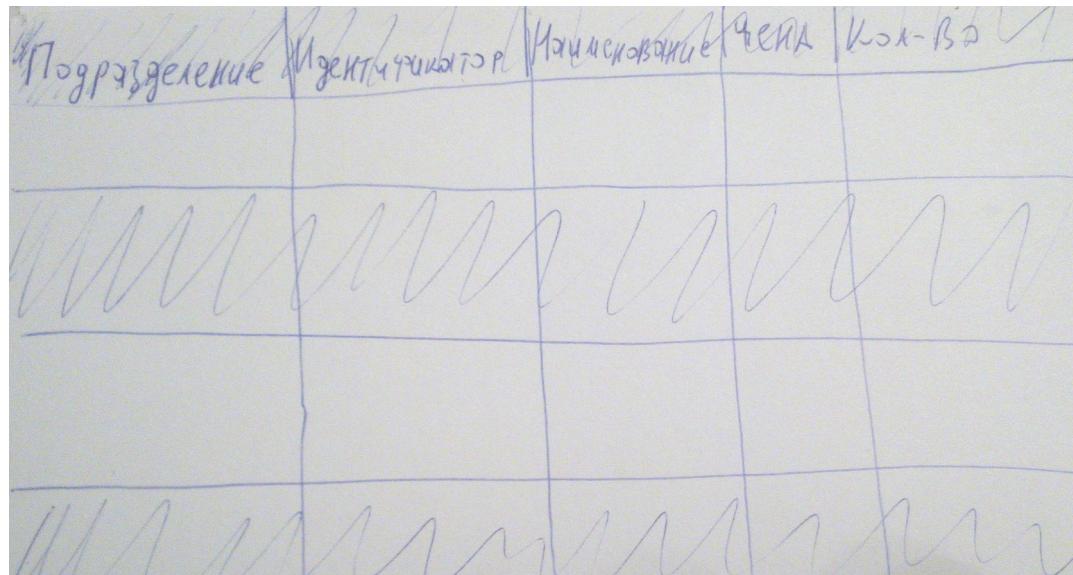
LightSearch Android

[07.09.2018]

Итак, создав клиента для ПК, я решил создать клиента для Android. До этого я знал, что для Android существует Android Studio, и имел небольшое представление о том, как пишутся приложения для этой операционной системы: где-то год назад я написал приложение Sudoku Solution — простенькая программа, которая решала за пользователя судоку. Но теперь мне предстояло написать программу на другом уровне, и я приступил.

Мощности моего ноутбука не хватило для Android Studio, поэтому я установил ее на свой ПК и стал изучать. Через несколько дней у меня уже получился примерный экран авторизации и экран поиска товара. Я себе представлял программу так:

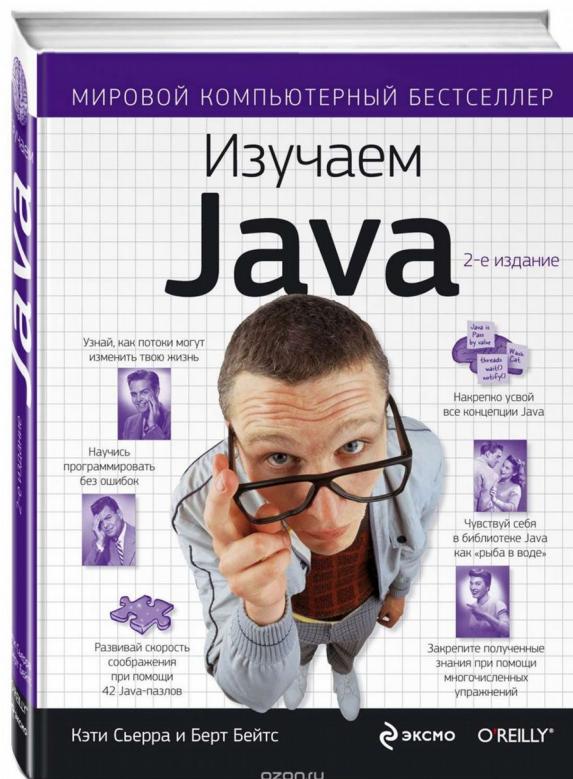




На первой картинке экран авторизации, на второй — экран поиска, на третьей — экран вывода результата.

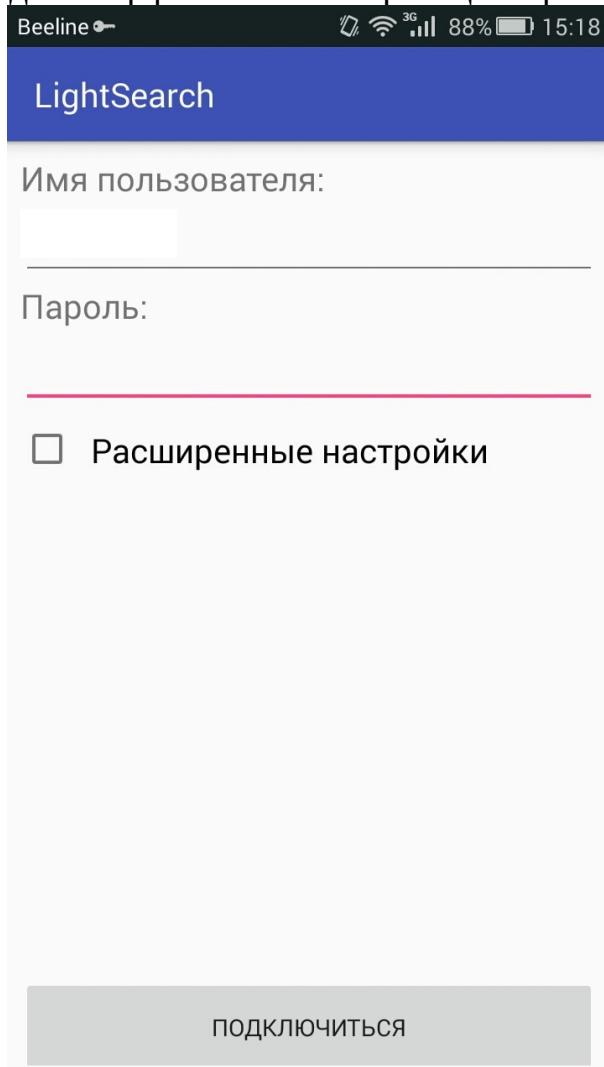
И вот, сделав такой интерфейс в своем проекте, я подключил JDBC драйвер в мой проект через Gradle — и это была моя первая ошибка. В Android используется немного другая версия Java, как я выяснил позднее, и не поддерживает JDBC-драйверы. И тогда я стал в тупик. Я стал искать Jaybird на Android — и нашел какой-то проект, который занимался этой задачей, но он давно не обновлялся и работал очень странно. Поэтому я стал думать, как можно решить данную проблему.

И я вспомнил о том, что мой наставник по языку Java (да, у меня такой есть, и я очень рад этому :)) посоветовал мне книгу как старт для программирования на этом языке: «Изучаем Java» авторов Кэти Съерра и Берт Бейтс:



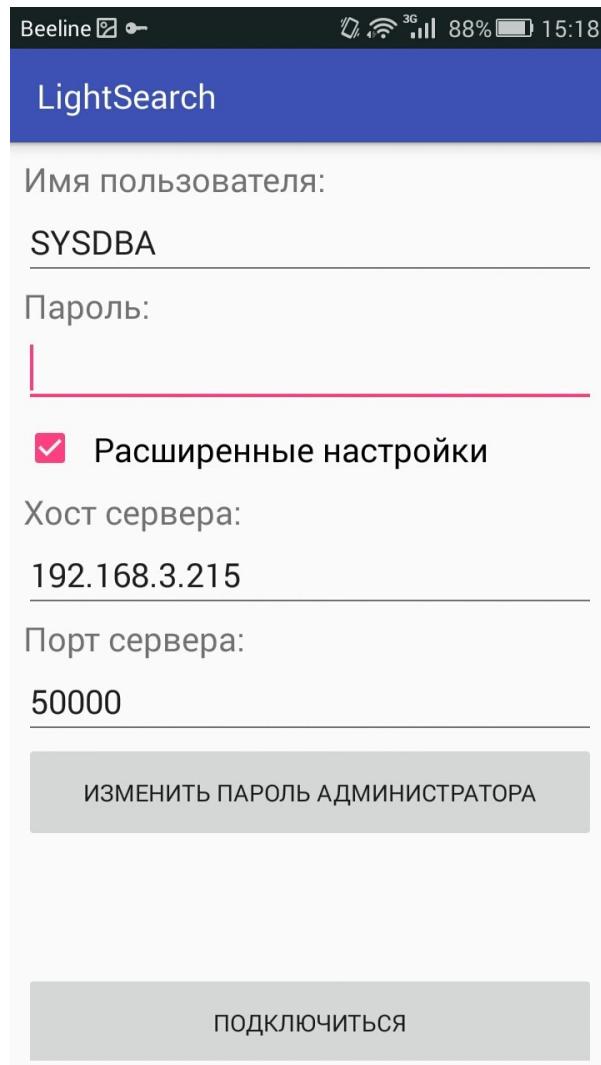
И он мне говорил, что там очень хорошо написано про сокеты. И тогда меня осенило: можно сделать сервер на Java, который будет общаться с моим Android-клиентом, и делать всю работу, связанную с бд! Сомнений было много: а заработает ли это? Как мне подключаться к серверу? Все ли получится? Но глаза боялись, а руки стали уже делать — конечно же, сначала я приступил к написанию сервера. Описание того, как я делал сервер можно прочитать в документе LightSearchServer ([ссылка](#)).

И вот, сервер более-менее написан, ушло примерно недели две, и теперь можно приступить к Android-клиенту. Первое, что я сделал — изменил интерфейс окна авторизации. Я понял, что клиенту ни к чему знать имя бд, хост, порт — это должен знать сервер. Что необходимо знать пользователю — это имя пользователя и пароль. Но мой руководитель практики подсказал мне, что можно сделать функцию, открывающую дополнительные настройки (хост и порт сервера), и доступ к ней сделать через пароль, который будет знать администратор. И тогда интерфейс окна авторизации принял следующий вид:



Флажок «Сохранить параметры подключения» я также убрал, так как нашел более лучшую альтернативу — использовать SharedPreferences. Очень удобно и довольно безопасно. Из параметров подключения пользователя я храню в ней лишь имя пользователя.

Потом я подумал, как можно сделать расширенные настройки администратора, и сделал их через флажок, придумав следующий механизм: при первой инициализации приложение попросит ввести пароль администратора. Потом она запомнит его SHA256-хэш в Shared Preferences. И теперь при нажатии на данный флажок будет выскакивать окно с вводом пароля администратора. После этого откроются дополнительные настройки:



Как видно на скриншоте выше, открываются не только поля хоста и порта сервера: еще и добавляется кнопка «Изменить пароль администратора». В первый раз я ее забыл добавить, но потом вспомнил о ней :)

В поле хост сервера я поставил фильтр, чтобы можно было вводить только строку в виде ip-адреса. Для этого я добавил в xml-файл макета окна авторизации в элементе поля ввода хоста сервера следующий код:

```
android:inputType="number"  
android:digits="0123456789."
```

И все заработало!

Для порта добавил следующий код:

```
android:inputType="number"
```

Этого вполне достаточно, ведь этими функциями будет пользоваться только администратор, и дополнительная фильтрация по моему мнению не нужна.

Теперь я приступил к окну поиска. И тут я столкнулся с проблемами. Первая проблема заключалась в том, что Android по умолчанию запрещает для приложения доступ в интернет. Я прочитал в интернете по поводу этой проблемы, и она решается довольно легко: необходимо лишь добавить в манифест приложения вот такую строчку:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Теперь проблема касалось того, что Android не позволяет работать в одном потоке UI и сокеты — для сокетов нужен отдельный поток. Хорошо, сделал отдельный поток. Но вот снова незадача: программа ругается на то, что я использую объекты UI в этом потоке, а ведь мне надо как-то вывести ошибку на экран, если что-то случится — нет соединения, неверное имя пользователя или пароль и т.д. И я снова пошел в интернет за решением данного вопроса, и там посоветовали два решения — через асинхронные задачи или через метод runOnUiThread. Я выбрал второй вариант по причине того, что он мне был более понятен.

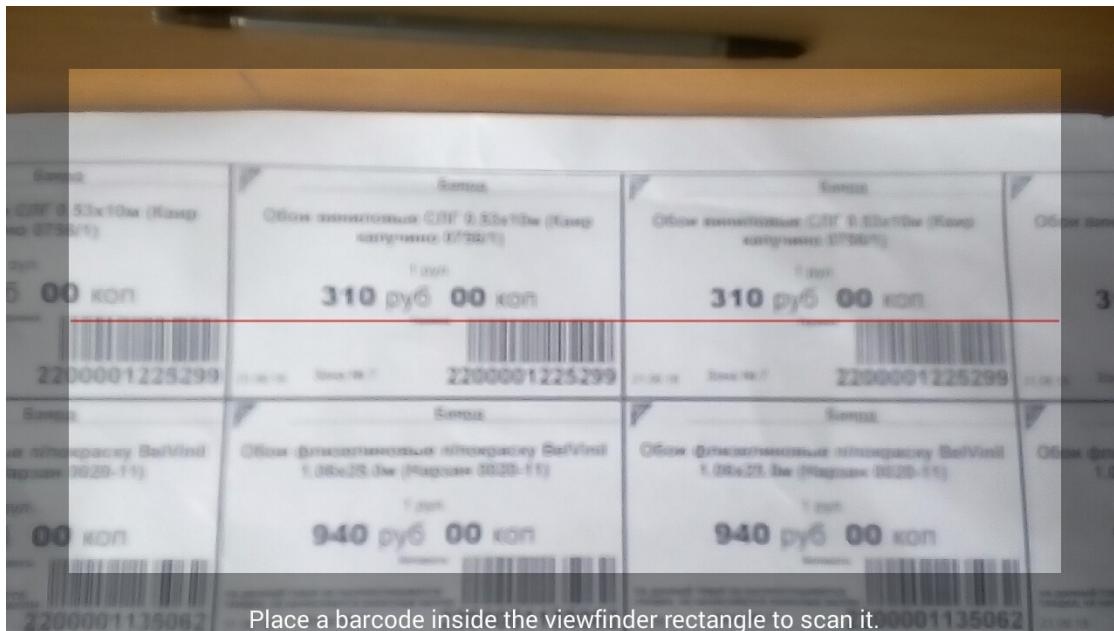
И вот: все написано, можно тестировать. Тестировал на своем смартфоне. Подключился к VPN компании через свой смартфон, чтобы можно было подключиться к серверу. Конечно, первый тест не прошел успешно, как и второй, и третий, и четвертый: было много мелких ошибок и недочетов, и я их исправлял по мере их «всплыивания». Больше всего конечно потратил времени на устранения недочетов между взаимодействием клиента и сервера, ведь как я описывал в документе про сервер([ссылка](#)), все построено по принципу «команда — ответ», и бывало такое, что команду не так написал, или JSON не так построил. И на третий экран, где отображается таблица, тоже потратил достаточно времени — изучал материалы по созданию динамических столбцов в Android, создание заголовочных столбцов, выравнивание таблицы, чтобы информация в ячейках читалась удобно, ну и чтобы таблица выглядела аккуратно. В итоге у меня получилось, но информация не вся помещалась в ячейки. И тогда я придумал следующую штуку: если нажать на строчку, то высветится окошко, в котором будет подробно написана информация.

Реализовать было ее не так проблематично, и все работало отлично.

Я показал программу своему руководителю практики — и он сказал, что в данном клиенте нет необходимости искать по наименованию или по части наименования, а достаточно лишь искать по штрих-коду. И дал мне еще одно задание: реализовать считывание штрих-кода через камеру смартфона.

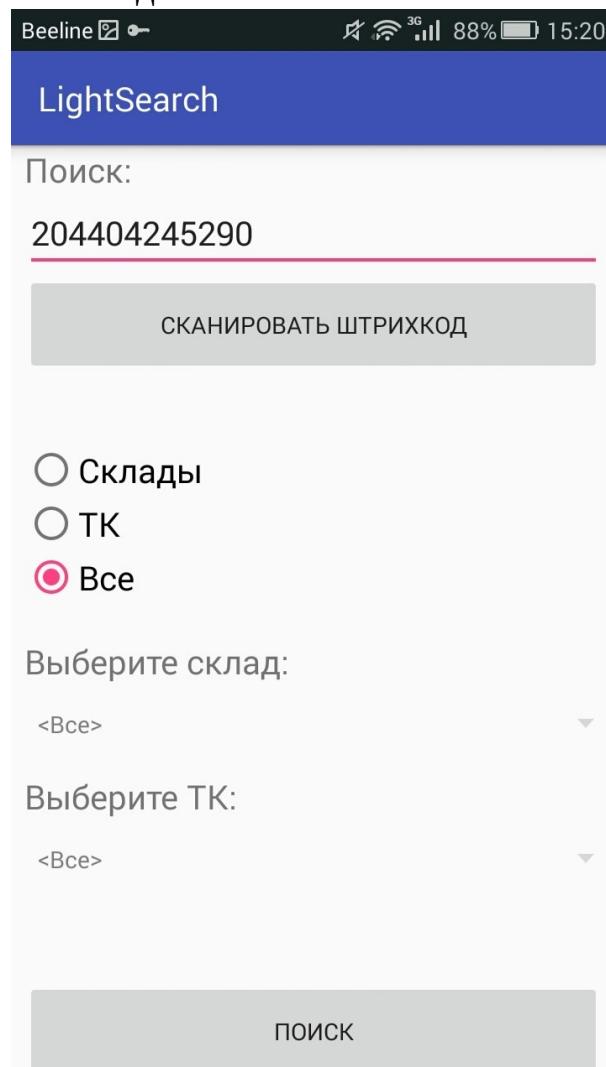
В интернете я нашел бесплатную библиотеку Zxing. Подключил ее к своему проекту, посмотрел, как его активировать, добавил кнопку «Считать штрихкод» на экран поиска, привязал к ней обработчик — и вот результат:

Все



работало! Ах, да, чуть не забыл: для поля поиска теперь нужно добавить условие, что можно вводить только цифры, и делать проверку на то, чтобы их было введено не менее пяти.

Теперь окно поиска стало выглядеть так:



А экран таблица — вот так:

The screenshot shows a mobile application interface with a blue header bar containing the text "LightSearch". Below the header is a table with the following columns: Подразделение (Department), Идентификатор (Identifier), Наименование (Name), Цена (Price), and Кол-во (Quantity). The table contains three rows of data:

Подразделение	Идентификатор	Наименование	Цена	Кол-во
Склад Комплекс 6	1135062	Обои флизелиновые п/ покраску BelVinil 1.06x25.0м (Нарзан 0020-11)	940.0	13.0
Склад Комплекс 4	1135062	Обои флизелиновые п/ покраску BelVinil 1.06x25.0м (Нарзан 0020-11)	940.0	1.0
Склад 200	1135062	Обои флизелиновые п/ покраску BelVinil 1.06x25.0м (Нарзан 0020-11)	940.0	0.0

Если нажать на строчку — то получаем вот такой результат:

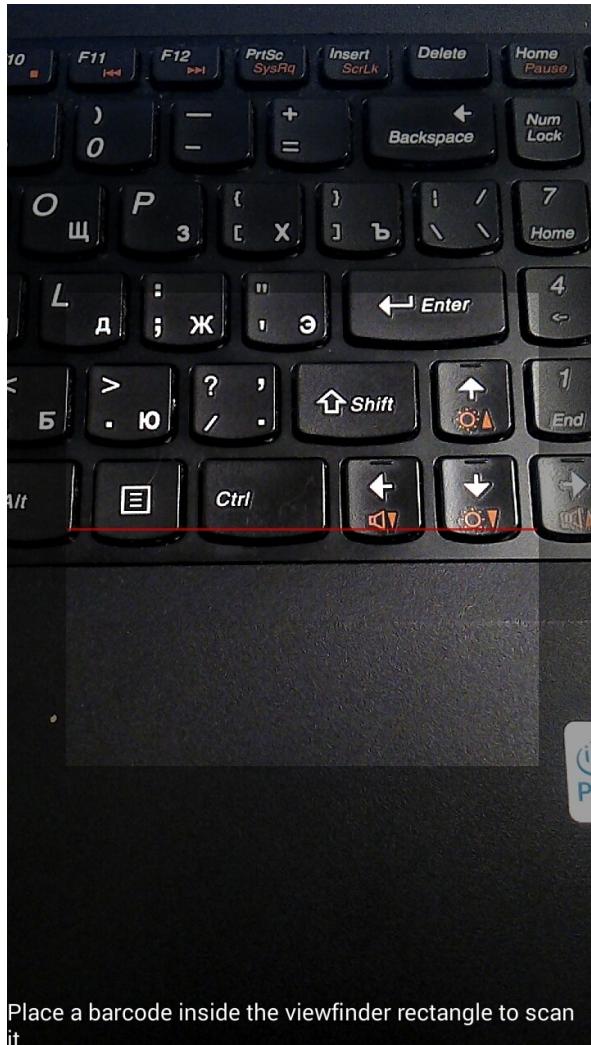
The screenshot shows a mobile application interface with a blue header bar containing the text "LightSearch". Below the header is a table with the same columns as the previous screenshot. A white callout box is overlaid on the screen, containing detailed information about the first row:

Подразделение: Склад Комплекс 6
(Проспект 60-лет Октября 178)
ИД: 1135062
Наим-ние: Обои флизелиновые п/покраску
BelVinil 1.06x25.0м (Нарзан 0020-11)
Цена: 940.0 руб.
Кол-во: 13.0 шт.

At the bottom right of the callout box is a red "OK" button.

Руководитель практики подсказал очень важную функцию, после того, как я показал ему еще раз программу: возможность использовать сканер в портретном режиме. По умолчанию доступен только ландшафтный режим.

Тогда я стал искать решение, конечно же, в интернете. И нашел! Правда, пришлось искать довольно долго: решение было предоставлено по сути кусками, и пришлось их складывать друг с другом, чтобы получить результат:



Портретный режим работает!

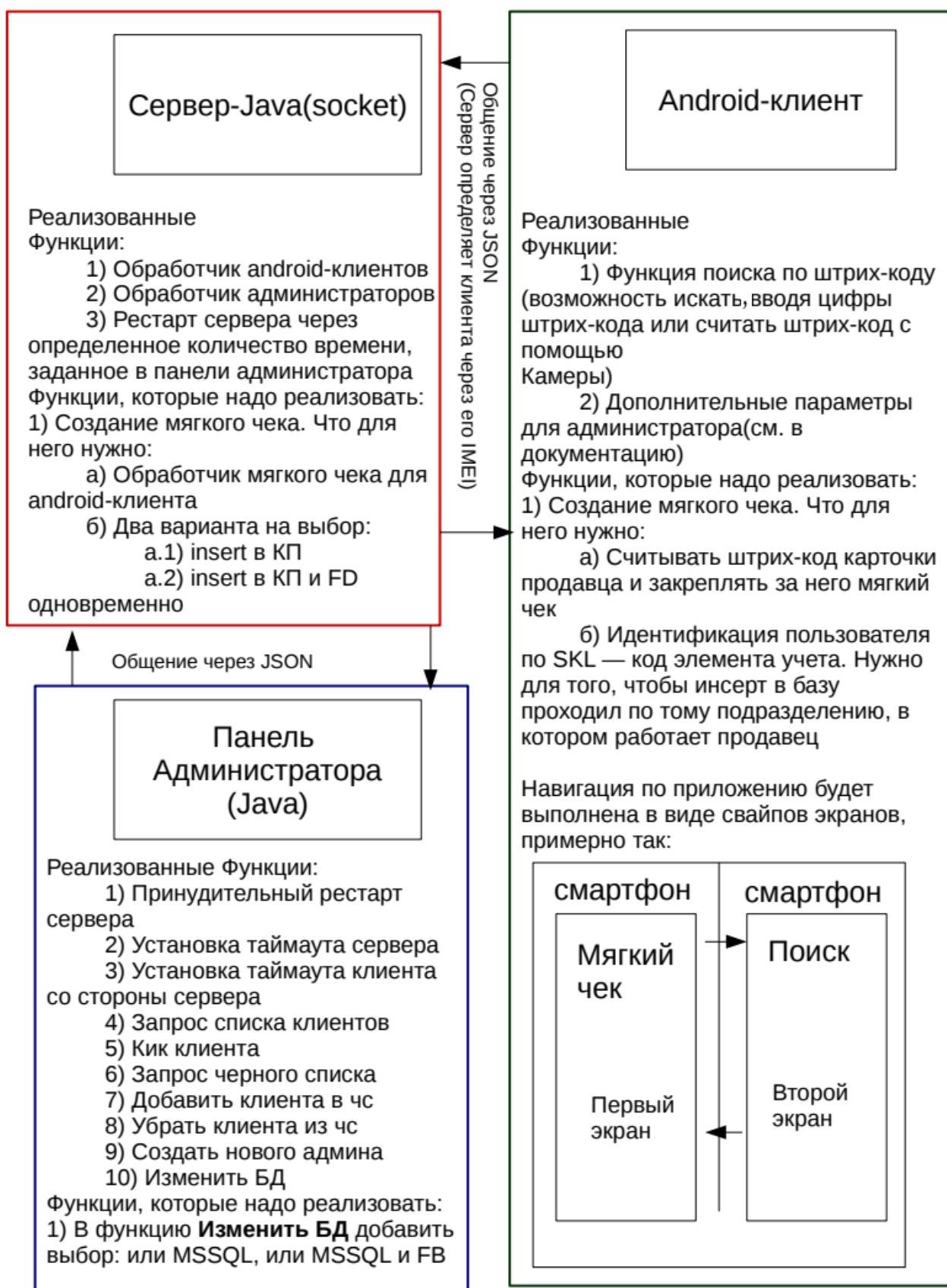
Руководитель практики скачал себе на смартфон мое приложение для тестов. И он сделал еще одно очень важное замечание: при подключении, если не включен VPN, не включен сервер, или если плохая связь, то программа бесконечно висит при подключении: бесконечное окошко с надписью «Подключение» и бегущими синими шариками... Тогда он мне посоветовал сделать так: в цикле несколько раз пытаться подключиться к серверу, например 3 раза по 10 секунд. Если все попытки исчерпаны, то высвечивать окошко с информацией, что подключение не установлено.

Когда я сел писать эту функцию, то у меня выключили свет, и дали его примерно через несколько часов. Я включаю свой ПК, открываю Android Studio, пишу функцию, жму на зеленый треугольник — и Android Studio не собирает приложение. Какая-то ошибка во внутреннем JSON-файле самой студии! Я не нашел ни одного внятного решения, кроме как одного: удалить проект и все с начала написать. Конечно ситуация из неприятных, но выбора не было:

пришлось копировать код из старого проекта в новый. После примерно получаса у меня уже был «старый-новый» проект, и он заработал.

После этого я уже не так активно занимался этим проектом — практика уже давно подошла к концу, было начало августа, и я только исправлял мелкие недочеты, когда они всплывали. Но зато потом, мой руководитель практики, который сейчас является уже руководителем проекта для меня, показал данное приложение генеральному директору компании «Баярд» — ему понравилось данное приложение, и он хотел бы, чтобы я и дальше развивал его! В приложении нужна еще одна функция, помимо поиска — создание мягкого чека. Это задание довольно тяжелое, но не менее интересное. Для его выполнения необходимо вынести всю бизнес-логику за сервер, чтобы весь комплекс приложений LightSearch был независим от какого-либо предприятия. Всю часть, связанную с сервером, я описал в соответствующем документе ([ссылка](#)). Здесь же будет вся информация, связанная с Android-клиентом.

Схема, которую я нарисовал для общего представления о проекте:



Как видно, схема не предполагала принципа «команда-ответ». Теперь insert в базу сервер делать не будет, а будет передавать команде базе, и сама база будет делать insert, и затем давать ответ серверу. Также будет произведен перенос select`ов из сервера в бд.

Предположительный интерфейс программы (на следующей странице):

АВТОРИЗАЦИЯ

Имя пользователя:

Пароль:

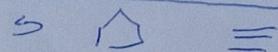
Расширенные настройки

Хост сервера:

Порт сервера:

[Изменить пароль администратора]

[Подключиться]



Мягкий чек

Поиск:

[СЧИТАТЬ ШТРИХ-КОД]

ШТРИХ-КОД (коинкарование)	
~~~~~	<input type="radio"/> 15 (⊕)
~~~~~	<input type="radio"/> 1 (⊕)
~~~~~	<input type="radio"/> 9 (⊕)

[ЗАКРЫТЬ МЯГКИЙ ЧЕК]



## Поиск товара

Поиск:

[СКАНИРОВАТЬ ШТРИХ-КОД]

- Склады
- ТК
- Все

Выберите склад:  
<Все>

Выберите ТК:  
<Все>

[Поиск]

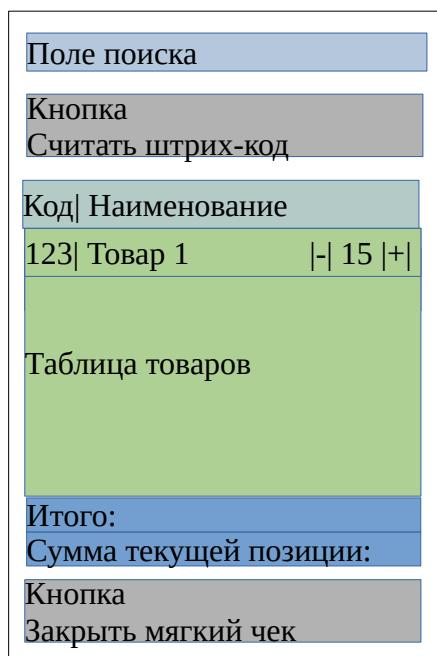


При подключении происходит переход из окна авторизации в окна «Мягкий чек» и «Поиск товара», переход между которыми осуществляется через свайп экрана вправо или влево. По умолчанию экран «Мягкий чек» является первым.

Сценарии мягкого чека:

1) Пользователь нажимает на кнопку «Создать мягкий чек», приложение открывает экран формирования мягкого чека. Пользователь в поле ввода штрих-кода вводит штрих-код товара, или считывает его при помощи камеры смартфона, нажав на кнопку «Считать штрих-код». В таблице товаров добавляется новая запись с соответствующим товаром. В строке отображается штрих-код и наименование товара. Справа от этой информации расположены кнопки «-» и «+», которыми можно регулировать количество товара. Между этими кнопками отображается количество товара. Можно написать количество вручную, нажав на количество товара. При нажатии на строку в поле «Сумма текущей позиции» отображается сумма выбранной в таблице позиции. В поле «Итого» отображается сумма всех позиций в таблице. Пользователь нажимает кнопку «Закрыть мягкий чек», чек передается на сервер, сервер передает чек в базу данных, база данных передает ответ серверу, сервер передает ответ приложению, приложение отображает на экране полученный ответ.

Схема интерфейса сценария 1:

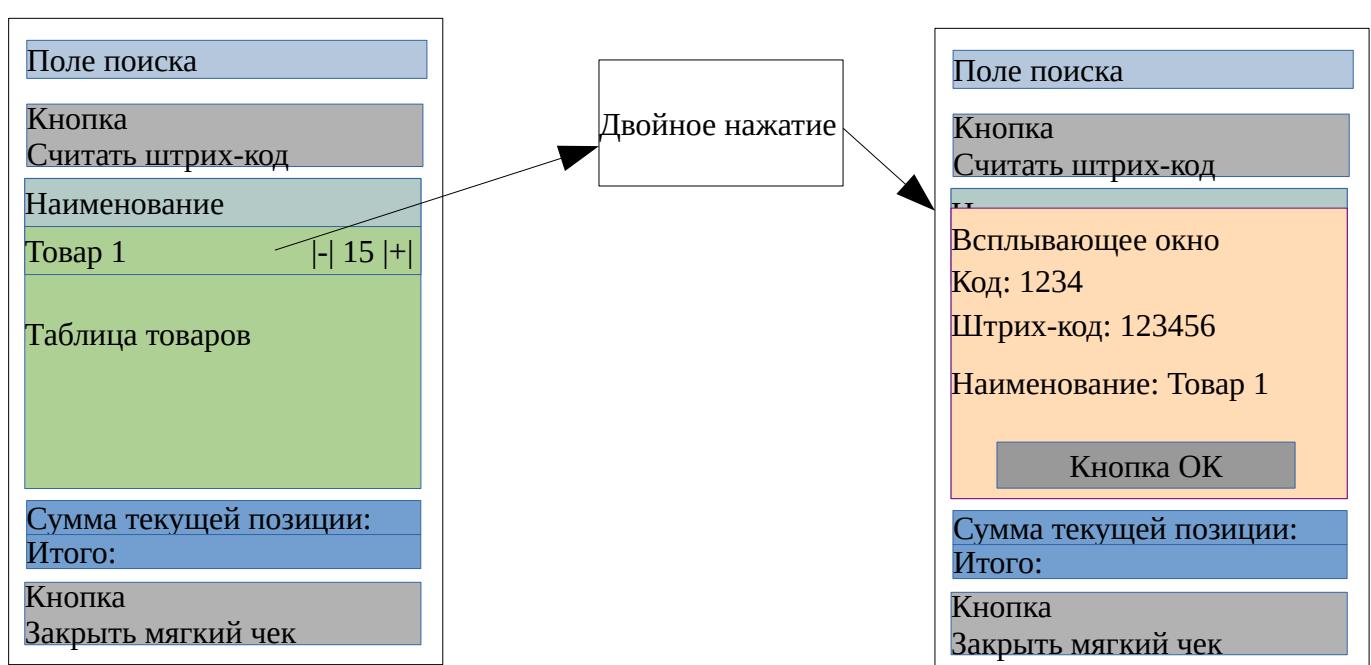


Минус данного подхода: так как экран смартфона ограничен, то поля «Код» и «Наименование» не будут полностью отображаться на экране. Переносить на новую строку символы, которые не вместились на экран, нельзя, так как информация в таблице станет неудобной для восприятия.

2) Пользователь нажимает на кнопку «Создать мягкий чек», приложение открывает экран формирования мягкого чека. Пользователь в поле ввода штрих-кода вводит штрих-код товара, или считывает его при помощи камеры

смартфона, нажав на кнопку «Считать штрих-код». В таблице товаров добавляется новая запись с соответствующим товаром. В строке отображается наименование товара. Справа от этой информации расположены кнопки «-» и «+», которыми можно регулировать количество товара. Между этими кнопками отображается количество товара. Можно написать количество вручную, нажав на количество товара. При нажатии на строку в поле «Сумма текущей позиции» отображается сумма выбранной в таблице позиции. При двойном нажатии на строку в таблице открывается всплывающее окно, в котором написаны полностью наименование, код, и штрих-код товара. В поле «Итого» отображается сумма всех позиций в таблице. Пользователь нажимает кнопку «Закрыть мягкий чек», чек передается на сервер, сервер передает чек в базу данных, база данных передает ответ серверу, сервер передает ответ приложению, приложение отображает на экране полученный ответ.

Схема интерфейса сценария 2:

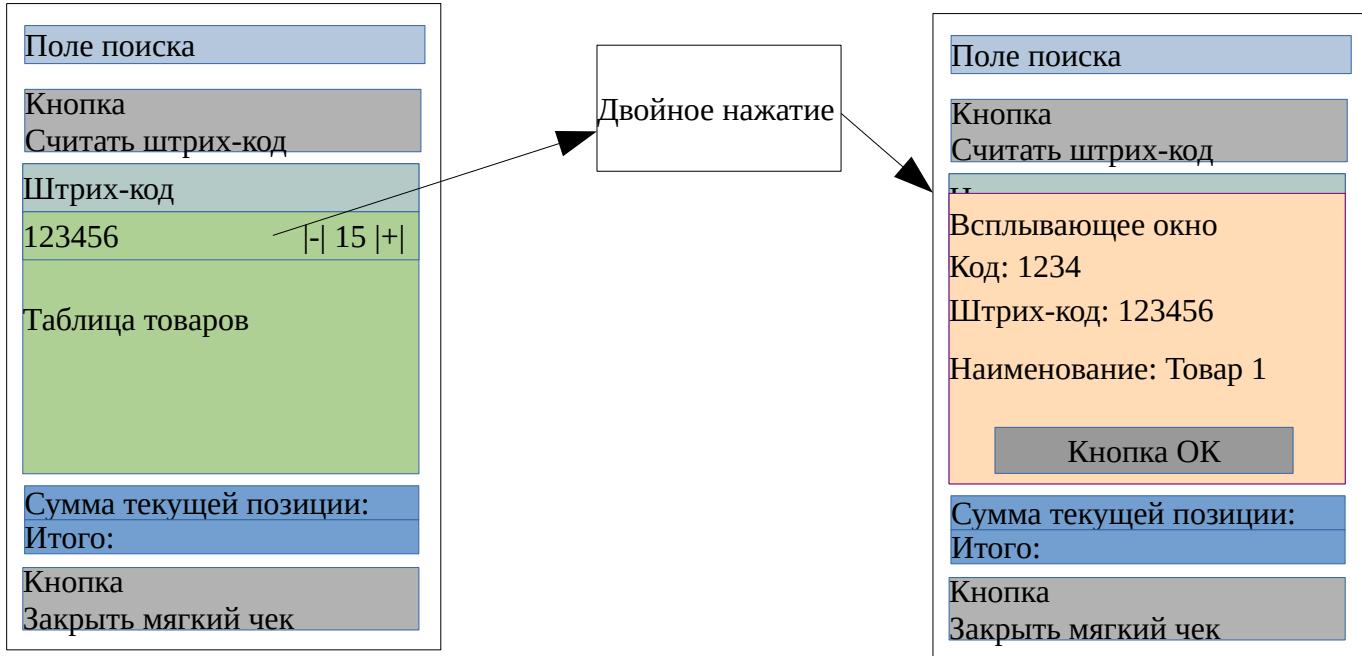


При таком подходе в строке таблицы отображается больше информации, а в всплывающем окне будет отображена полная информация о товаре. Также в всплывающем окне допустимо использовать перенос строки — это не будет неудобно, как в первом подходе.

3) Пользователь нажимает на кнопку «Создать мягкий чек», приложение открывает экран формирования мягкого чека. Пользователь в поле ввода штрих-кода вводит штрих-код товара, или считывает его при помощи камеры смартфона, нажав на кнопку «Считать штрих-код». В таблице товаров добавляется новая запись с соответствующим товаром. В строке отображается штрих-код товара. Справа от этой информации расположены кнопки «-» и «+», которыми можно регулировать количество товара. Между этими кнопками отображается количество товара. Можно написать количество вручную, нажав

на количество товара. При нажатии на строку в поле «Сумма текущей позиции» отображается сумма выбранной в таблице позиции. При двойном нажатии на строку в таблице открывается всплывающее окно, в котором написаны полностью наименование, код, и штрих-код товара. В поле «Итого» отображается сумма всех позиций в таблице. Пользователь нажимает кнопку «Закрыть мягкий чек», чек передается на сервер, сервер передает чек в базу данных, база данных передает ответ серверу, сервер передает ответ приложению, приложение отображает на экране полученный ответ.

Схема интерфейса сценария 2:

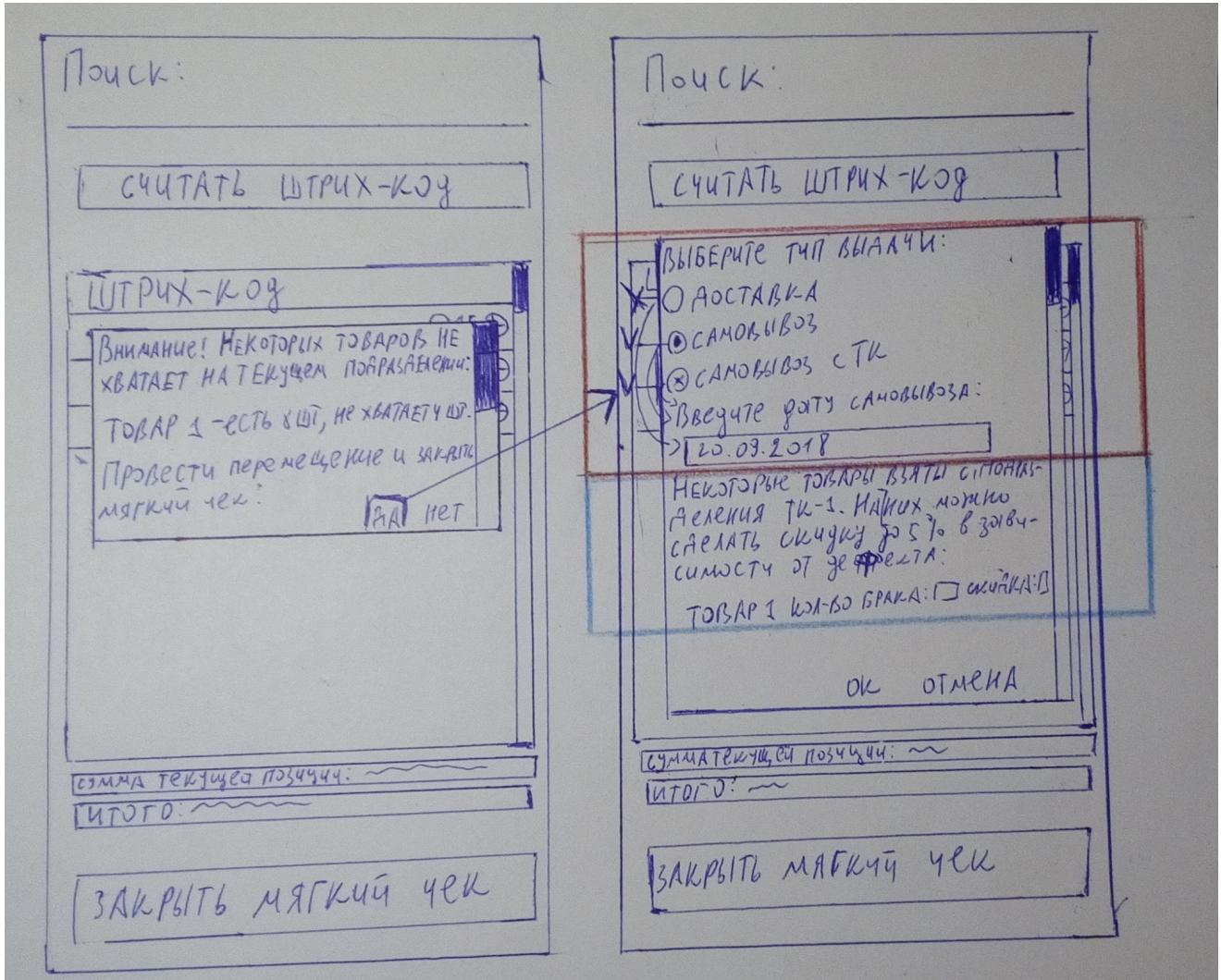


Замечание, касаемо всех подходов: можно менять количество товаров четырьмя способами:

- 1) Нажатием кнопок «-» и «+»;
- 2) Нажатием на количество товара. В этом случае откроется цифровая клавиатура;
- 3) Считать товар при помощи сканера, который уже добавлен в таблицу;
- 4) Написать штрих-код товара, который уже добавлен в таблицу.

[19.09.2018]

Сегодня пришло техническое задание. После его переработки я нарисовал, как будет выглядеть всплывающие окна после нажатия на кнопку «Закрыть мягкий чек»:



Из-за того, что во всплывающем окне будет довольно много элементов, необходимо добавить вертикальную прокрутку. Первое всплывающее окно появляется только тогда, когда товара не хватает в текущем подразделении. Тогда необходимо провести перемещение. Перемещение полностью автоматизировано на предприятии, поэтому клиент просто будет посыпать команду серверу, что необходимо провести перемещение.

При нажатии на кнопку «Да» в первом окне открывается еще одно всплывающее окно, в котором необходимо указать тип выдачи. Если тип выдачи — самовывоз и самовывоз с ТК, то необходимо указать дату самовывоза. Если перемещение необходимо, то может возникнуть ситуация, когда товар берется с какого-то склада, но он с дефектами, и поэтому на него можно сделать скидку. В этом случае необходимо указать количество дефектного товара данной позиции (на картинке я написал кол-во брака только для того, чтобы все слова

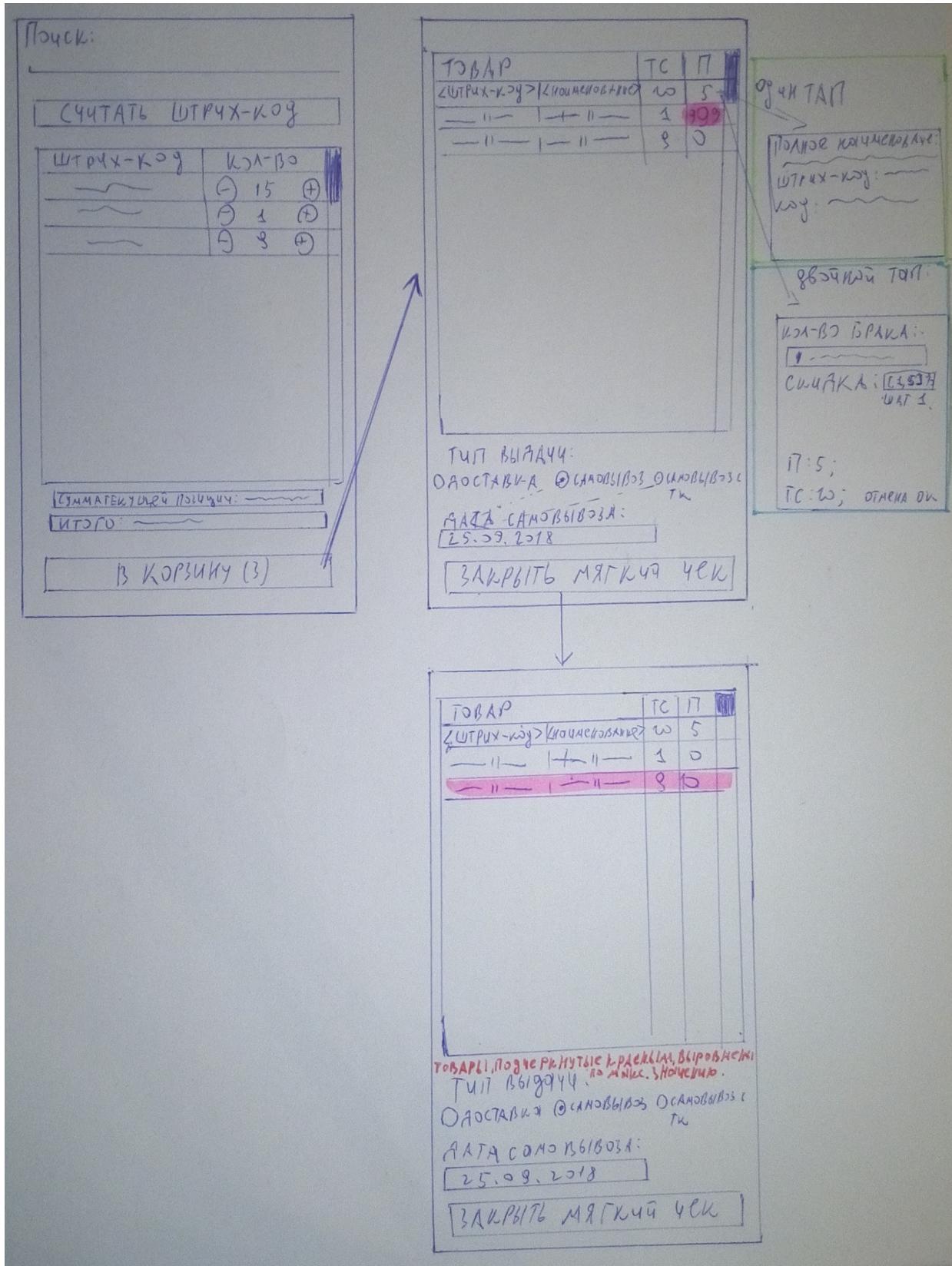
вместились в одну строчку :). После нажатия на кнопку «OK» мягкий чек закрывается.

Если перемещение не потребуется, то будет выведено только то, что выделено на картинке красным, и кнопки «OK» и «Отмена».

Если перемещение требуется, то будет выведено то, что выделено и красным, и синим, и кнопки «OK» и «Отмена».

[24.09.2018]

Техническое задание переделывается, как и интерфейс программы. Отображение предупреждения перемещения, выбор вида выдачи было решено не отображать во всплывающем окне, а сделать для этого отдельное окно. Пока что в моем представлении это будет выглядеть как-то так:



Теперь кнопка «Закрыть мягкий чек» перенесена на экран «Заказ», а на экране мягкого чека добавлена кнопка «В корзину», в скобках которой отображается количество товаров в списке. При нажатии на нее открывается окно заказа, в котором также перечислены товары, но уже с учетом перемещения. ТС — количество товара из текущего склада, П — количество товара из перемещения. При одном нажатии на строчку открывается полное наименование, штрих-код, и код товара. При двойном нажатии открывается окошко с полями: количество брака и процент скидки — от 1 до 5%. Брак возможен только с текущего склада, но для удобства в это всплывающее окошко также будет добавлена информация о количестве данного товара в текущем складе и в перемещении.

Насчет двойного нажатия я еще не уверен, потому что есть вариант длительного удержания по строчке. Будем тестировать, когда придет время :)

Также я рассмотрел ситуацию, когда в полях ТС и П указывают значения больше, чем есть на самом деле. Как видно на экране справа, продавец ввел 999 товара второй строки, и нажал кнопку «закрыть мягкий чек». После ожидания, ему подсветится красным цветом те строки, в которых было превышено значения количества либо по ТС, либо по П, и сделает выравнивание по обоим полям по максимальному значению. Насчет сообщения, написанное под таблицей красным мелким шрифтом, я еще не уверен, нужен ли он вообще. Скорее всего, опишу в документации, что значит подсветка красным цветом, и на этом закончим. В общем, все еще впереди :)

## [12.10.2018]

Неделю назад пришел смартфон для тестов, который заказывала компания — Huawei Y5 prime 2018. Установив на него LightSearch, меня ждала неприятная картина — смартфон «вешал» сервер. На сервере завершалась работа с исключением `java.lang.Thread.run<unknown source>`. Не найдя внятного ответа на вопрос, по какой причине может возникать данная ошибка, я решил обновить Android Studio, сделать Invalidate Caches/Restart, очистить и собрать проект, создать APK и установить ее на смартфон. После данной процедуры эта ошибка больше не появлялась. Но неприятности на этом не закончились.

После того, как я считал штрих-код, выбрал подразделение и нажал кнопку «Поиск», мне открылась таблица с товарами. Когда я выхожу из нее, то смартфон виснет на секунды 3-4, затем два раза выдает сообщение: Успех! Подключение установлено! Это неадекватное поведение мне сразу не понравилось. Мало того, что он два раза переподключается к серверу, так еще и запоминает состояние активити «Поиска» очень странно: радио кнопка стоит на «Подразделение: Все», а при этом я могу выбрать склады из выпадающего списка! И это еще не всё. Заметил также, что смартфон иногда переподключается к серверу один раз тогда, когда он находится в режиме сканирования штрих-кода.

Посмотрев на логи сервера, я увидел, как этот телефон подключается два раза к серверу каждый раз, когда я выхожу из таблицы. Также я ставил сервер на отладку, пытаясь снова поймать ошибку `java.lang.Thread.run<unknown source>`, но этого больше не происходило после пересборки APK.

Также я ставил отладку и на Android-приложение — и увидел, как оно, после закрытия окна с таблицей, перескакивает на строчку с созданием активити, причем два раза!

Я конечно сделал заглушку, объявив в классе окна авторизации статическую переменную, которая проверяет, сколько раз было установлено подключение, и в зависимости от этого не выводит сообщения успеха подключения при закрытии таблицы. Но это все «костыли»: сервер не обманешь.

Успев расстроиться от того, что смартфон не подходит (потому что LightSearch прекрасно работал на всех устройствах, на которых я его тестировал — Huawei Honor 3C, meizu m5C, Prestigio PSP350(!!!)). Скорее всего, смартфону не хватает мощности, подумал я — на нем стоит Android 8.1, а оперативной памяти — 2 гб, как на моем Honor 3C — так я сначала подумал. И решил, что возьму данный смартфон на постоянную основу для проверки его работоспособности. И вот к каким выводам я пришел: да, он тормозит. Но не настолько критично, чтобы какие-либо приложения вылетали. Единственно странно ведет себя YouTube — при попытке загрузить видео в 720p 60 fps, оно тормозит секунд 5, потом нормально воспроизводится. Но остальные приложения, такие как Chrome, Telegram, Twitter, Instagram, VK работают отлично. Даже такое приложение, как Asos, которое нещадно греет Huawei Honor 3C и потребляет очень много энергии — не тормозит, не «кушает» аккумулятор, и не греет телефон! И я понял, что что-то не так с моим приложением.

И оказывается, да, так оно и есть. Прочитав в интернете, я узнал, что использовать много Activity в одном приложении — очень неправильно, так как любое Activity, уходя в паузу, становится «котом Шредингера» - оно либо после возобновления продолжит работу на том месте, на котором закончило, либо перезагрузится. К тому же Activity копятся в памяти устройства и делает его работу более медленной и нестабильной. Поэтому для решения данных проблем были придуманы фрагменты (Fragment).

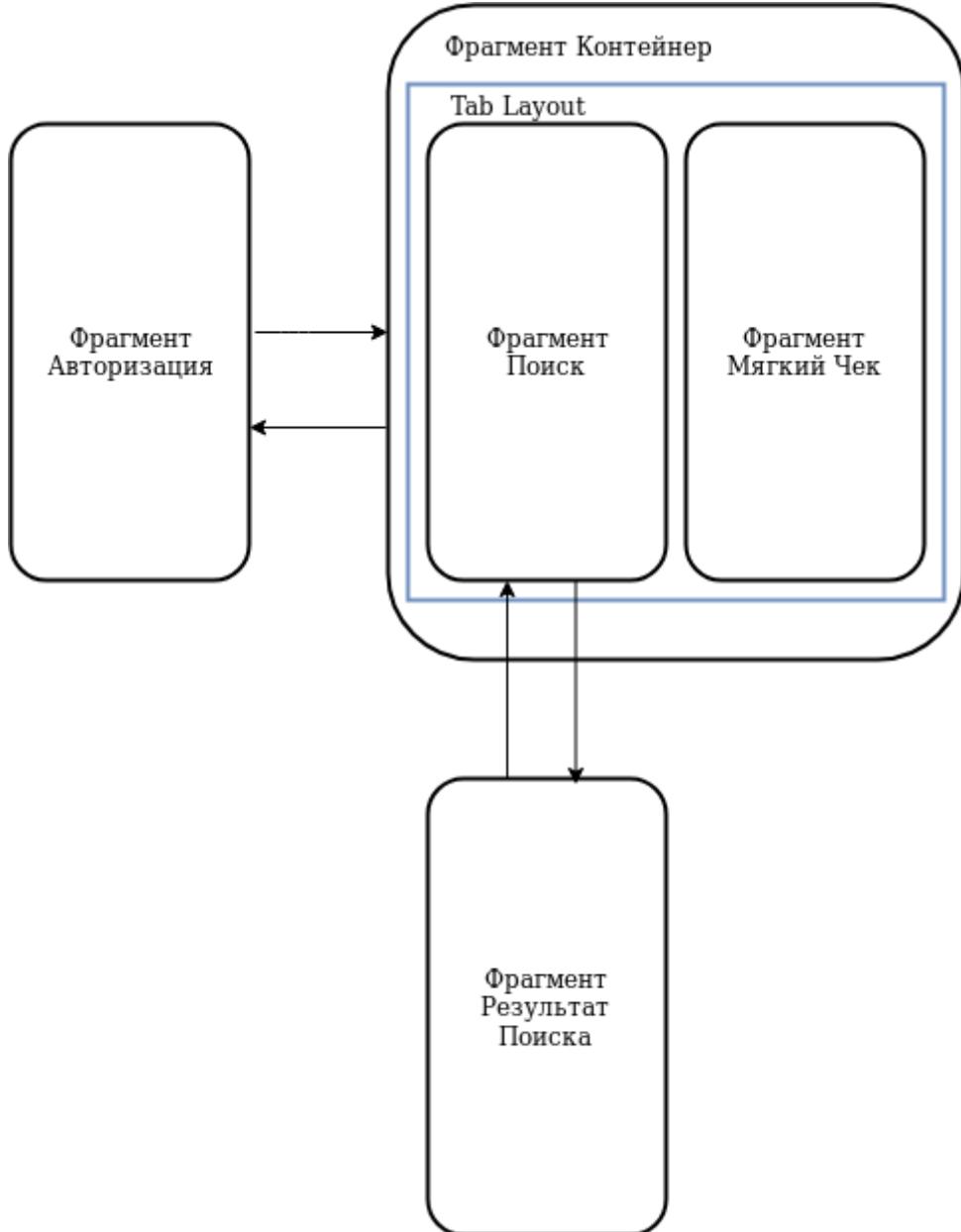
Фрагменты хороши тем, что их можно загружать в одно Activity, тем самым решая проблему заполнения памяти устройства. И самое важное это то, что основные действия с фрагментами (add, del, replace) происходят в транзакции, что гарантирует выполнения этих действий тогда, когда это необходимо и тогда, когда это ожидается.

Из всего вышеперечисленного я сделал вывод, что приложение необходимо перевести на фрагменты. Поэтому сейчас я занимаюсь их изучением.

[09.11.2018]

Итак, около двух недель назад я перевел приложение на фрагменты. Теперь оно корректно работает на смартфоне Huawei Y5 Prime 2018. Проблем с переводом было много, поэтому я начну.

Первое, что я сделал — схему структуры приложения. Вот такая она у меня получилась:



В приложении работает одно Activity, называемое ManagerActivity. Сначала в него загружается фрагмент авторизации — AuthorizationFragment. Затем, если в AuthorizationFragment были корректно введены имя пользователя и пароль, то при нажатии на кнопку «Подключиться» идёт замена фрагмента авторизации на фрагмент контейнер — ContainerFragment. В свою очередь, ContainerFragment содержит в себе два фрагмента — SearchFragment и SoftCheckFragment — фрагмент поиска и фрагмент мягкого чека соответственно. SearchFragment вызывает фрагмент результата поиска, если результат поиска содержит в себе

более двух записей. Тогда ManagerActivity меняет ContainerFragment на ResultSearchFragment.

Смена фрагментов происходит в транзакции и выглядит в общем виде следующим образом:

```
1 SomeFragment someFragment = new SomeFragment();
2 FragmentTransaction transaction = this.getSupportFragmentManager().beginTransaction();
3 transaction.replace(R.id.FrameLayoutSome, someFragment, getString(R.string.fragment_some));
4 transaction.addToBackStack(getString(R.string.fragment_some));
5 transaction.commit();
```

Строка `transaction.addToBackStack(getString(R.string.fragment_some))` добавляет эту транзакцию в back stack. Это значит, что транзакция будет запомнена после вызова метода `commit`. В фрагмент авторизации возвращаться обратно не нужно, поэтому метод для вызова `AuthorizationFragment` будет выглядеть так:

```
1 AuthorizationFragment authorizationFragment = new AuthorizationFragment();
2 FragmentTransaction transaction = this.getSupportFragmentManager().beginTransaction();
3 transaction.replace(R.id.FrameLayoutManager, authorizationFragment,
4         getString(R.string.fragment_authorization));
5 transaction.commit();
6 setTitle(getString(R.string.fragment_authorization));
```

Для проведения транзакции необходимо создать объект типа `FragmentTransaction`. Далее в методе `replace` заменяется контейнер `R.id.FrameLayoutManager` на `authorizationFragment`. В методе `setTitle` устанавливаем заголовок приложения — переменная `R.string.fragment_authorization` содержит значение «Авторизация».

Вызов `ContainerFragment` делается так:

```
1 ContainerFragment searchFragment = new ContainerFragment();
2 FragmentTransaction transaction = this.getSupportFragmentManager().beginTransaction();
3 transaction.setCustomAnimations(R.anim.enter_from_right,
4         R.anim.exit_to_left, R.anim.enter_from_left, R.anim.exit_to_right);
5 transaction.replace(R.id.FrameLayoutManager, searchFragment,
6         getString(R.string.fragment_container));
7 transaction.addToBackStack(getString(R.string.fragment_container));
8 transaction.commit();
9 setTitle(getString(R.string.fragment_container));
10 StackFragmentTitle.push(getString(R.string.fragment_authorization));
```

Данную транзакцию необходимо добавить в back stack, поэтому вызываем функцию `addToBackStack`.

Строка `transaction.setCustomAnimations(R.anim.enter_from_right, R.anim.exit_to_left, R.anim.enter_from_left, R.anim.exit_to_right)` устанавливает анимацию перехода между фрагментами. В данном случае из названий можно понять, что для предыдущего фрагмента (т. е. `AuthorizationFragment`) устанавливается анимация сдвига окна влево, а для

фрагмента, который заменить AuthorizationFragment(т.е. ContainerFragment) — анимация сдвига окна справа налево. Вот как реализована анимация сдвига влево при выходе(R.anim.exit_to_left):

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <set xmlns:android="http://schemas.android.com/apk/res/android"
3     android:shareInterpolator="false">
4
5     <translate
6         android:fromXDelta="0%" android:toXDelta="-100%"
7         android:fromYDelta="0%" android:toYDelta="0%"
8         android:duration="600"/>
9 </set>
```

Итак, первая проблема — как поместить в фрагмент другие фрагменты? Это оказалось не сложно — это делается также, как если бы вместо ContainerFragment был ActivityManager. Конечно же для того, чтобы реализовать два фрагмента в одном окне, и их можно было менять свайпом влево или вправо, необходимо в разметке FragmentContainer поместить tabLayout.

Вторая проблема — как общаться фрагментом между собой, и как общаться активити между фрагментами, и наоборот? В интернете существует множество решений данной проблемы, но как по мне, лучшая из них приводится в официальной документации Android, а именно — использовать интерфейсы. Также, из документации я узнал, что общение фрагментов между собой происходит через активити.

Итак, для осуществления общения вида активити-фрагмент я написал следующий интерфейс:

```
1 public interface IManagerActivity {
2     void setAuthorizationData(String username, String password, String host, String port);
3     void setSearchData(String barcode, String podrazdelenie, String TK, String sklad);
4     void initSearchFragment();
5     void initResultSearchFragment(ResultSearchFragment resultSearchFragment);
6     void closeConnection();
7 }
```

Так, например, если мне необходимо получить имя пользователя и пароль от AuthorizationFragment, я делаю следующее:

- 1) Определяю в AuthorizationFragment поле: IManagerActivity mIManagerActivity;
- 2) Далее в методе onAttach инициализируем данный объект;
- 3) И теперь, когда необходимо передать данные в ManagerActivity, пишем такую строчку:

```
1 mIManagerActivity.setAuthorizationData(editTextUsername.getText().toString(),
2         editTextPassword.getText().toString(),
3         editTextHost.getText().toString(),
4         editTextPort.getText().toString());
```

ManagerActivity должен реализовывать интерфейс IManagerActivity. Для этого при объявлении данного класса необходимо написать так:

```
1 public class ManagerActivity extends AppCompatActivity implements IManagerActivity
```

И переопределить метод `setAuthorizationData`. И вуаля, все работает!

Далее, с чем необходимо было решить проблему — общение вида фрагмент-фрагмент. Для него тоже необходимо реализовать интерфейсы. Для начала надо выяснить, с кем нам надо общаться.

Налаживать отношения будем между SearchFragment и ContainerFragment. Напрямую общаться с ManagerActivity мы можем только в случае передачи ему информации для поиска — это штрих-код и подразделение. Поэтому здесь ситуация та же, что и с AuthorizationActivity, только теперь в интерфейсе IManagerActivity реализуем метод `setSearchData`. А вот например при вызове окна ZXing для считывания штрих-кода и помещения его в строку штрих-кода фрагмента поиска напрямую мы не можем, потому что для получения штрих-кода товара необходимо в ManagerActivity переопределить метод `onActivityResult`, и если бы мы сразу передали результат в SearchFragment, то программа бы выдала ошибку — т. к. данного объекта не существует.

SearchFragment существует внутри ContainerFragment, поэтому сначала надо передать информацию, полученную со сканера ContainerFragment`у, а затем ContainerFragment передает эту информацию SearchFragment`у, который существует внутри данного ContainerFragment`а. Итого — пишем интерфейс, который будет реализовывать SearchFragment:

```
1 public interface ISearchFragment {
2     void setSearchBarcodes(String barcode);
3 }
```

Теперь пишем интерфейс, который будет реализовывать ContainerFragment:

```
1 public interface.IContainerFragment {
2     void initSearchSpinners(String[] skladArray, String[] TKArray);
3     void setSearchBarcodes(String barcode);
4 }
```

Как это использовать, и как это будет работать:

- 1) В SearchFragment при нажатии на кнопку «Считать штрих-код» будет вызываться сканер штрих-кода:
- 2) В ManagerActivity будет происходить его обработка при помощи переопределения метода onActivityResult. В этом методе, после того, как получили номер штрих-кода, перебираем все фрагменты на поиск тех, кто реализует интерфейс IContainerFragment:

```
1  @Override
2  public void onActivityResult(int requestCode, int resultCode, Intent intent) {
3      IntentResult scanningResult = IntentIntegrator.parseActivityResult(requestCode,
4          resultCode, intent);
5      if(scanningResult != null) {
6          String scanContent = scanningResult.getContents();
7          FragmentManager fragmentManager = this.getSupportFragmentManager();
8          IContainerFragment containerFragment = null;
9          for(Fragment fragment : fragmentManager.getFragments()) {
10              if(fragment instanceof IContainerFragment) {
11                  containerFragment = (IContainerFragment) fragment;
12                  break;
13              }
14          }
15
16          if(containerFragment != null)
17              containerFragment.setSearchBarcodes(scanContent);
18      }
19  }
```

Так как IContainerFragment реализует только один класс — FragmentContainer, то именно он и только он будет найден. Далее вызываем метод setSearchBarcodes;

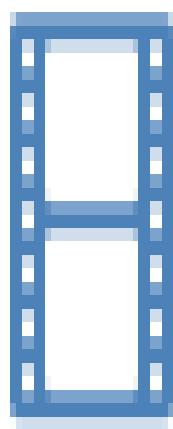
- 3) В ContainerFragment в переопределенном методе setSearchBarcodes ищем фрагмент, который реализует интерфейс ISearchFragment, то есть ищем фрагмент SearchFragment. Перебор аналогичен с перебором в ManagerActivity, за исключением, что вместо метода getSupportFragmentManager надо вызывать getChildFragmentManager. После того, как нашли, вызываем метод setSearchBarcodes интерфеса ISearchFragment;
- 4) В SearchFragment в переопределенном методе setSearchBarcodes в поле ввода штрих-кода вписываем пришедшие циферки. Все работает!

После этого надо было решить вопрос с переопределением кнопки back для фрагментов. Если с активити все просто (для этого необходимо только переопределить метод onBackPressed), то в случае фрагментов — не все так просто. Я пытался делать разными способами, но все они работали криво. И самый лучший и рабочий способ как всегда оказался... через интерфейс :). Для переопределения метода onBackPressed я создал интерфейс OnBackPressedListener, в котором определил один единственный метод, который ничего не возвращает — onBackPressed. Данный интерфейс должны реализовывать все фрагменты, у которых необходимо поменять обработку

кнопки назад. Тогда, в ManagerActivity мне остается только переопределить метод onBackPressed, в котором перебираются фрагменты, реализующие onBackPressed: если реализует, вызываем соответствующий метод, иначе — вызываем обработку по умолчанию:

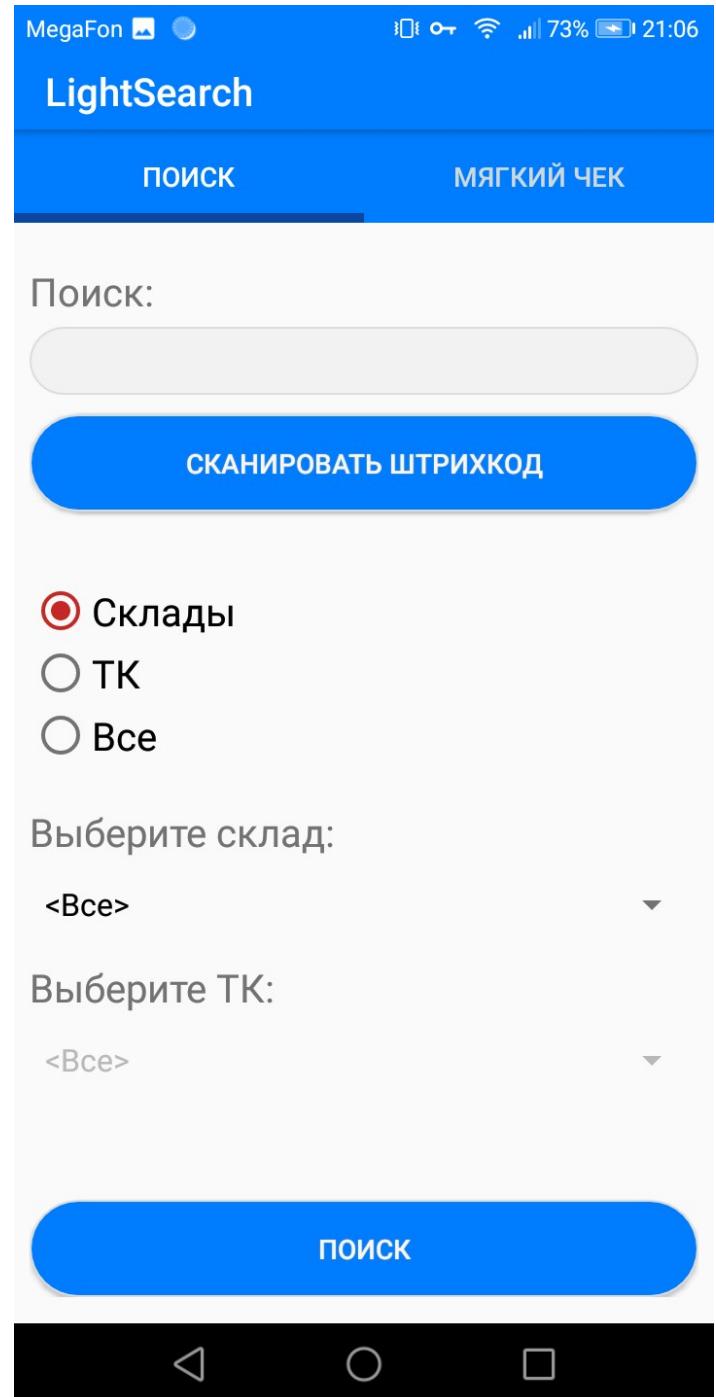
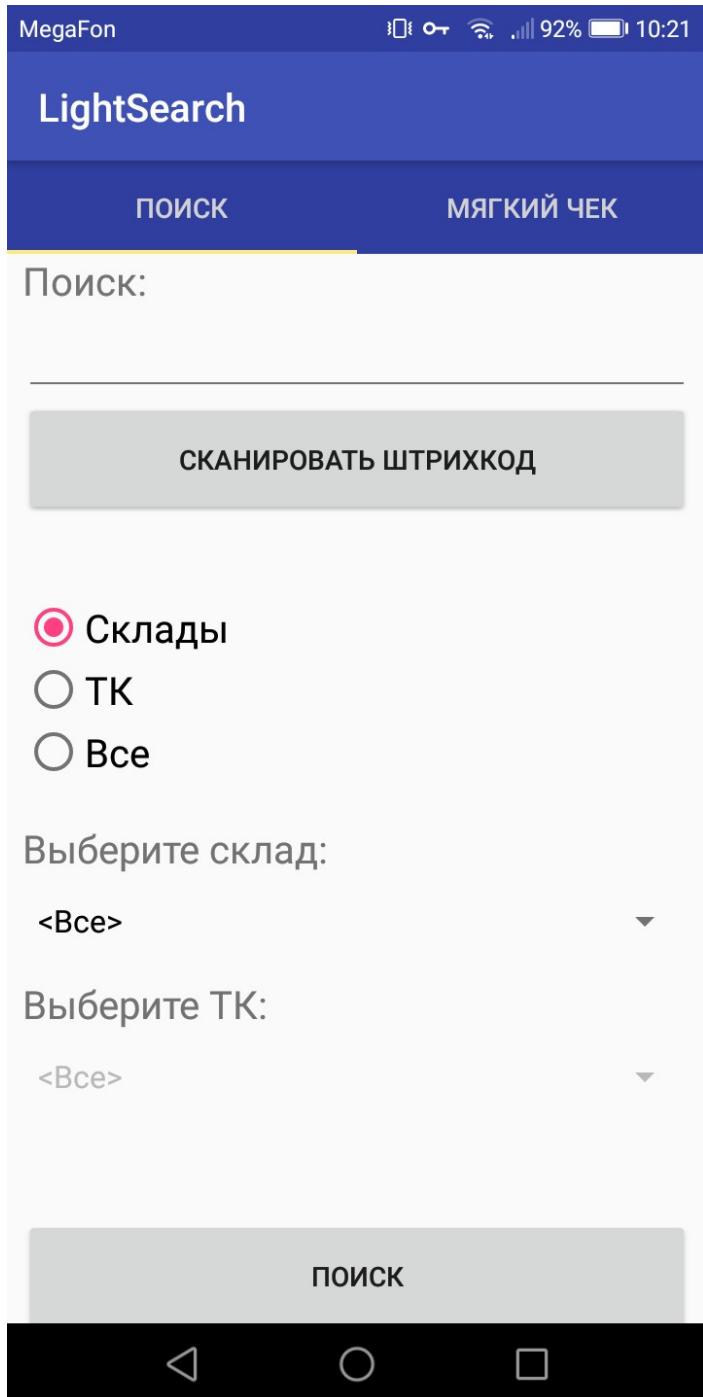
```
1 @Override
2 public void onBackPressed() {
3     FragmentManager fragmentManager = this.getSupportFragmentManager();
4     OnBackPressedListener backPressedListener = null;
5     for(Fragment fragment : fragmentManager.getFragments()) {
6         if(fragment instanceof OnBackPressedListener) {
7             backPressedListener = (OnBackPressedListener) fragment;
8             break;
9         }
10    }
11
12    if(backPressedListener != null)
13        backPressedListener.onBackPressed();
14    else
15        super.onBackPressed();
16 }
```

Еще несколько слов про анимацию. Я также сделал анимацию перехода между ContainerFragment и ResultSearchFragment. На ее работу можно взглянуть на [этом видео](#):

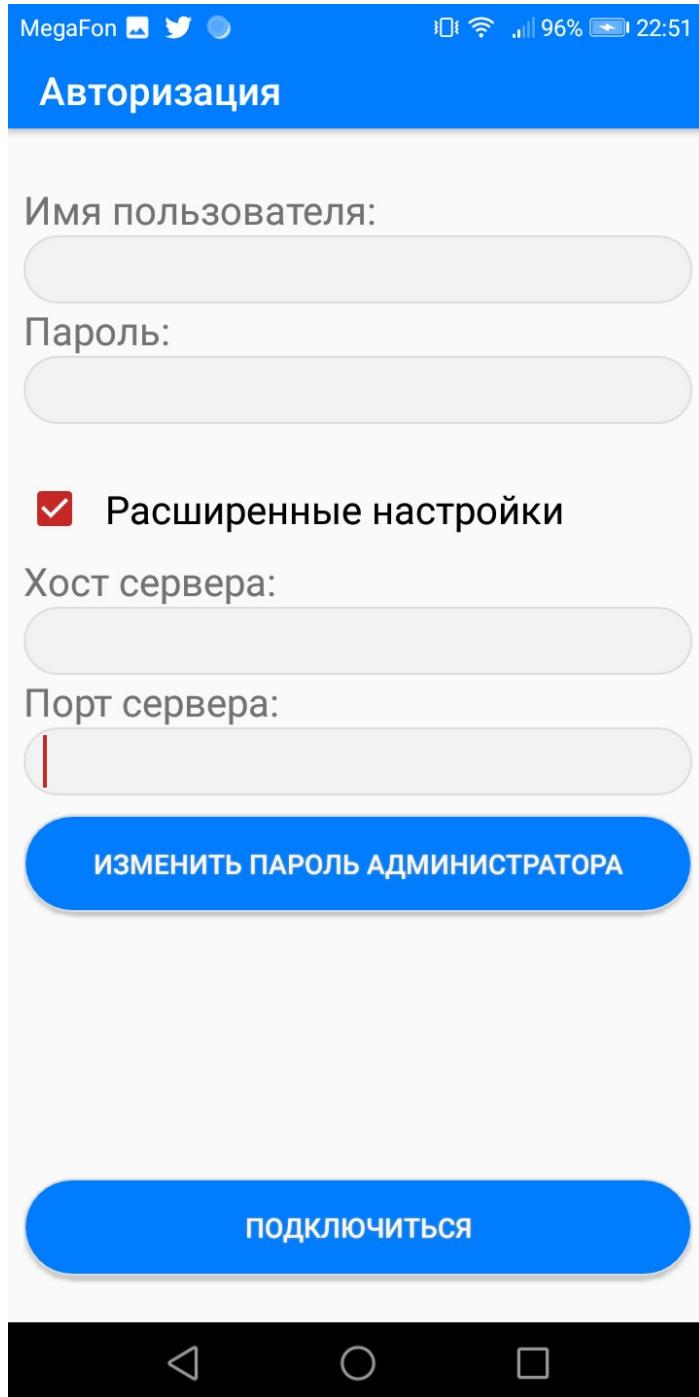


Также было решено избавиться от таблицы: теперь ее заменил список. Так намного лучше и удобнее. ResultSearchFragment наследуется от класса ListFragment.

Также я изменил дизайн программы. Я сделал кнопки с закругленными краями, изменил цвет самого приложения: теперь он соответствует цвету из таблицы MaterialDesign. Изменил также дизайн окон ввода. Теперь приложение стало более приятным на глаз, и им стало пользоваться намного приятнее. Вот для сравнения два изображения: на левом — старый дизайн, на правом — новый.



Я довольно долго подбирал цвета для приложения, осталось много скриншотов с неудачными вариантами :). Я лучше покажу скриншот фрагмента авторизации: фрагмент мягкого чека не покажу, потому что там пока что только пустой экран:



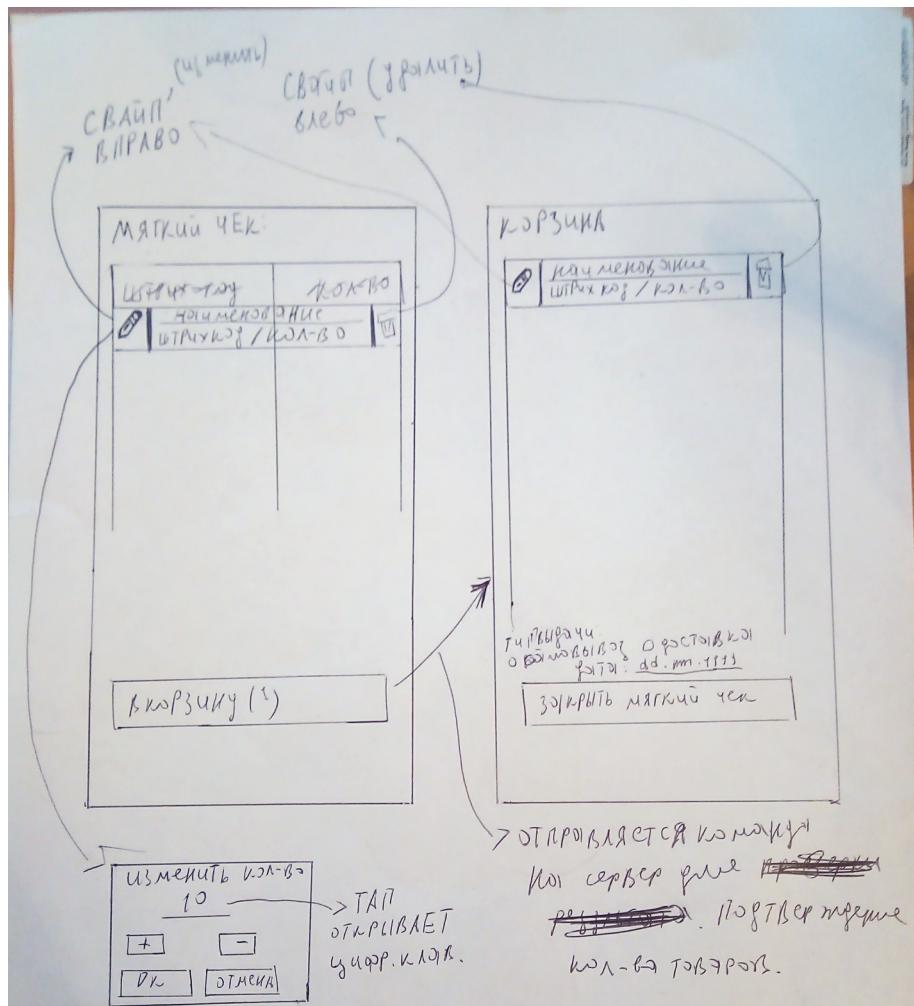
Потом я хотел добавить SplashScreen, но почитав комментарии умных людей на форумах, я пришел к выводу, что моему приложению SplashScreen не нужен. Итак, приложение успешно переведено на фрагменты, но я пришел к выводу, что в дальнейшем надо будет переписать приложение, используя паттерны и некоторые библиотеки, иначе код в дальнейшем превратится в «мессиво». Сейчас, я работаю над сервером, а именно привожу код в нормальный вид, и совсем скоро буду переводить все в таблички базы данных предприятия.

[01.06.2019]

Много времени прошло с тех пор, когда я в последний раз описывал историю LightSearch Android. Надо это дело исправлять. К тому же материала накопилось предостаточно.

Первое, с чего надо начать — я переписал клиент более-менее под принципы SOLID. Почему более-менее? Все дело в том, что либо сам Android не позволит сделать это так, как я примерно себе представлял это в голове, либо потому, что мне все еще не хватает квалификации и опыта, чтобы сделать это нормально. К тому же, я не использовал какие-либо сторонние фреймворки или библиотеки. Но сейчас могу сказать, что результатом я вполне удовлетворен.

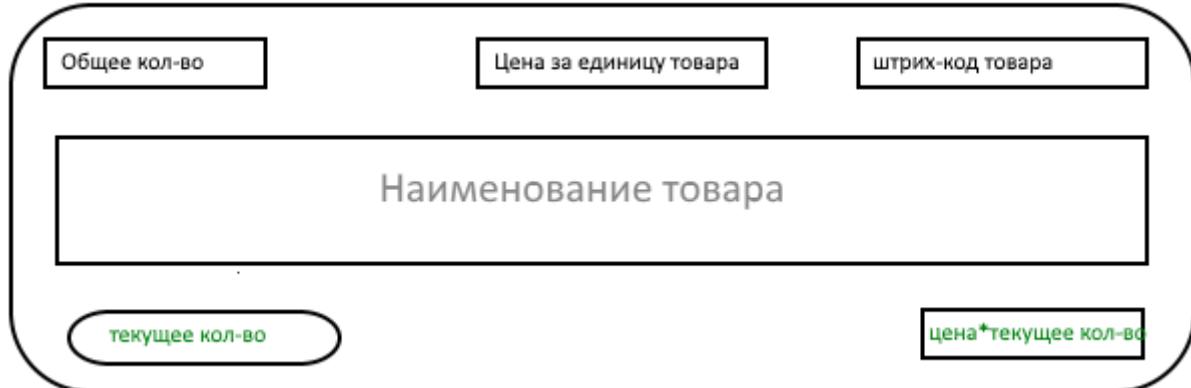
Второе важное изменение — концепция экрана мягкого чека. Когда я стал продумывать его снова, то у меня получилось это:



Для реализации свайпа влево и вправо я хотел использовать библиотеку `AndroidSwipeLayout`, но посмотрев в интернете еще побольше, увидел лучшее решение — использование `RecyclerView`.

Для того, чтобы его использовать, для начала сделать макет карточки, которая будет в нашем случае представлять товар. И я подумал о том, что если я могу создать какой угодно макет для карточки, то могу ли я использовать для изменения текущего количества товара EditText? Я попробовал, и оказалось, что да. Как же хорошо, что я не использовал ту библиотеку :) Это сделало приложение намного удобнее. Сейчас об этом подробнее расскажу.

Теперь макет выглядит так:



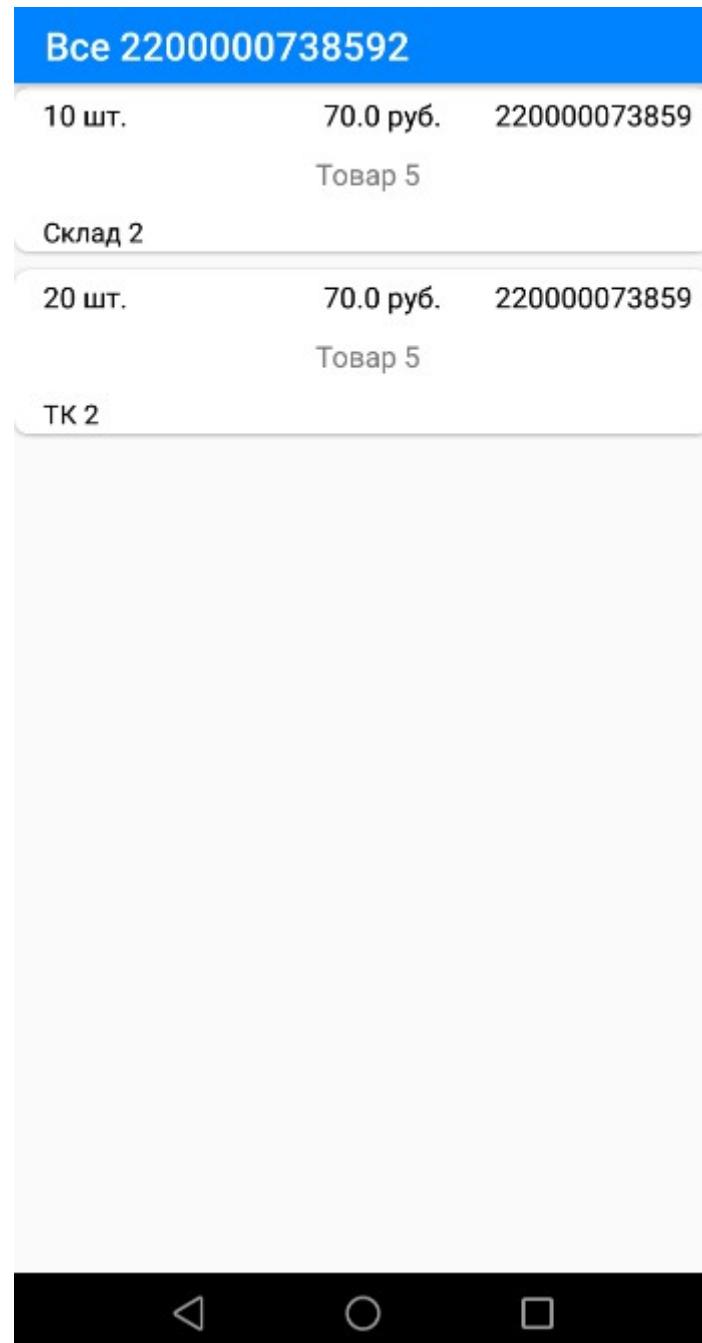
На карточке слева вверху находится общее количество товара на всех подразделениях предприятия, вверху в центре — цена за одну единицу данного товара, слева вверху — штрих-код товара, в центре — наименование товара, слева внизу — текущее количество товара, а справа внизу — текущая стоимость карточки, которая рассчитывается как цена за единицу товара, умноженная на текущее количество товара.

Все поля, за исключением текущего количества, являются TextView, то есть не редактируются пользователем. Текущее количество является EditText, и в нем пользователь может редактировать текущее количество данного товара. При этом изменяется значение текущей цены данной карточки, и общая сумма мягкого чека.

По ТЗ количество товара измеряется числом с плавающей запятой. Так сделано потому, что у них есть некоторые товары, у которых даже единица измерения штуки, которые можно продать не целым значением. Например, дверные откосы. Их в одной пачке 5 штук, но одна штука считается за упаковку. В «Баярде» есть такая возможность, как купить только один из откосов. Таким образом, если покупатель берет один откос из пачки, то это будет 0.2 штуки. Поэтому минимальное текущее количество товара равняется 0.2, а максимальная — общему количеству товара на всех подразделениях. Таково требование заказчика.

При таком подходе мне совершенно не требуется вешать на клик по карточке какое-либо действие. Теперь свайп влево отвечает, как и раньше, за удаление товара, а свайп вправо — за вывод подробной информации о данном товаре. К тому же RecyclerView(название говорит само за себя) позволяет восстанавливать раннее удаленные карточки. В совокупности со Snackbar'ом это работает очень даже хорошо.

Прежде, чем показывать экран с мягким чеком, я хотел бы показать экран результата поиска при результате больше, чем 1 товар. Я решил его тоже сделать в виде карточки, так как я также выяснил, что и на ListView можно вешать на каждый элемент свой макет.

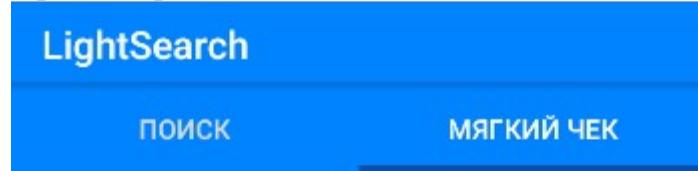


Теперь полезной информации вмещается куда больше, чем было до этого. При нажатии на карточку, также, как и прежде, открывается диалоговое окно с подробным результатом.

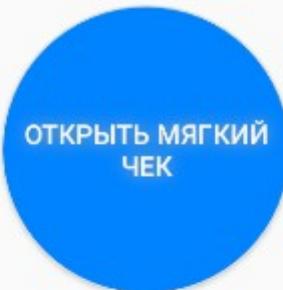
Перед тем, как помещать товары в мягкий чек, его нужно открыть. Для этого я сделал новый макет и новый фрагмент. Точнее новые фрагменты. Теперь в ContainerFragment содержится два фрагмента: SearchFragment, фрагмент поиска и SoftCheckContainerFragment, фрагмент контейнер для мягкого чека. Сначала в SoftCheckContainerFragment помещается фрагмент OpenSoftCheckFragment, а

после успешного открытия мягкого чека заменяется фрагментом SoftCheckFragment.

Вот так выглядит экран открытия мягкого чека:

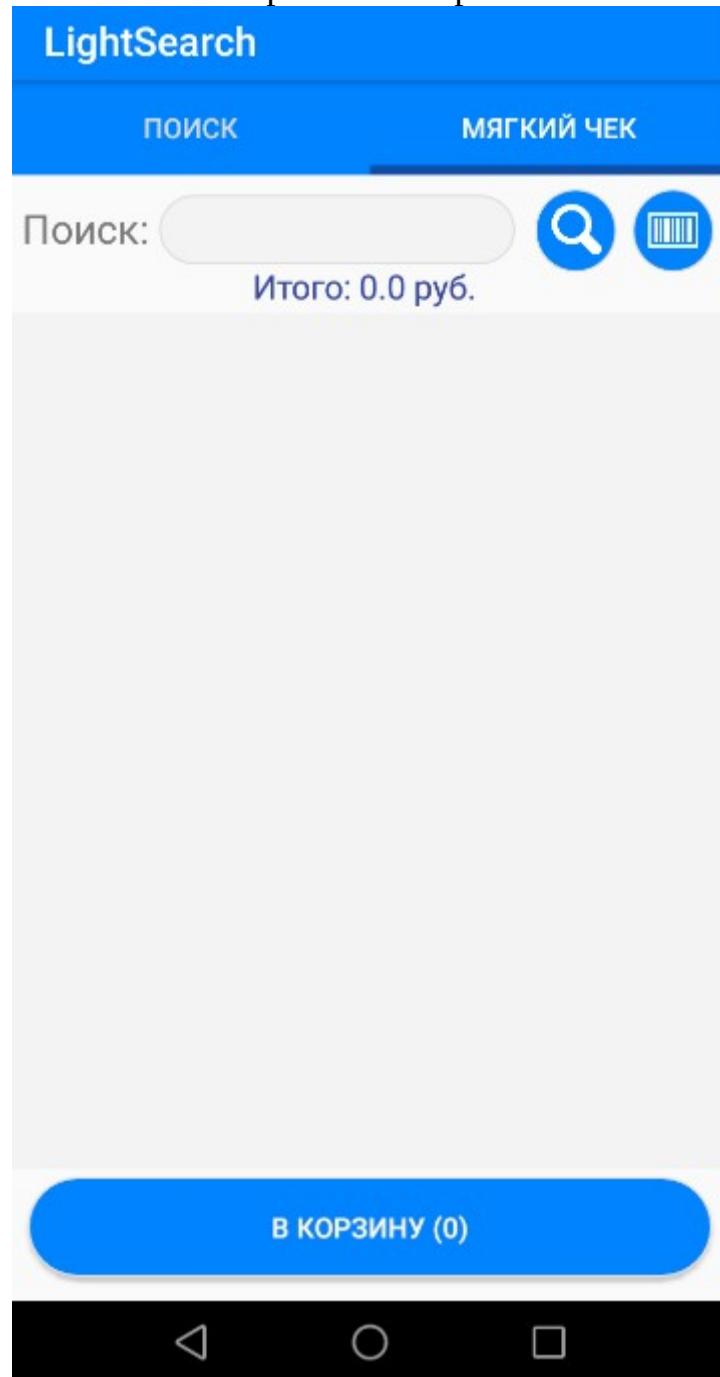


Мягкий чек не открыт.

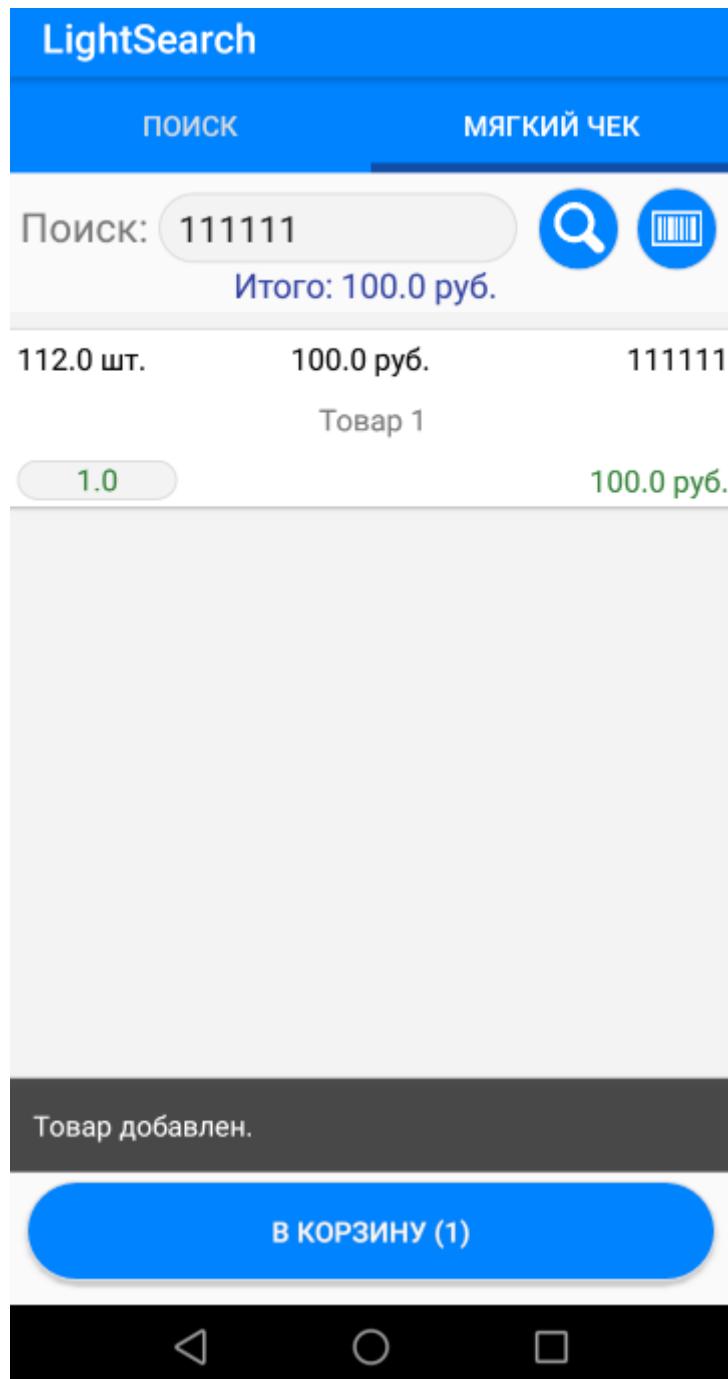


При нажатии на большую синюю кнопку «Открыть мягкий чек» запускается сканер штрих-кода, для считывания кода карточки, к которой будет привязан данный чек.

После успешного считывания открывается экран мягкого чека:



Верху находится поле поиска, в которую можно вводить штрих-код товара, и после нажатия на круглую кнопку с лупой произвести поиск. Если товар был найден, то он добавится в мягкий чек.



Также при добавлении товара работает Snackbar, который уведомляет о том, что товар был добавлен.

Если мы попытаемся снова добавить данный товар через поиск, то обновится текущее значение количества данного товара на 1.0.

Как видно из скриншота, изменилась итоговая стоимость всего мягкого чека.

Также можно нажать на круглую кнопку с штрих-кодом. Тогда откроется сканер штрих-кода, и после удачного считывания будет произведен поиск товара. Если товар найден, то он добавится в мягкий чек.

# LightSearch

ПОИСК

МЯГКИЙ ЧЕК

Поиск:



Итого: 170.0 руб.

30.0 шт.

70.0 руб.

2200000738592

Товар 5

1.0

70.0 руб.

112.0 шт.

100.0 руб.

111111

Товар 1

1.0

100.0 руб.

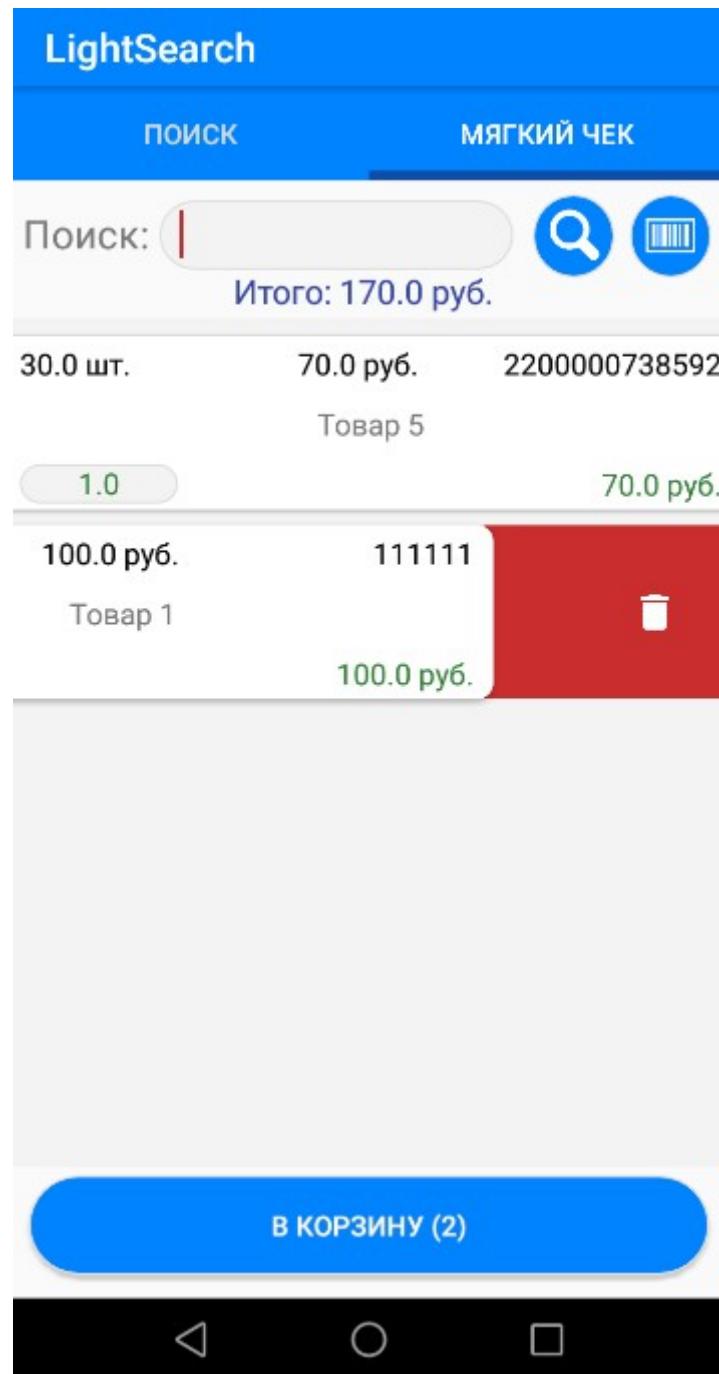
Товар добавлен.

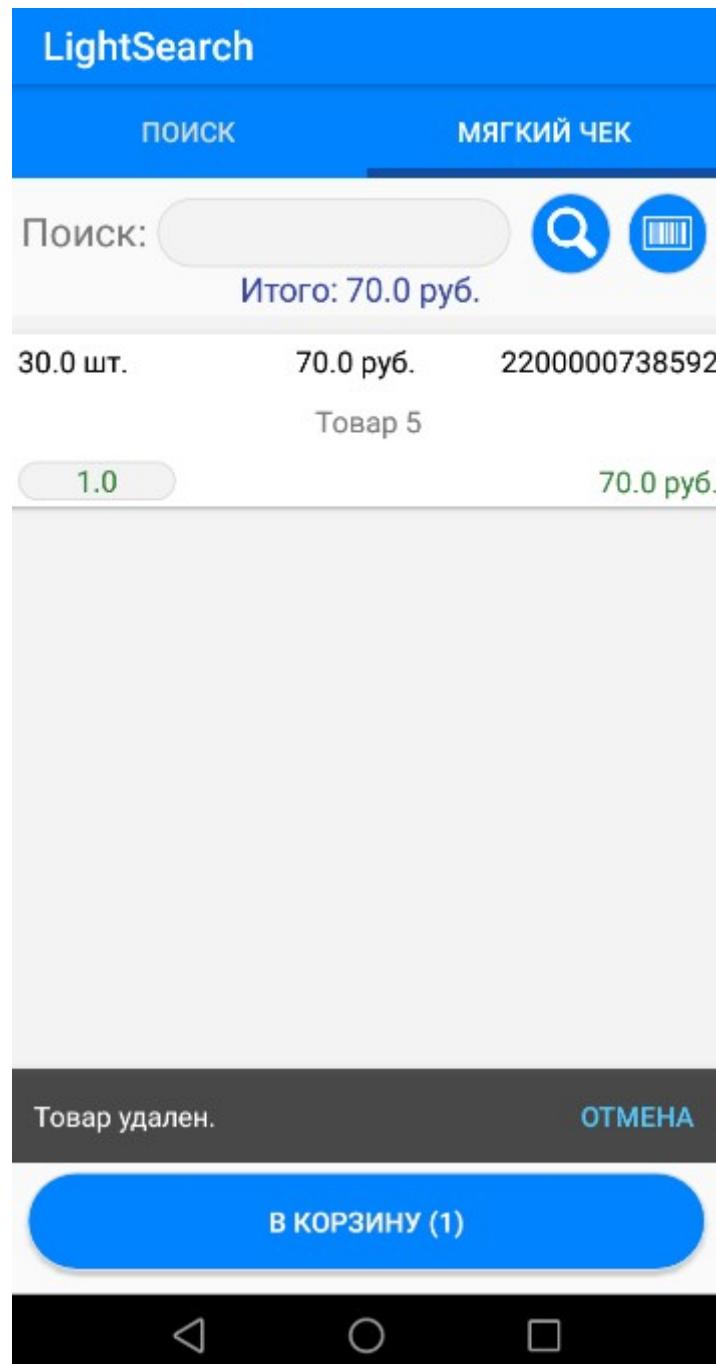
В КОРЗИНУ (2)



Забыл упомянуть о том, что в скобках на кнопке «В Корзину» число уникальных товаров.

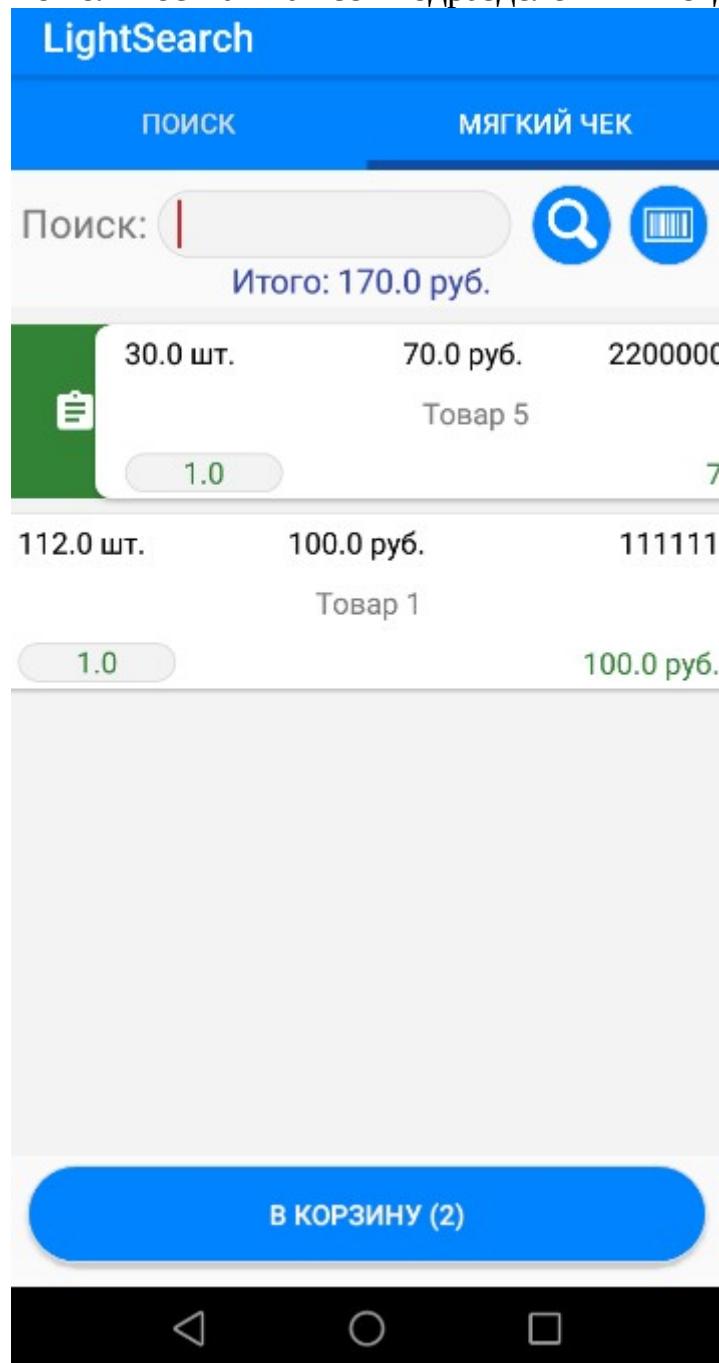
Свайп влево удаляет товар из мягкого чека.

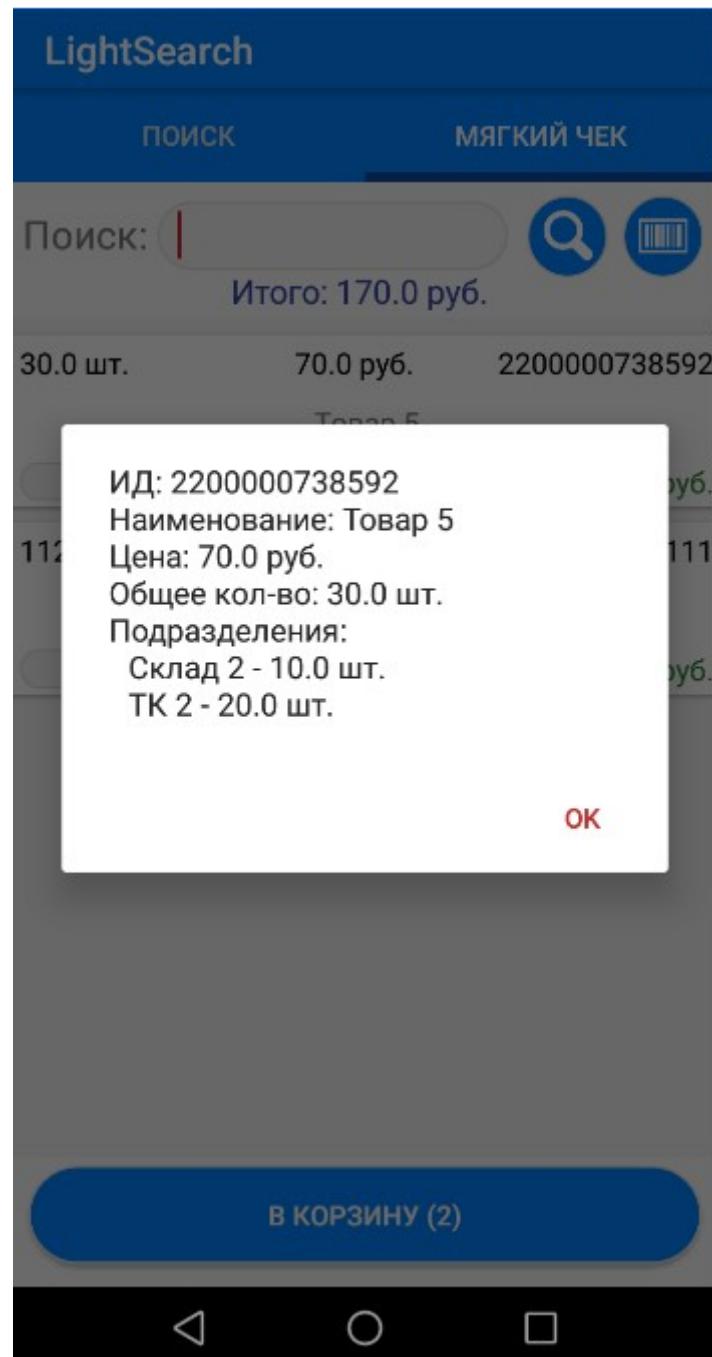




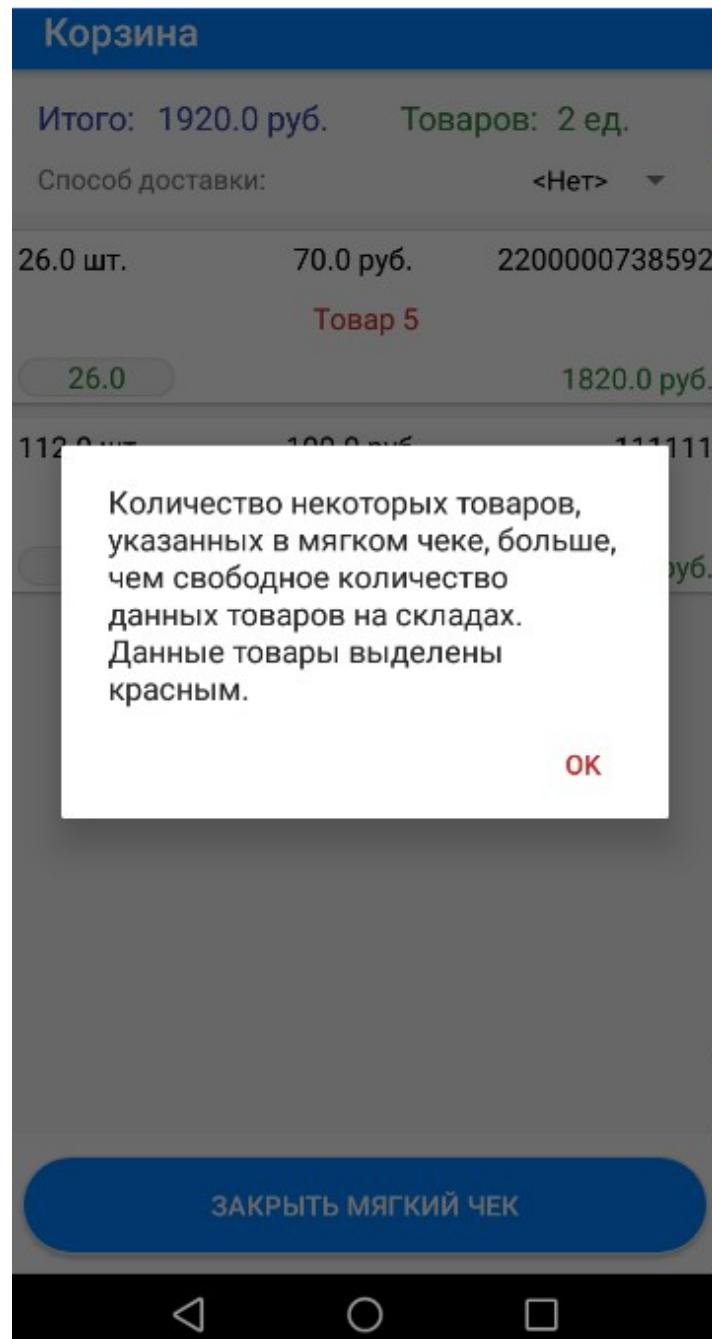
Snackbar уведомил о том, что товар был удален, и имеется возможность отменить текущее действие, нажав на кнопку «Отмена» на Snackbar'e.

Свайп вправо открывает полную информацию о товаре. В ней содержится также информация о количествах на всех подразделениях в отдельности.





После того, как все товары были выбраны, продавец нажимает кнопку «В корзину». Перед этим LightSearchAndroid отправляет LightSearchServer'у команду на подтверждение количества товаров в мягком чеке. Описание протокола есть в документе, посвященный LightSearch Server'у. Если есть какие-то проблемы, то приложение об этом сообщит.



Как видим, «Товар 5» выделен красненьким, и его значение поменялось с 30 до 26 штук. Также видно, что текущее количество «Товара 5», введенное ранее (30 шт.) поменялось также на 26.

## Корзина

Итого: 1920.0 руб. Товаров: 2 ед.

Способ доставки:

26.0 шт.

26.0

112.0 шт.

1.0

100.0 руб.

<Нет>

7 92

Доставка со складов

Самовывоз со складов 16.

10 11

Самовывоз с ТК

ЗАКРЫТЬ МЯГКИЙ ЧЕК



Сверху написаны итоговая сумма чека и количество уникальных товаров. Также есть выбор доставки. Поле <Нет> введено для того, чтобы сделать меньше вероятность закрытия мягкого чека с неправильной доставкой.

В корзине мы также можем удалять товары свайпом влево, и просматривать общую информацию о товаре свайпом вправо. Только теперь если товар подсвечен красным, то в подробной информации будет показано количество до проверки количества товара и после. Если же с количеством все хорошо, то просто пишется общее количество товара.

## Корзина

Итого: 520.0 руб. Товаров: 2 ед.

Способ доставки: Доставка со складов ▾

26.0 шт. 70.0 руб. 2200000738592

Товар 5

6.0

420.0 руб.

112.0 шт. 100.0 руб. 111111

ИД: 2200000738592

Наименование: Товар 5

Цена: 70.0 руб.

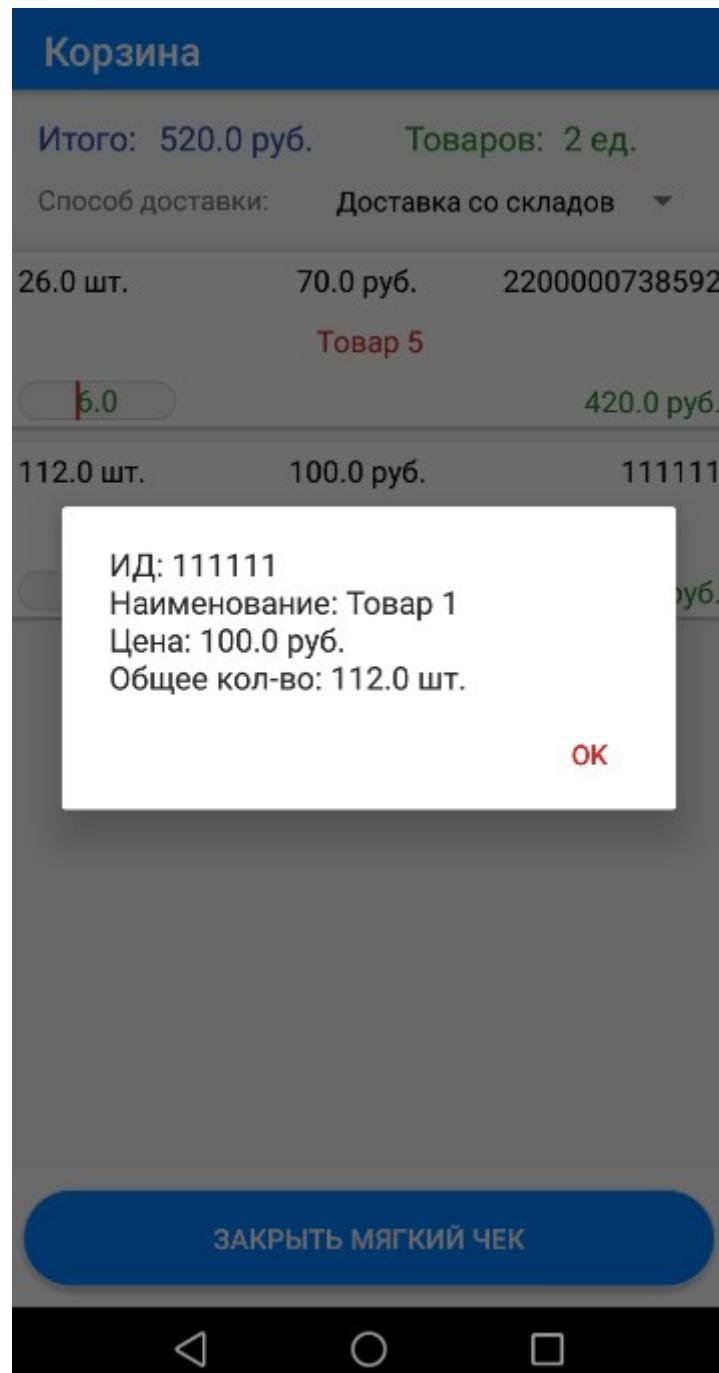
Общее кол-во ДО: 30.0 шт.

Общее кол-во ПОСЛЕ: 26.0 шт.

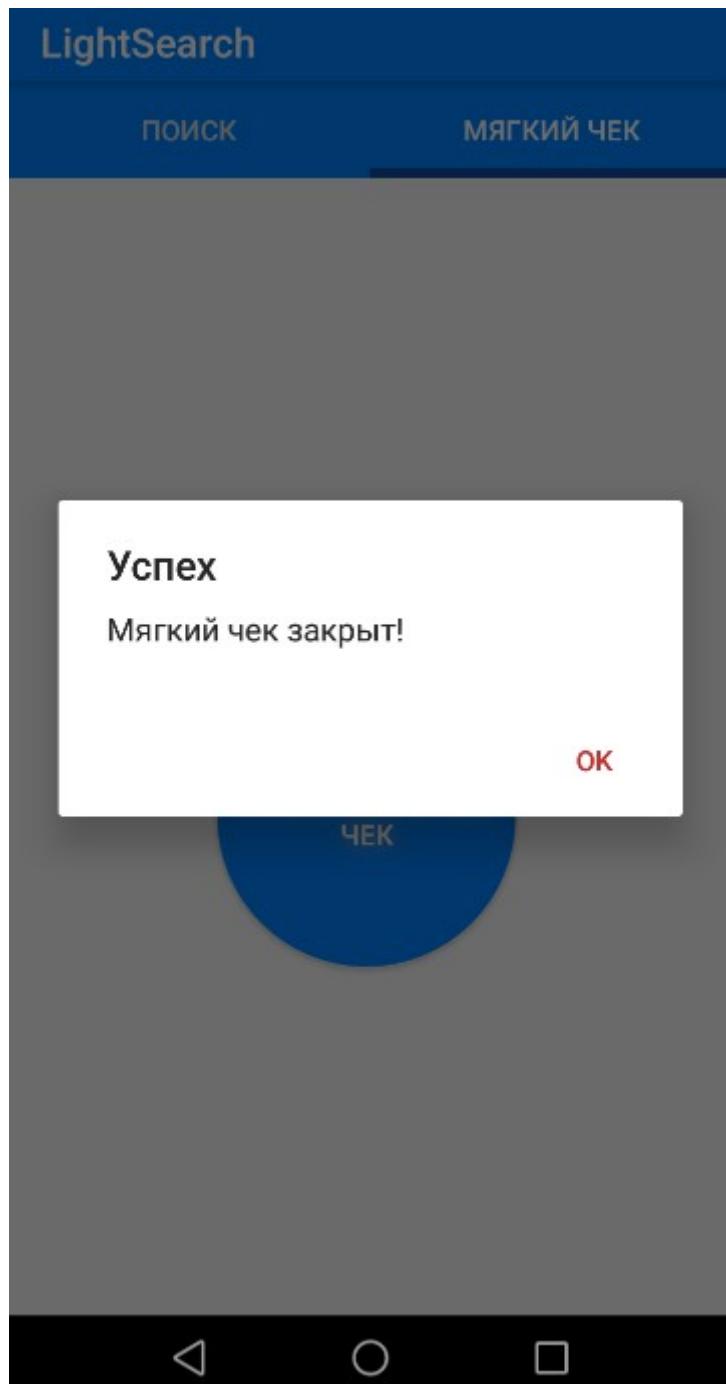
OK

ЗАКРЫТЬ МЯГКИЙ ЧЕК





После нажатия на кнопку «Закрыть мягкий чек» клиент еще раз отправляет серверу команду о подтверждении количества товаров. Если текущее количество снова превышает текущее количество данного товара на складах, то снова выводится соответствующее сообщение, и чек не будет закрыт. Иначе клиент отправляет серверу команду на закрытие мягкого чека, и если все хорошо, то чек будет успешно закрыт.



Вот так это на текущий момент выглядит и работает.

Было много подводных камней. Например, возникла проблема с тем, что при выполнении какой-либо команды не отображалось диалоговое окно с индикатором о том, что процесс выполняется. Выяснилось то, что нельзя вызывать у AlertDialog метод show и dismiss не то что в разных классах, а в разных методах. Если вызвать show в одном месте, а dismiss в другом, то диалоговое окно никогда не закроется, при этом оно не отображается вообще. То есть создается ощущение, что приложение просто зависло. Если же вызывать show и dismiss в одном методе, то они просто не отображаются. В интернете с полной уверенностью писали о том, что так будет работать и что так можно делать, но увы, нет. Так не работает.

Потом я случайно вспомнил, что когда только начал свое знакомство с Android, я мельком читал про AsyncTask, но так и не применил его к своему проекту. Теперь же я принял решение изучить его более детально, и не зря: данное решение решает мою задачу легко и элегантно. Поначалу мне AsyncTask показался чем-то страшным и непонятным, но все оказалось проще, чем я думал. И на данный момент он справляется со своей задачей так, как надо.

Повозился много с адаптером для RecyclerView, особенно с TextWatcher'ом, который ну очень какой-то кривой :( Но я с ним более-менее справился, работает как надо :)

Были проблемы с тем, что товары мягкого чека не сохранялись, если вдруг пользователь захотел сделать поиск на экране поиск, результат которого перекинул бы его на окно результата поиска. Пришлось интерфейс SoftCheckRecord, который является сущностью товара(строки, карточки) мягкого чека, сделать расширением интерфейса Parcelable. Все дело в том, что перед паузой фрагмент вызывает метод onSaveInstanceState, в качестве параметра которого передается Bundle. В этот Bundle можно сохранить значение переменных или объектов, которые потом можно будет считать в методе OnCreate, когда фрагмент выйдет из паузы. Если с типами данных все просто(просто записываем их и все), то с объектами все намного сложнее. Принимаются только те объекты, которые реализуют интерфейс Parcelable. Именно по этой причине SoftCheckRecord расширяет Parcelable.

Из-за использования Function<F, T> пришлось перейти на SDK минимум версии 24, а это Android 7.0. К сожалению, версии Android ниже 7.0 не поддерживают Function<F, T>. Зато после перехода стало возможным использование лямбда-выражений. И обновил ядро ZXing. Теперь сканер работает быстрее, чем раньше.

Ну и напоследок — рисовать иконки для кнопок та еще морока -_-

В планах проверить работоспособность приложения при сервере, который будет подключаться к базе, написать для тех классов, для которых возможно, юнит-тесты, и потихоньку улучшать код. После того, как данная версия будет полностью готова, надо будет написать новую версию сервера на фреймворке Spring. Это также коснется и Android, так как вместо сокета будет использоваться REST-запросы.