ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК
КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ УЧЕТА РАСХОДОВ ФИЗИЧЕСКИХ ЛИЦ

Выпускная квалификационная работа
обучающегося по направлению подготовки
02.03.03 Математическое обеспечение и администрирование информационных систем
очной формы обучения,
группы 07001402
Попова Дмитрия Вадимовича

Научный руководитель:
доцент Бурданова Е.В.

БЕЛГОРОД 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С развитием человека как одного из самых умных существ на земле, человек стал все больше узнавать новой информации которую нужно было запоминать и постоянно держать в уме, с самого начала это была информация основанная на рефлексах и природных инстинктах, дальше появилось взаимодействие людей между собой и общение, жесты и несвязные звуки переросли в язык, язык дал возможность для разговора и договоров между людьми, что в свою очередь дало товарный обмен, бартер и торговлю. Все нюансы человеку нужно было держать в голове чтобы не стать жертвой хищника, поддерживать диалог и успешно обмениваться товарами. С течением времени все аспекты жизни человека начали все больше углубляться в детали и становится сложнее с каждым веком, который приносил новые открытия, научные труды а также бытовую мудрость и опыт.

В какой-то момент информации стало слишком много и человеческий мозг больше не мог помнить все ее детали. Тогда люди начали записывать всю важную информацию на бумагу, придумав при этом себе ежедневники, торговые и чековые книги. С наступлением цифровой эры информации стало в разы больше, но отпала надобность держать все ее в голове, когда любая информация доступна в несколько кликов на компьютере или телефоне. Что бы человеку освободить место для новой более важной информации в голове, ему требуется качественное, удобное и надежное программное обеспечение, которое максимально будет удовлетворять все потребности человека.

Целью данной дипломной работы является приложение которое будет предоставлять помощь в таких важных и требующих постоянного подсчета вещах, как затраты физических лиц.

В ходе данной выпускной квалификационной работы будет разработана автоматизированное приложения поддержания и ведения всех расходов физических лиц, способное ответить всем потребностям в хранении и помощи учета всех финансовых затрат появляющихся в цифровой век.

Основными задачами данного приложения будут являться сканирование чеков из магазинов, аптек и других торговых площадок, сохранение чеков и отображение в удобном формате с возможностью сортировки и детализированной системы типизации.

Для того чтобы достигнуть перечисленных выше задач, следует указать ключевые задачи:

1. анализ особенностей финансовых операций;
2. выявление основных задач приложения;
3. обзор аналогичных приложений;
4. проектирование приложения;
5. реализация спроектированного приложения;

Данная выпускная квалификационная работа состоит из введения, четырех глав, заключения, списка литературы и приложения.

Введение содержит общие сведения о работе, ее актуальность, цели, задачи и способы их достижения.

Первая глава содержит анализ финансовых операций, обзор аналогичных приложений и постановку задач.

Вторая глава состоит из проектирования инфологической и даталогической модели базы данных.

В третьей главе проводится описание средств разработки и разработка приложения.

В ходе выполнения четвертой главы разрабатывается методика испытаний, а так же проводятся соответствующие испытания.

В заключении подводится итог проведенной работе.

Выпускная квалификационная работа состоит из 63 страниц, 30 рисунков 6 таблиц и приложения.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ

1.1. Особенности финансовых операций

Человек находится в постоянном движении, каждый день ему нужно решать очень важные для него дела, работа, быт, дети. Каждое из этих дел не обходится без затрат как моральных так и финансовых. В информационную эру с появлением всевозможных электронных расчетных систем как карточки, безналичный расчет и терминалы, человеку стало трудно следить за своими покупками имея после каждой, бумажный чек, который так легко потерять, порвать или случайно выкинуть вместе с остальными скопившимися старыми чеками в кармане.

У каждого человека случается момент когда он садится, задумывается и спрашивает себя «Куда же я потратил деньги?». Когда такой момент наступает, каждый начинает вспоминать какие у него были за этот месяц покупки, и когда не может вспомнить, начинает искать чеки или какие либо записи о своих покупках, но так как магазинные чеки зачастую не сохраняются то у человека возникают порой большие дыры в финансовой отчетности.[14]

Из этой неприятной ситуации есть несколько выходов:

- Складывать все чеки в коробочку и перебирать их каждый раз когда появляется надобность. У такого метода есть большие минусы с тем, что за год или месяц скапливается очень большое количество чеков, которые нужно будет где-то хранить, так же не все чеки доживут до складирования и попросту порвутся, утилизируются или потеряются что не дает уверенности в данном методе. Следующим минусом является фильтрация и поиск определенной даты, типа товара или магазина, в большом количестве бумаги с очень мелким шрифтом это будет не самым приятным занятием.

- Записывать чеки в книгу или тетрадь на подобии товарной книги. Главным минусом данного метода является большая потеря времени на перенос данных из чека в книгу. Чеки после архивирования таким образом невозможно будет фильтровать и систематизировать без переписывания упорядоченных чеков и значений в другую тетрадь.
- Для пользователей персональных компьютеров также возможен вариант перенос чеков в специальные программы типа «Excel» что дает более удобную и быстрою сортировку чеков по любым критериям в любой момент. Но минусы у такого метода все равно есть, большая трата времени на перенос чеков в электронный справочник.

В наше время, хоть и наблюдается экономический кризис в стране, но на полках очень большое разнообразие товаров как отечественных так и зарубежных производителей которые с каждым годом вносят все большее количество продукции, которые жители с удовольствием себе не отказывают покупать. Таким образом у нас складывается огромный массив товаров на которые среднестатистический житель тратит свои деньги. Так же каждой отдельной группе товаров отведен отдельный тип который и объединяет товары по одному или нескольким признакам.[15]

Каждый товар из каждой группы должен соответствовать своей группе и быть типизирован для упрощения фильтрации и удобства ведения отчетности. Для того что бы разбить чек по товарам не совершая лишних манипуляций можно прибегнуть к использованию закона от 22 мая 2003 г. № 54-ФЗ о контрольно-кассовой технике.[13] Одним из требований Закона № 54-ФЗ в статье 4.7 является возможность печати «QR-кода»» на всех кассовых чеках. На данный момент печать «QR-кодов» на кассовых чеках является желательной услугой но не обязательной что относиться и к печати наименования и количества товаров только для владельцев ИП на спецрежимах которые не производят торговлю табака и алкогольной продукции и интернет магазинов только до 1 февраля 2021 г. (п. 17 ст. 7 Закона

№ 290-ФЗ), а во всех остальных местах торговли, где еще не производится печать QR кода на кассовом чеке, такая услуга должна появиться до 1 июня 2018 г. (п. 9 ст. 7 Закона № 290-ФЗ, которым внесены изменения в Закон № 54-ФЗ).

Кроме «QR» кода на чеке имеется много важной для покупателя информации. Эту информацию продавец предоставляет покупателю для того что бы покупатель имел полный отчет о законности, точности и правильности финансовой операции совершаемой продавцом.

Информация поступает в чек из нескольких источников. Это и техника, и программное обеспечение, которое связывает несколько устройств в одну систему. Например, фискальный накопитель, прикассовые весы, сканер штрих-кодов. На рисунке 1.1. представлены пути поступления информации и ее источники.

Часть реквизитов поставляет программное обеспечение кассового аппарата (см. рис. 1.1.), а именно:

- тип чека (приход или расход);
- номер чека;
- количество купленного товара;
- стоимость нескольких единиц одного товара;
- сумма налога;
- итоговая сумма покупки;
- форма расчета (наличная или безналичная форма, полученная сумма от покупателя, размер сдачи);
- сумма НДС;
- электронная почта покупателя и отправителя;
- дата и время покупки;
- рекламный текст.

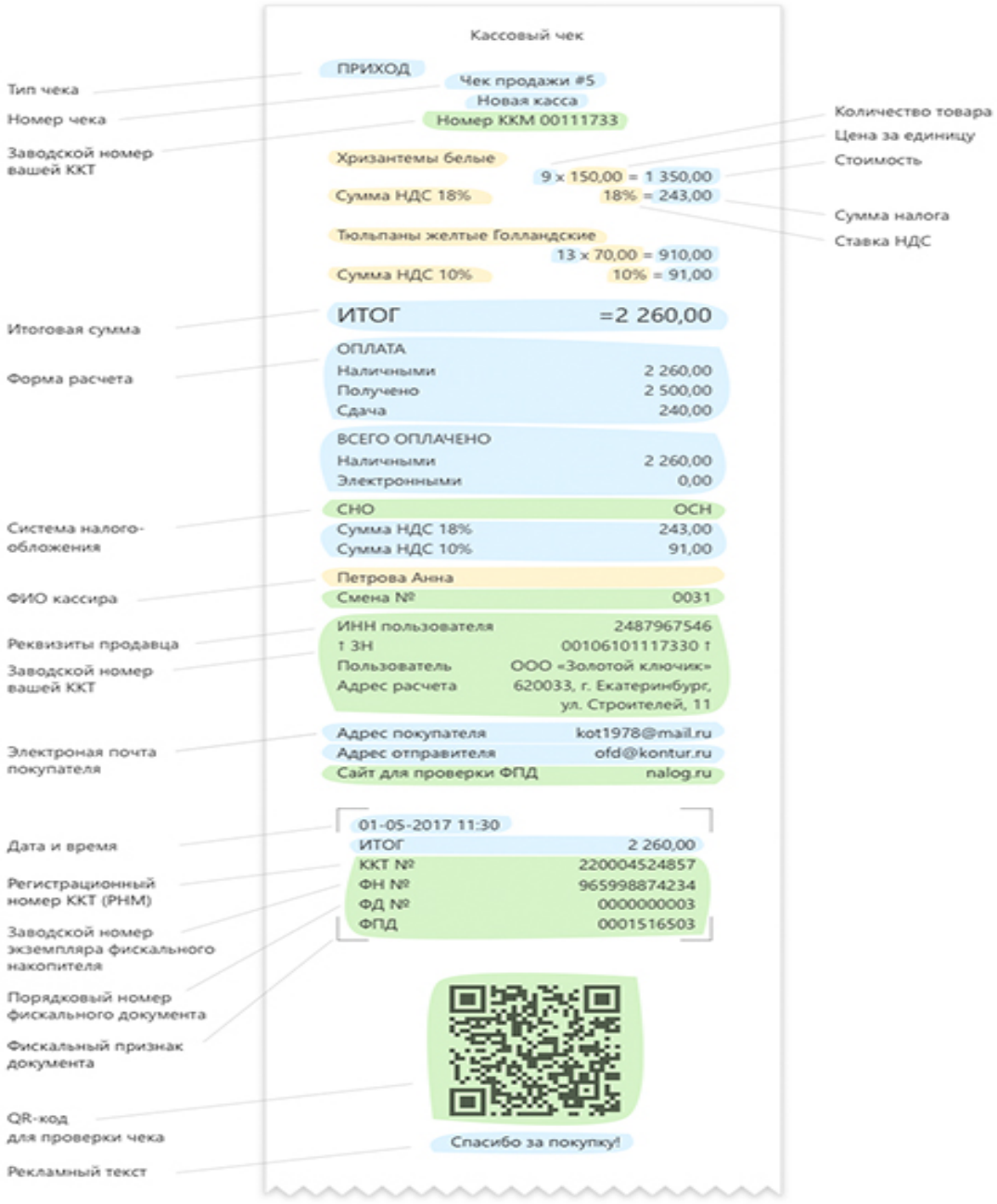

Рис. 1.1. Источники информации для кассовых чеков.

Некоторые данные в чек поступают непосредственно из товароучетной системы :

- наименование товара;
- стоимость единицы товара;
- ставка НДС;
- ФИО кассира.

Еще часть реквизитов отправляет в чек сама онлайн-касса (см. рис. 1.1.):

- заводской номер ККТ;
- система налогообложения;
- номер смены;
- реквизиты компании-продавца;
- адрес сайта для проверки фискального признака документа (ФПД);
- регистрационный номер ККТ (регистрационный номер машины, РНМ);
- заводской номер экземпляра фискального накопителя;
- порядковый номер фискального документа;
- фискальный признак документа;
- QR-код.

Некоторые реквизиты в чеке могут отсутствовать, они не являются обязательными. Например, реквизит – «контакты покупателя и отправителя». Если покупателю нужна электронная копия бумажного чека, то кассир обязан ее предоставить. Для этого во время формирования чека кассир вводит адрес электронной почты или номер мобильного телефона покупателя.[15] В этом случае кассовое ПО автоматически подставит в чек не только контакты покупателя, но и e-mail магазина-отправителя этого чека (см. рис. 1.2.).

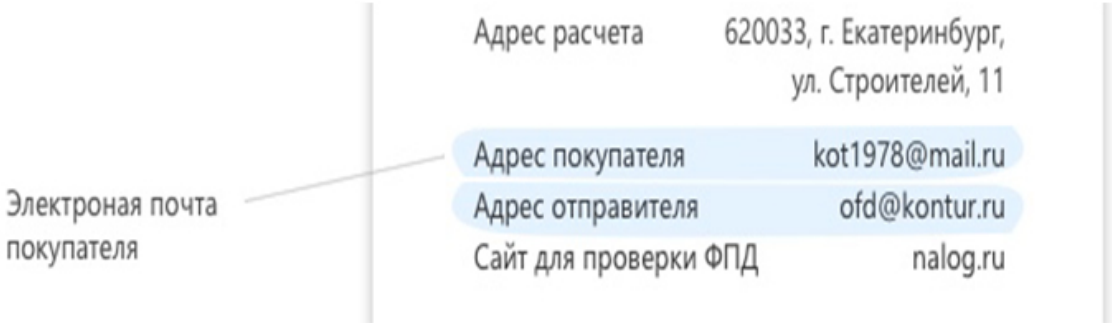

Рис. 1.2. Контакты покупателя и продавца.

Если клиенту не требуется электронный дубликат, то бумажный чек напечатается без строк «Адрес покупателя» и «Адрес отправителя».

Следующий из необязательных реквизитов – «заводской номер ККТ». Его не следует путать с регистрационным номером кассового аппарата, который присваивает налоговая инспекция в момент постановки кассы на учет. Регистрационным номером кассового аппарата изображен на рисунке 1.4. Заводской номер кассе присваивает изготовитель, а кассовое ПО автоматически добавляет этот номер в чек.[17] Эта информация может дублироваться в разных товарных позициях на чековой ленте, хотя и не является обязательной для указания в ней (см. рис. 1.3.).

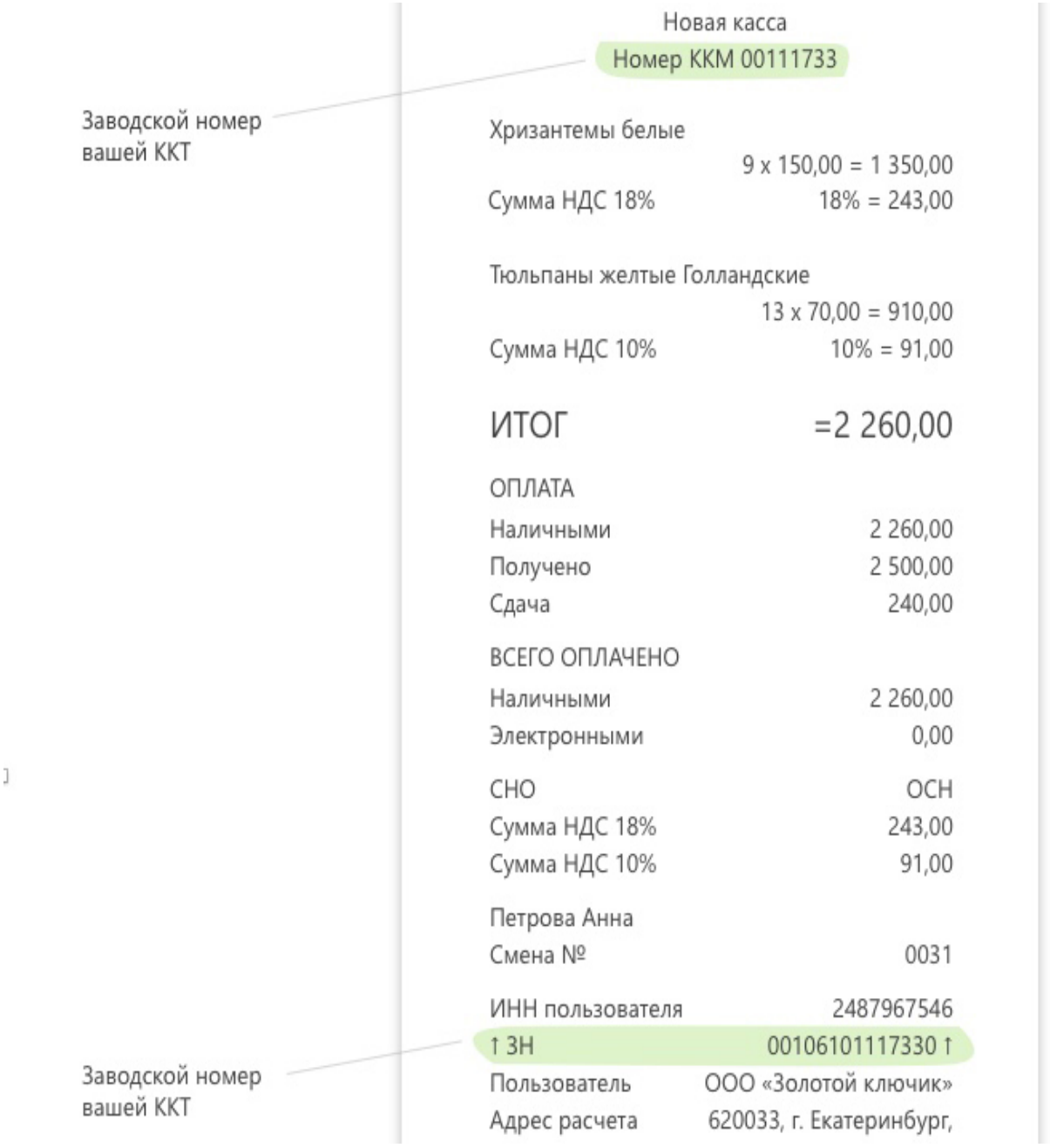

Рис. 1.3. Заводской номер контрольно-кассовой техники.

Обычно в начале чека есть надпись «Добро пожаловать!», а в конце — «Спасибо за покупку!». Эти надписи называются клише (вверху) и рекламный текст (внизу) (см. рис. 1.5.).

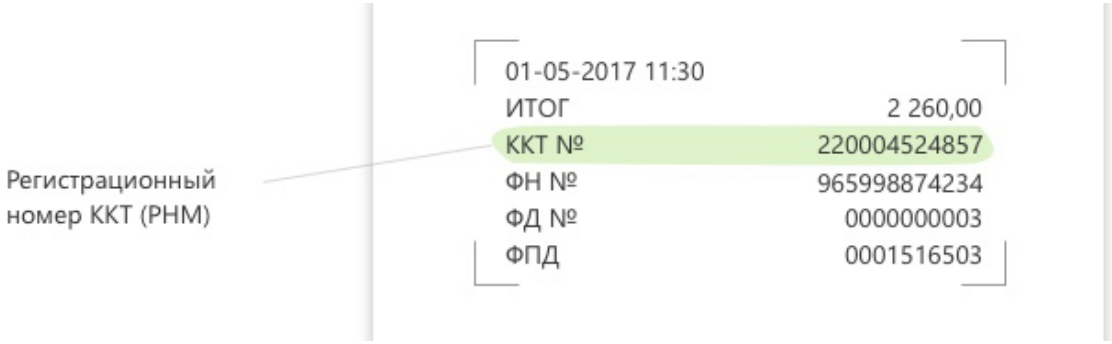

Рис. 1.4. Регистрационный номер контрольно-кассовой техники.

Без этих строчек чек считается законным, однако их нетрудно создать во время настройки кассового ПО — внимание к покупателю никогда не помешает.

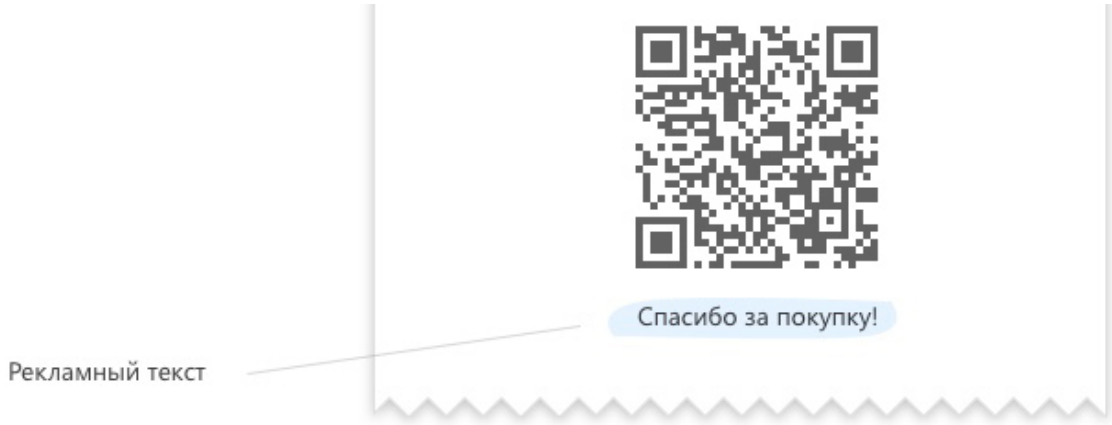

Рис. 1.5. Рекламный текст.

Налоговые органы ввели QR-код как инструмент гражданского контроля. Покупатель может скачать бесплатное мобильное приложение

«Проверка кассового чека в ФНС России», отсканировать QR-код и получить от налоговиков информацию о добросовестности продавца.

В списке требований к кассовому чеку «QR-код» не упомянут, однако о нем говорится в статье 4 Закона № 54-ФЗ «Требования к контрольно-кассовой технике». Там, в частности, сказано, что ККТ должна «обеспечивать возможность печати на кассовом чеке» «QR-кода» размером не менее 20 х 20 мм. Получается, что «QR-код» — реквизит не обязательный, но касса должна уметь его печатать. Пример «QR-кода» изображен на рисунке 1.6.

Рис. 1.6**.** Матричный «QR-код» для проверки чека покупателем с помощью мобильного приложения ФНС.

В закодированном виде в «QR-коде» содержится такая информация:

- дата и время покупки;
- порядковый номер фискального документа;
- признак расчета (приход или расход, возврат);
- сумма расчета;
- заводской номер фискального накопителя;
- фискальный признак документа.[1]

1.2. Обзор и анализ аналогичных приложений

В нынешний информационный век с каждым днем растет количество всевозможных приложений и программ которые создавались для облегчения той или иной сферы человеческой деятельности. Каждая новая программа если она не первая в своем роде, создается на анализе характеристик аналогичных программ и несет в себе все то, чего не хватало предыдущим представителям для максимального удобства и производительности. Каждый разработчик не смотря на свой уровень, желание заработать или помочь людям пытается вытеснить другие программы своим продутом в котором по его мнению собраны все нюансы открывающие новый уровень удобства, функциональности и дизайна.

Как бы не старался разработчик его приложение не всегда удовлетворяет требованиям и потребностям всех пользователей в той мере в которой они желают видеть продукт. Чтобы в этом убедиться рассмотрим несколько аналогичных приложений и оценить на сколько хорошо они справляются с поставленной задачей.

Приложение 1. «Деньги ОК»

Первым рассмотрим приложение группы разработчиков «Mobion» которое было скачано из стандартного магазина «iOS» приложений «AppStore». Данный продукт имеет обычный, распространённый дизайн что можно увидеть на рисунке 1.7. Рассмотрим основные функции которые должно иметь приложение для ведения финансовой отчетности.

1) Добавление новой записи о потраченных средствах происходит с главного экрана (см. рис. 1.7.) приложения с минимум критериев для отчетности. Критериями выступают сумма которую пользователь

вводить в калькуляторе расходов, число которое присваивается автоматически и тип потраченных средств. Тип приобретенного товара или услуги выбирается из 10 стандартных иконок, в приложении имеется возможность создавать свои типы товаров, но для этого требуется купить Pro версию приложения.

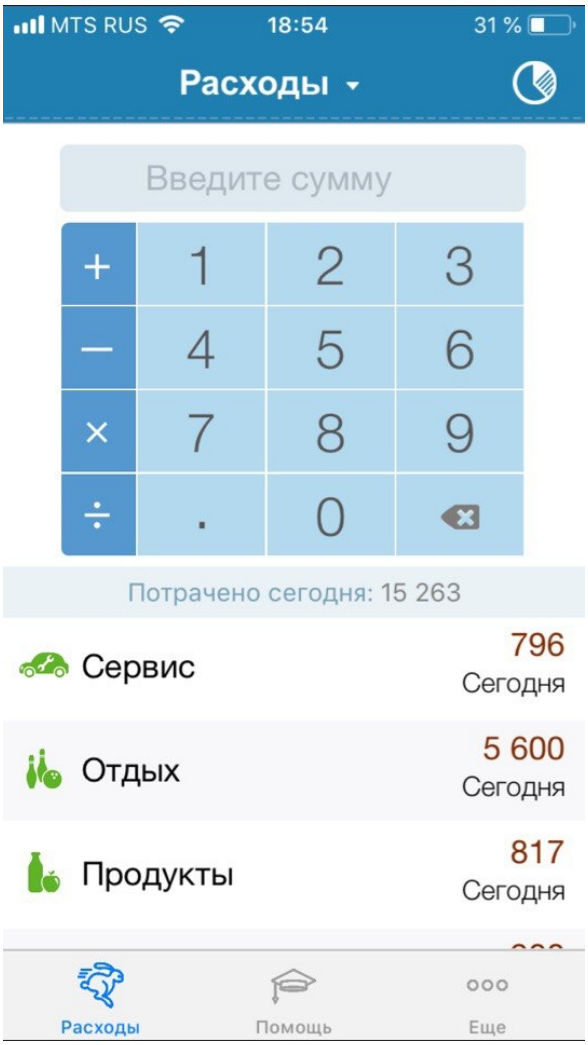

Рис. 1.7. Главный экран приложения «Деньги ОК».

2) Просмотр списка расходов производится двумя способами, на главной странице под калькулятором цен находится список и константного числа записей, содержащий последние добавленные записи о трате средств (см. рис. 1.7.). Так же просмотреть список трат можно и на экране «Отчет» (см. рис. 1.9.), на этом экране список товаров выводится строго за месяц, но так же присутствуют пункты день и год

которыми возможно пользоваться только при покупке «Pro» версии приложения.

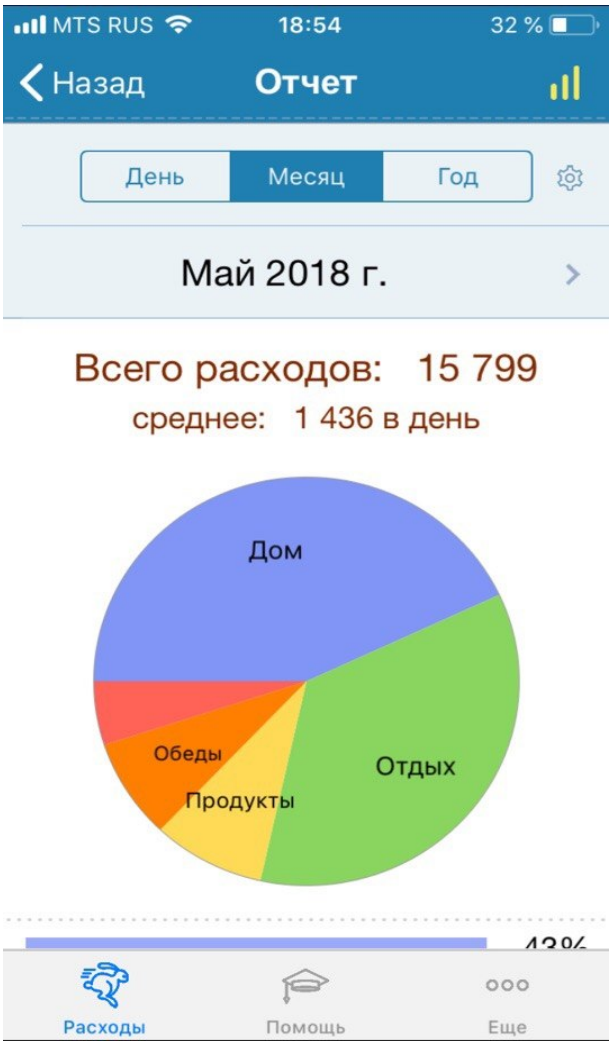

Рис. 1.8. Экран с отчетом расходов за месяц.

3) Ведение отчетности и составления графиков в данном приложении показано на рисунке 1.8. Строится круговая диаграмма из записей трат пользователя за месяц, за исходные данные берется тип покупки, услуги и количество потраченных средств. На самой круговой диаграмме кроме как названия типа для определённой области нечего не находится, что вызывает некоторые трудности с определением процента затраченных от общей суммы средств. Так же на экране отчетности присутствуют строки в которых записаны сумма расходов всех типов за месяц и среднее значение затрачиваемых в день. Ниже диаграммы расположен список со всеми совершенными за месяц покупками,

который отсортирован по убыванию. В каждой ячейке списка находится иконка типа товара, линия показывающая процент затраченных от общей суммы денег и количество затраченных на данный тип средств.

Рис. 1.9. Экран со списком расходов за месяц.

4) Фильтрация, сортировка и поиск необходимые для точного анализа финансовых затрат отсутствуют в данным приложении полностью, такой возможности нет даже в «Pro» версии приложения, что является большим минусом.

Рассмотрев данное приложение и оценив все возможности, можно сделать вывод, что данное приложение слабо выполняет свои прямые функции, имеет скудный функционал и создано больше для заработка на нем денег чем для облегчения финансовой отчетности.

Приложение 2. «Pocket Finance»

Следующим рассмотренным приложением будет «Pocket Finance» от разработчика «Rudak Aliaksei», которое было так же скачано из стандартного магазина «AppStore». Приложение имеет минималистичный дизайн в мягких тонах приятных для глаз.

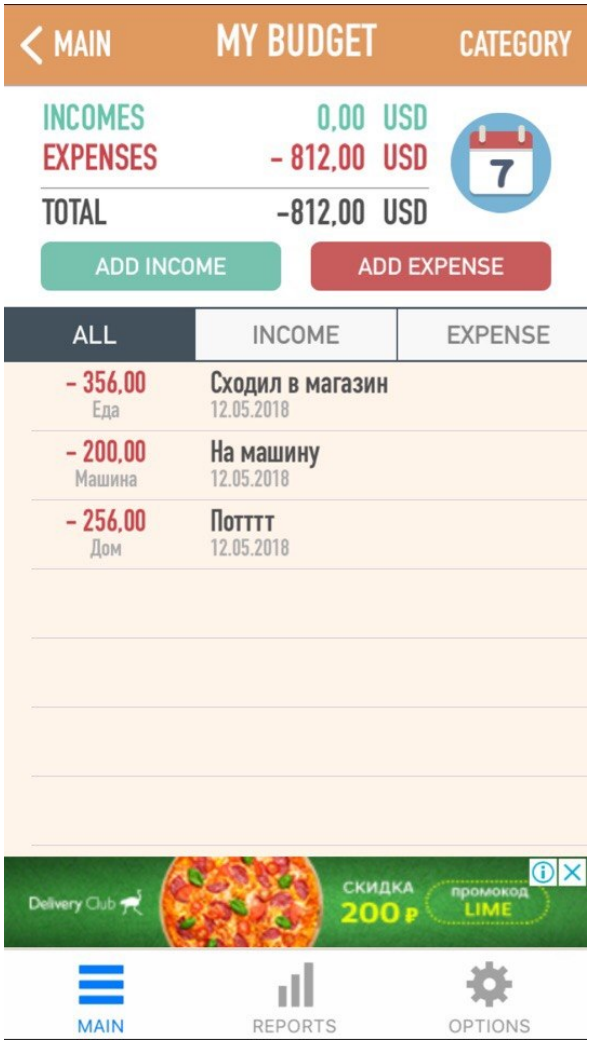

Рис. 1.10. Главный экран приложения.

1) На главном экране расположен список ваших затрат и пополнений, приложение умеет работать не только с затраченными средствами, но и бюджетом (см. рис. 1.10.). Для добавления записи о потраченных вами средствах нужно нажать на кнопку «ADD EXPENSE», что в свою очередь откроет экран с полями для добавления

нового чека (см. рис. 1.11.). В данном приложении обязательным условием служит поле «NAME» которое предполагает ввод комментария к покупке. Поле дата в котором сразу записана сегодняшняя дата, но ее можно изменять, так же сумма вашей «AMOUNT» и тип покупки «CATEGORY». Тип покупки предполагает только название вашей покупки. Типы создаются отдельно пользователем и служат только для визуального различия покупок.

Рис. 1.11. Экран добавления записи о расходах.

2) Просмотр списка ваших расходов происходить только на главном экране приложения (см. рис. 1.10.). В списке можно отобразить все типы записей «ALL», только пополнения бюджета «INCOME» или только ваши траты «EXPENSE». В ячейке списка ваших расходов отображается сумма

вашей затраты, тип который был выбран, комментарий и дата, все данные что были введены на экране добавления расходов (см. рис. 1.11.).

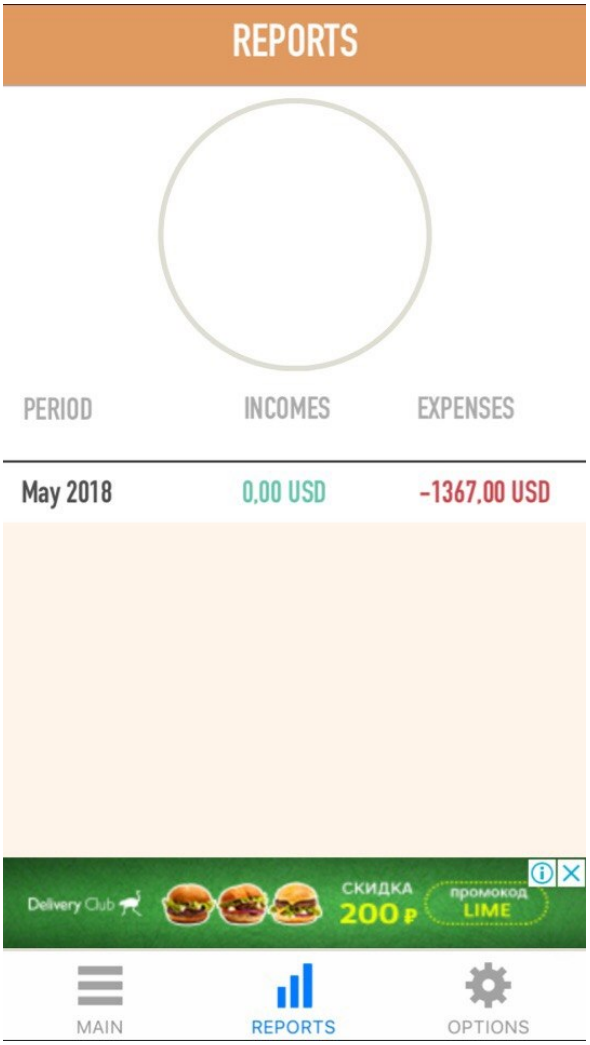

Рис. 1.12. Экран с месячной отчетностью.

3) Отчетность данного приложения сильно не продумана и не доработана (см. рис. 1.12.). На экране отчета «REPORTS» находится сумма всех записей за месяц. В ячейке списка отображается месяц и год записи, сумма пополнений и сумма затрат за период данного месяца. Изменять периоды отображения записей не представляется возможным, что вносит большие неудобства в работу с данной программой.

4) Так же как и в предыдущем приложении работа с фильтрацией и поиском записей отсутствует полностью. Единственная возможность представленная в данном приложении это сортировка записей исходя из

затрат или пополнения. Так на главном экране (см. рис. 1.10.) есть возможность выбирать временной период за который будут выбраны записи.

Проанализировав данное приложение становится понятным что приложение сильно не доделано и не выполнят большинства требований которые присущи финансовым программам. Так же большое количество рекламы в приложении говорит о том что приложение задумывалось в первую очередь не для помощи в ведении финансовой отчетности.

Приложение 3. «Smart Finance»

Последним рассмотренным приложением будет «Smart Finance», от разработчика «Alexander Survillo», скачанное так же из «AppStore».

Рис. 1.13. Главный экран приложения.

Приложение имеет довольно обычный интерфейс, без продуманного дизайна. Привыкнуть к приложению довольно сложно из-за его не ориентированного под пользователей интерфейса.

1) На главном экране (см. рис. 1.13.) приложения расположен список со всеми затратами за месяц. В данном приложении нет возможности просмотреть статистику за какой либо другой месяц кроме нынешнего, без покупки полной версии. В ячейках списка затрат отображаются сумма вышей траты, тип и дата которую выбирали на экране добавления записи о расходах (см. рис. 1.14.).

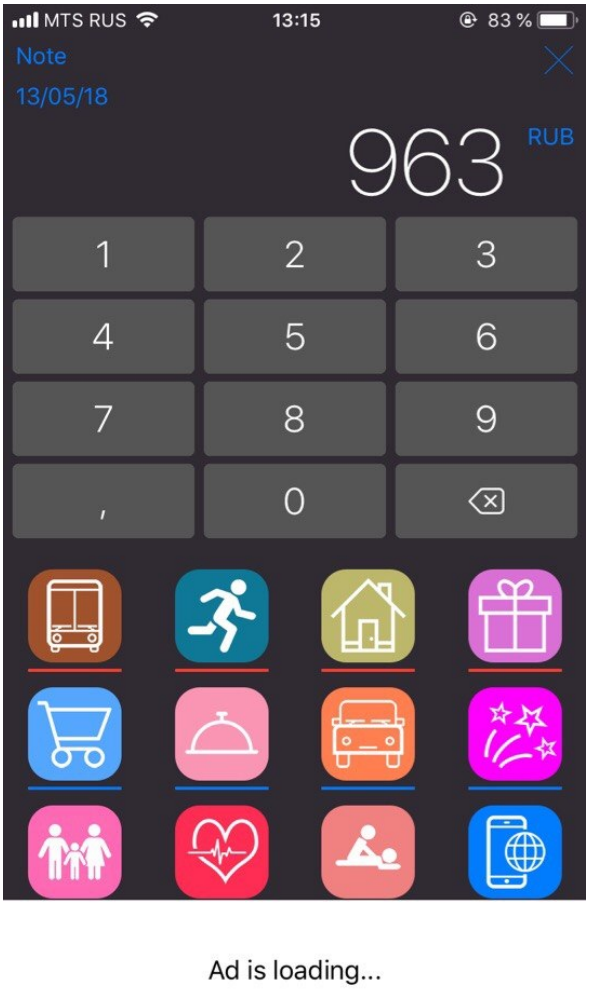

Рис. 1.14. Экран добавления записи о расходах.

2) На экране добавления записи о расходах находится числовая клавиатура с иконками типа товаров (см. рис. 1.14.). Для того чтобы добавить запись нужно ввести сумму вашей траты и выбрать ячейку типа товара, есть возможность изменять дату покупки, для этого нужно нажать на неприметную копку сегодняшней даты в верхней левой части экрана.

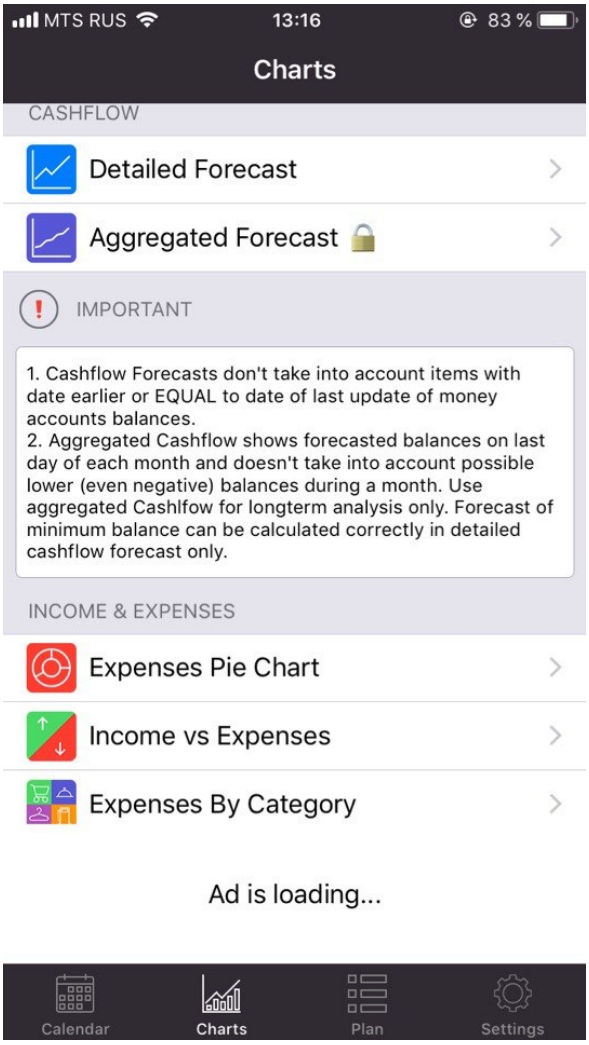

Рис. 1.15. Меню отображения данных о расходах.

3) Ведение финансовой отчетности в данном приложении разбито на десять категорий (см. рис. 1.15.). Многие из этих категорий предоставляют по большей степени не нужную информацию, которую можно было объединить на одном экране а не распределять по разным

категориям. Меню выбора типа отчетности выглядит очень сложно и имеет не погружаемый рекламный блок.

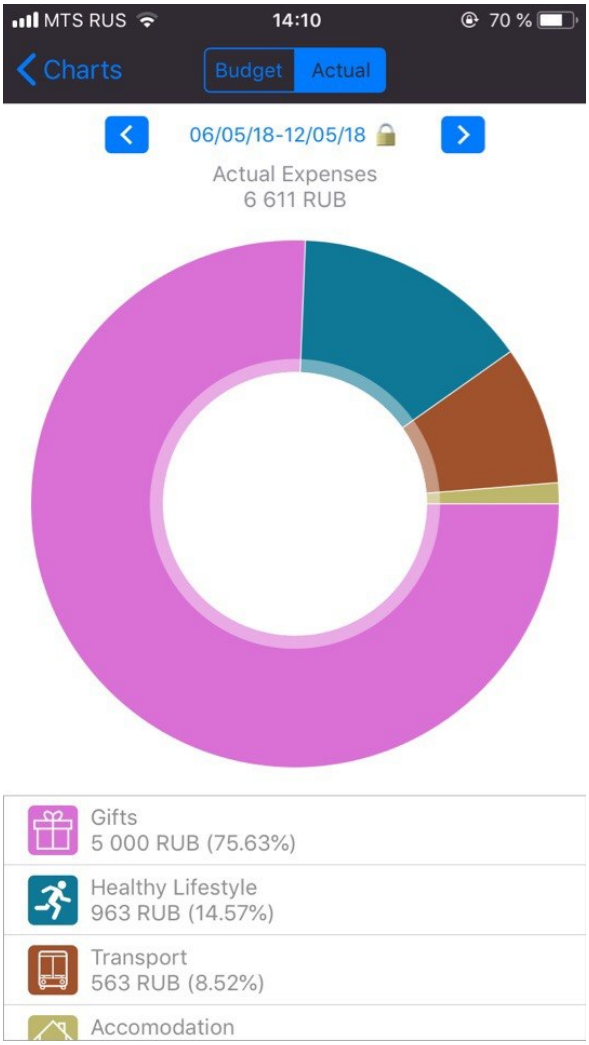

Рис. 1.16. Экран отчетности за неделю.

4) Экран отчетности приложения представлен в виде круговой диаграммы и списка объединённых типов затрат (см. рис. 1.16.). Все покупки за неделю объединяются в обобщённые типы и эти объединенные типы служат данными для построения круговой диаграммы. Типы покупок объединяются в непонятные по смыслу группы, при чем просмотреть из чего состоит группа посмотреть не предоставляется возможным. Отчет отображается только за нынешнюю неделю и изменить данный временной период можно только после покупки полной версии приложения. В ячейках списка отображается

название объединений группы, сумма затрат всех записей входящих в нее и процентное соотношение с другими группами. Фильтрация и какая либо сортировка записей невозможна (см. рис. 1.16.).

Рассмотрев данное приложение, с уверенностью можно сказать что приложение не доделано, вероятно разработчики хотели заложить большой функционал но не продумали его надобность и удобство. Со всем этим идет большое навязывание покупки полной версии приложения.

После рассмотрения нескольких приложения, можно приди к выводу, что все приложения не имеют полного функционала что бы отвечать всем потребностям пользователей, обладают очень большим навязыванием рекламы или покупки полной версии своего приложения, обладают не нужными большинству пользователей функциями и совсем не имеют практичного и удобного интерфейса. Анализ данных приложений основывался на нескольких главных критериях которыми должно обладать данное приложение. Для большей наглядности результаты анализа приведены в таблице 1.1.

Таблица 1.1

Анализ существующих приложений

	Удобство и доступность интерфейса	Типизация данных	Полная отчетность	Сортировка записей	Дизайн
«Деньги ОК»	+	+	+/-	+/-	+
«Pocket Finance»	+/-	+	-	-	+/-
«Smart Finance»	-	+	-	-	-

Исходя из анализа, возникает потребность разработать собственное приложение, которое будет удовлетворять всем потребностям пользователей и критериям описанным выше.

1.3. Требования к приложению

Требования к функциональным характеристикам проектирования и разработки мобильного приложения учета расходов физических лиц.

1) Приложение обязательно должно выполнять перечисленные функции:

- предоставлять пользователю возможность ввода основных данных о проделанной ранее покупке. Ввод должен быть как автоматизированный с использованием камеры телефона так и ручной;
- предоставлять возможность типизации своей покупки. Покупка типизируется минимум двумя параметрами: магазин и тип покупки;
- предоставлять полный отчет о потраченных средствах за любой временной период. Отчет можно составить по магазинам и типам товара в убывающем и возрастающем порядке по цене или по дате;
- реализовывать возможность продолжительного хранения данных о чеках с последующей возможностью использования данной информации в отчетах;
- реализовать возможность составления графиков, инфографики или диаграмм для более удобного и понятного восприятия данных отчетов о расходах;
- реализовать качественный, удобный и практичный во всех смыслах дизайн интерфейса. Разработка интуитивно понятного интерфейса на основе UX/UI дизайна.

2) Требования к организации входных данных

Входными данными в приложении являются те данные, которые пользователь вводит в процессе добавления записи о расходах с помощью ручного ввода – такие как:

- дата покупки;

- время покупки;
- сумма покупки.

Так же данные, которые попадают в приложение при добавлении записи о расходах с помощью сканирования QR кода на чека с использованием камеры смартфона – такие как:

- дата покупки;
- время покупки;
- сумма покупки;
- признак расчета (приход или расход, возврат);
- порядковый номер фискального документа;
- заводской номер фискального накопителя;
- фискальный признак документа.

Дополнительными входными данными являются критерии выбираемые из предоставляемого приложением списка – такие как:

- магазин в котором была сделана покупка;
- тип товара, который находится в чеке.

3) Требования к организации выходных данных

Вывод данных в приложении осуществляется путем вывода требуемой информации на экран смартфона. Экраны с данными делятся по запрошенной пользователем и типу выводимой информации.

На главном экране выводятся несколько последних добавленных чеков, которые представляют собой ячейку списка в которой отображены данные;

- картинка магазина в котором совершалась покупка;
- типы купленного товара;
- дата покупки;
- сумма покупки;
- тип ввода данных.

На экране просмотра всех добавленных чеков, выводится информация о всех чеках с возможностью фильтрации по множеству критериям, данные состоят из:

- картинка магазина в котором совершалась покупка;
- типы купленного товара;
- дата покупки;
- сумма покупки;
- тип ввода данных.

4) Требования к временным характеристикам

Требования к временным характеристикам не предъявляются, т.к. работа приложения не зависит от подключения к стороннему серверу, а выполняется исключительно на устройстве пользователя.

5) Требования к надежности

Надежная работа приложения всецело обеспечивается операционной системой устройств и языком приложения.

6) Отказы из-за некорректных действий пользователя

Отказы выполнения приложения возможны в случаях некорректных действий пользователя при взаимодействии с приложением. Возможностей добиться отказа выполнения приложения существует очень мало и во избежание этого были приняты меры защиты и предупреждения пользователя.

7) Климатические условия эксплуатации

Климатические эксплуатационные условия работы приложения, в которых оно обязано обеспечивать данные характеристики, должны соответствовать условиям использования и требованиям к эксплуатации которые предъявляются к смартфонам и другим техническим средствам на которых будет использоваться приложение.

8) Требования к параметрам технических средств

Для полноценной работы приложения достаточно устройства на операционной системе «iOS 10.0»

9) Требования к информационным структурам

Интерфейс взаимодействия пользователя должен быть полностью понятным как интуитивно так и логически, должен использовать методы «UX/UI» дизайна, работа с приложением не должна занимать много времени.

10) Требования к исходным кодам и языкам программирования

Основным языком программирования приложения выступает «Swift» с использованием «Cocoa» и «Cocoa Touch Framework». Интегрированной средой разработки приложения используется «XCode 10».

11) Требования к программным средствам, используемым приложением

Для работы приложения, у пользователя должно быть устройство поддерживающее работу операционной системы «iOS 10.0» и выше.

12) Требования к защите информации и программ

В Системе должен быть обеспечен надлежащий уровень защиты информации в соответствии с законом о защите персональной информации и программного комплекса в целом от несанкционированного доступа - “ Об информации, информатизации и защите информации” РФ N 24-ФЗ от 20.02.95.

13) Специальные требования

Приложение должно обеспечивать полное взаимодействие с пользователем посредством графического пользовательского интерфейса. Приложение должно обеспечивать полный контроль над вводимой информацией, удобный просмотр необходимой информации.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

2.1. Инфологическое проектирование

Концептуальное (инфологическое) проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Ориентированная на человека и не зависимая от типа СУБД модель предметной области, определяющая совокупности информационных объектов, их атрибутов и отношений между объектами, динамику изменений предметной области, а также характер информационных потребностей пользователей. Инфологическая модель предметной области может быть описана моделью "сущность-связь" (моделью Чена), в основе которой лежит деление реального мира на отдельные различимые сущности, находящиеся в определенных связях друг с другом, причем обе категории — сущность и связь полагаются первичными, неопределенными понятиями.

Целью инфологического моделирования является – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическую модель данных пытаются строить по аналогии с естественным языком (последний не может быть использован в чистом виде из-за сложности компьютерной обработки текстов и неоднозначности любого естественного языка). Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).[3]

Анализ будущего приложения и его концепции позволяет выделить сущности проектируемой базы данных и приняв во внимание что используется реляционная база данных, построить ее инфологическую модель на языке «Таблицы-связи»

1) Чек (Код_чека, Цена_чека, Дата_покупки, Признак_расчета, Номер_фиксального_документа, Заводской_номер_фиксального_накопителя, Фиксальный_признак_документа).

Эта сущность отводится для хранения главной информации о чеке, при помощи которой в будущем будут загружаться данные по каждому пункту покупок в чеке. Данная сущность добавляет несколько основных атрибутов, которые будут учувствовать в фильтрации всего стека чеков. Что бы была возможность ссылаться на чек, добавлен целочисленный атрибут «Код_чека», который будет автоматически наращиваться на единицу при добавлении в базу данных нового чека.

2) Магазин (Код_магазина, Название_магазина, Логотип_магазина).

Сущность магазин добавляется для возможности добавления дополнительной характеристики чека. Для каждого чека будет выделен магазин из заранее созданного списка магазинов со своим названием и логотипом в виде картинки. Так же данная сущность добавит возможность дополнительной фильтрации. Так как в большинстве чеков будет встречаться упоминание про магазин в котором была совершена покупка, то их стало целесообразно нумеровать и в чеках ссылается на эти номера. Для этого вводится целочисленный атрибут «Код_магазина», который будет автоматически наращивать единицу при добавлении нового магазина в базу данных.

3) Тип товара (Код_типа_товара, Название_типа_товара, Логотип_типа_товара).

Данная сущность вводится для хранения всех возможных типов товаров, наиболее часто встречающихся в числе покупок пользователя. Каждому чеку будут выделены несколько типов товара которые присутствуют в чеке. Пользователь сам будет выделять типы товара которые будет сохранены в чеке. Так же пользователю будет доступно добавление новых типов товара на свое усмотрение, каждый со своим названием и логотипом в виде картинки. Добавление сущности «Тип товара» добавит дополнительную возможность

фильтрации и будет одним из основных критериев для составления отчетов. Для того что бы было удобно ссылаться на тот или иной тип товара, был добавлен целочисленный атрибут «Код_типа_товара».

4) Свойства чека (Код_свойства_чека, Код_чека, Тип_ввода_данных, Дата_создания, Код_магазина).

Эта сущность выделяться для хранения всех дополнительных свойств добавленных чеков. Данная сущность имеет цель описания и уточнения сущности «Чек». Атрибуты данной сущность добавят возможность выделить дополнительные критерии для фильтрации. У каждого свойства чека будет целочисленный атрибут «Код_свойства_чека» на который будут ссылаться другие сущности.

5) Тип_товара_в_чеке (Код_свойства_чека, Код_типа_товара).

Данная сущность хранит в себе множество типов товаров записанных в тот или иной чек. Данные в атрибутах будут целочисленными кодами «Код_свойства»чека» и «Код_типа_товара», на один чек будет возможно добавить до пяти типов товаров. Эта сущность дополняет сущность «Свойства чека», данными о типах товарах в чеке на который ссылается атрибут «Код_чека».

Одним из главных критериев концептуального проектирования базы данных, является «ER-моделирование». Данная модель позволяет описывать концептуальные схемы предметной области. «ER-моделирование» используется при высокоуровневом (концептуальном) проектировании базы данных, с ее помощью можно выделить основные сущности и обозначить связи, которые могут устанавливаться между этими сущностями.[5]

Для базы данных была построена модель при помощи языка «ER-диаграмм». Данная модель ясно выделяет сущности с основными атрибутами, так же показаны связи между сущностями. Инфологическая модель изображена на рисунке 2.1.

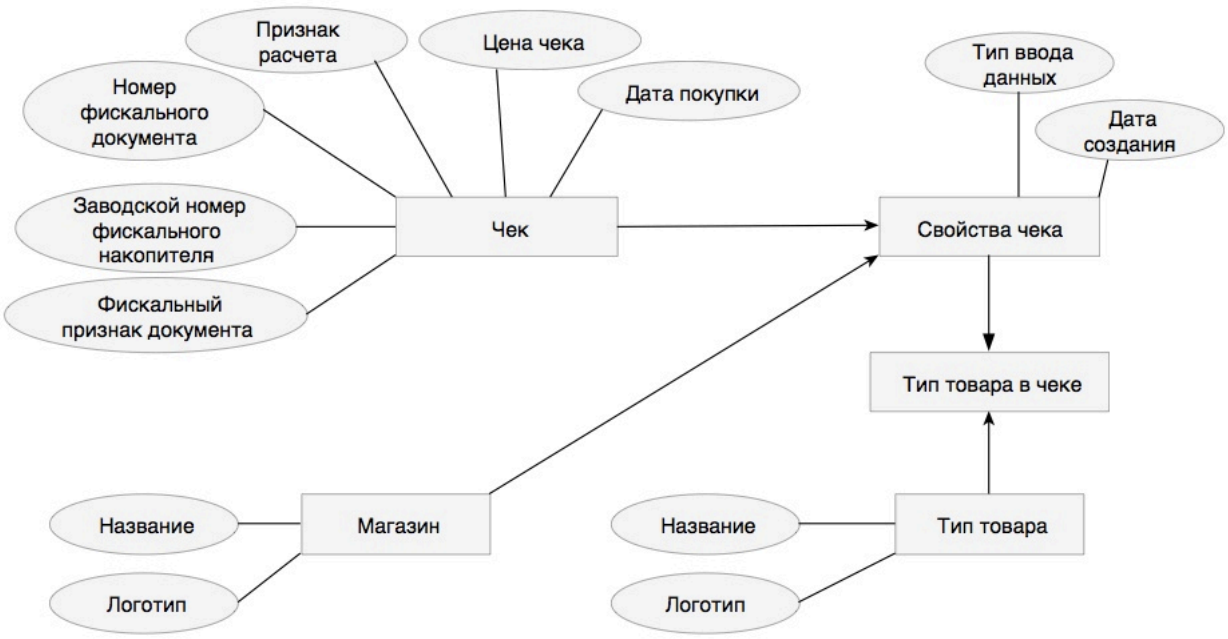

Рис. 2.1. Инфологическая модель, построенная с помощью «ER-диаграмм».

2.2. Даталогическое проектирование

Логическое (даталогическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных даталогическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

Исходными данными для даталогического проектирования является инфологическая модель предметной области.

На этапе логического проектирования разрабатывается логическая структура базы данных, соответствующая логической модели. Решение этой задачи существенно зависит от модели данных, поддерживаемой выбранной СУБД.[2]

База данных создаётся на основании схемы базы данных. Инфологическую модель данных, построенную в виде ER-диаграммы или «Таблицы-связи», следует преобразовать в схему базы данных. Преобразование схемы «Таблицы-связи» в схему базы данных выполняется путем сопоставления каждой сущности и каждой связи, имеющей атрибуты, в таблицы-отношения.[6]

Для этого необходимо выполнить следующие шаги процедуры проектирования модели.

1. Представить каждый стержень (независимую сущность) таблицей базы данных (базовой таблицей) и специфицировать первичный ключ этой базовой таблицы.
2. Представить каждую ассоциацию (связь вида «многие-ко-многим» или «многие-ко-многим-ко-многим» и т.д. между сущностями) как базовую таблицу. Использовать в этой таблице внешние ключи для идентификации участников ассоциации и специфицировать ограничения, связанные с каждым из этих внешних ключей.
3. Представить каждую характеристику как базовую таблицу с внешним ключом идентифицирующим сущность, описываемую этой характеристикой. Специфицировать ограничения на внешний ключ этой таблицы и её первичный ключ – по всей вероятности, комбинации этого внешнего ключа и свойства, которое гарантирует «уникальность в рамках описываемой сущности».
4. Представить каждое обозначение, которое не рассматривалось в предыдущем пункте, как базовую таблицу с внешнем ключом, идентифицирующим обозначаемую сущность. Специфицировать связанные с каждым таким внешним ключом ограничения.
5. Представить каждое свойство как поле в базовой таблице, представляющей сущность, которая непосредственно описывается этим свойством.

6. Для исключения из проекта непреднамеренных нарушений каких-либо принципов нормализации, выполнить процедуры нормализации.

7. Если в процессе нормализации было произведено разделение каких-либо таблиц, то следует модифицировать инфологическую модель базы данных и повторить перечисленные шаги.

8. Указать ограничения целостности проектируемой базы данных и дать (при необходимости) краткое описание полученных таблиц и их полей.[5]

Таблица 2.1

Схема отношения Чек «Check»

Содержание поля	Имя поля	Тип, длина	Примечание
Код чека	id_check	Int(11)	Первичный ключ
Цена чека	check_price	Double(10)	Обязательное поле
Дата покупки	check_date	Date	Обязательное поле
Признак расчета	i	Int(1)	
Номер фискального документа	n	VarChar(10)	
Заводской номер фискального накопителя	fn	VarChar(10)	
Фискальный признак документа	fp	VarChar(10)	

Таблица 2.2

Схема отношения Магазин «Shop»

Содержание поля	Имя поля	Тип, длина	Примечание
Код магазина	id_shop	Int(11)	Первичный ключ
Название магазина	shop_name	VarChar(50)	Обязательное поле
Логотип магазина	shop_logo	VarChar(50)	

Таблица 2.3

Схема отношения Тип товара «ProductType»

Содержание поля	Имя поля	Тип, длина	Примечание
Код типа товара	id_product_type	Int(11)	Первичный ключ
Название типа товара	type_name	VarChar(50)	Обязательное поле
Логотип типа товара	type_logo	VarChar(50)	

Таблица 2.4

Схема отношения Свойства чека «CheckProperties»

Содержание поля	Имя поля	Тип, длина	Примечание
Код свойства чека	id_check_properties	Int(11)	Первичный ключ
Код чека	id_check	Int(11)	Внешний ключ (к Check)
Тип ввода данных	input_type	Int(1)	Обязательное поле
Дата создания	create_date	Date	Обязательное поле
Код магазина	id_shop	Int(11)	Внешний ключ (к Shop)

Таблица 2.5

Схема отношения Типы товара в чеке «TypeInCheck»

Содержание поля	Имя поля	Тип, длина	Примечание
Код свойства чека	id_check_properties	Int(11)	Внешний ключ (к CheckProperties)
Код типа товара	id_product_type	Int(11)	Внешний ключ (к ProductType)

Для представленной ранее инфологической модели (см. рис. 2.1.) была разработана даталогическая модель базы данных изображенная на рисунке 2.2.

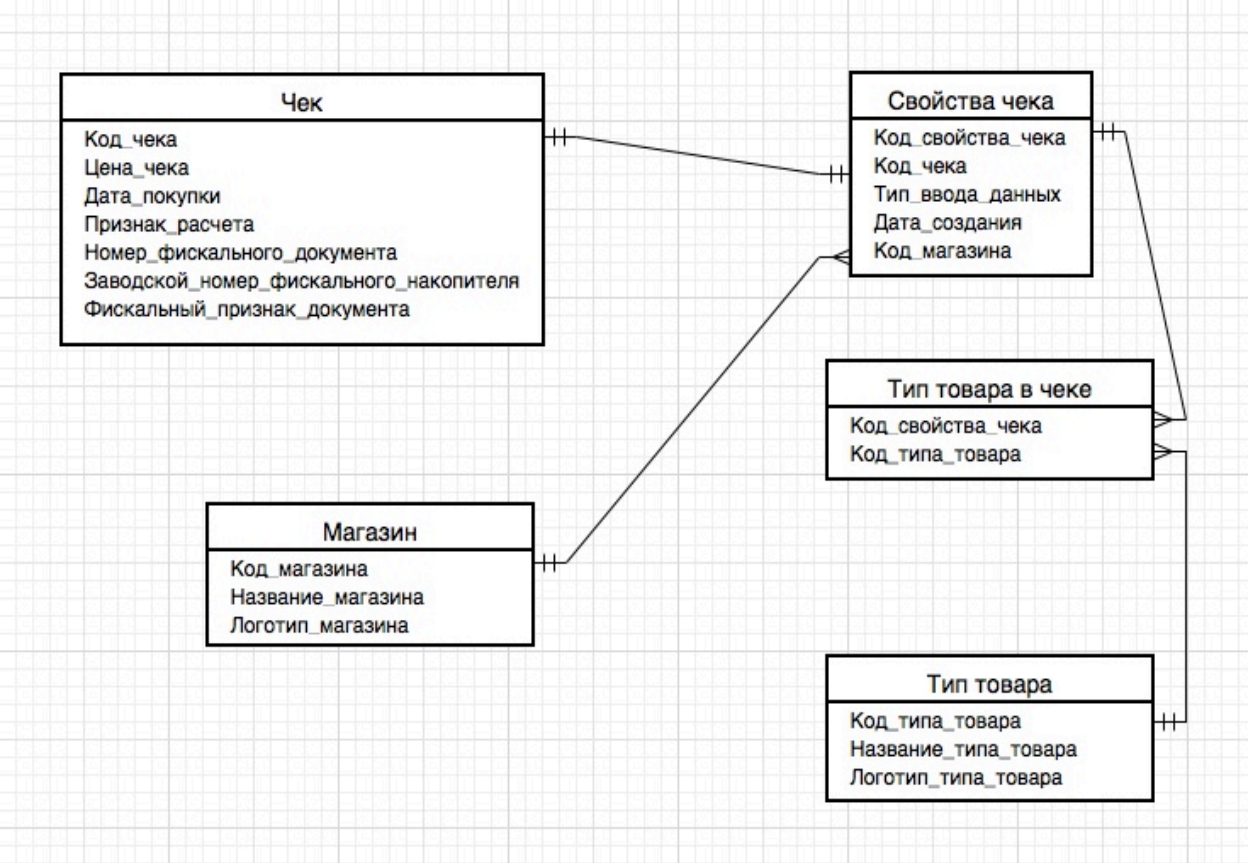

Рис. 2.2. Даталогическая модель базы данных.

Таким образом было выполнено логическое проектирование базы данных приложения, выполнена нормализация, построенные отношения между всеми сущностями, что дает возможность полностью исключить избыточность информации и противоречивость хранимых данных.

ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

3.1. Выбор средств разработки

Существует много вариантов выбора средств разработки, с помощью которых можно реализовать приложение финансовой аналитики, которые смогут обеспечить автоматизированность и динамичность приложения с использованием среды разработки визуальной части. Основываясь на множестве критериев, для реализации приложения были выбраны открытый мультипарадигмальный компилируемый язык программирования «Swift» и интегрированная среда разработки программного обеспечения «Xcode».[10] Эти средства разработки были выбраны сходя из нескольких основных качеств:

- скорость работы;
- интегрированная среда;
- встроенный эмулятор;
- поддержка разработки под «iOS» и «macOS» платформы;

Swift – новый язык программирования созданный в 2014 году компанией «Apple» в первую очередь для разработки приложений для «iOS», «macOSX», «watchOS» и «tvOS». Язык является открытым, что дает возможность разработчикам писать на нем как приложения так и операционные системы. Данный язык программирования задумывался как более легкий для чтения и устойчивый к ошибкам человеческого фактора в сравнении с предшествующем ему строгим «Objective-C». «Swift» работает с фреймворками «Cocoa» и «Cocoa Touch», используются паттерны безопасного программирования так же добавлены современные функции, превращающие создание приложения в простой, более гибкий и увлекательный процесс. Данный язык компилируется при помощи «Low Level Virtual Machine», входящей в среду разработки, так же он поддерживает рантайм, что делает

возможным использование «Swift», «Objective-C» а также «C» в рамках одного приложения, что является большим преимуществом для разработчиков.[7][8]

«Xcode» - интегрированная среда разработки (IDE) программного обеспечения для всех платформ «OSX» разработанная корпорацией «Apple». Первая версия продукта была выпущена в 2001 г. Стабильные версии распространяются бесплатно через «Mac App Store».[11] Зарегистрированные разработчики также имеют доступ к бета-сборкам через сайт «Apple Developer». «Xcode» включает в себя большую часть документации разработчика от «Apple» и «Interface Builder» — приложение, использующееся для создания графических интерфейсов. Пакет «Xcode» включает в себя изменённую версию свободного набора компиляторов «GNU Compiler Collection» и поддерживает языки «C», «C++», «Objective-C», «Objective-C++», «Swift», «Java», «AppleScript», «Python» и «Ruby» с различными моделями программирования, включая (но не ограничиваясь) «Cocoa», «Carbon» и «Java».[9] Сторонними разработчиками реализована поддержка «GNU Pascal», «Free Pascal», «Ada», «C#», «Perl», «Haskel»l и «D». Пакет «Xcode» использует «GDB» в качестве back-end'а для своего отладчика. Данная среда разработки является основной для языка «Swift» и имеет весть спектр возможностей для удобной разработки приложений.

Дополнительным критерием для выбора данных средств послужило наличие удобного механизма разработки интерфейса программы – «Storyboard». Данный механизм позволяет заметно облегчить создание объектов интерфейса приложения путем переноса нужного объекта на экран будущего приложения и обращения к нему в классе программы, что дает возможность заметно уменьшить количество кода связанного с переходами между экранами, показами «Popover», кнопок, ячеек и настройкой функциональности ячеек в таблице.[12]

3.2. Программная реализация приложения

Для написания всех модулей и экранов данного использовался большой объем кода, что не дает возможности описать код всего приложения полностью. Будут представлены основные модули разработанного кода которые выполняют основную функциональную задачу. В требования к приложению были заявлены основные функции реализующие сканирование QR кода с чека, создание типов товара и магазинов, добавление чеков, отображение чеков в удобном и читаемом виде пользователю. Рассмотрим функцию реализующую сканирование QR кода с помощью камеры устройства (листинг 3.1).

Листинг 3.1. Сканирование QR кода.

```
let session = AVCaptureSession()
 guard let captureDevice = AVCaptureDevice.default(for:   AVMediaType.video)
else { return }
 do {
    let input = try AVCaptureDeviceInput(device: captureDevice)
    session.addInput(input)
  } catch {
    print("ERROR")
  }
  let output = AVCaptureMetadataOutput()
  session.addOutput(output)
  output.setMetadataObjectsDelegate(self, queue: DispatchQueue.main)
  output.metadataObjectTypes = [AVMetadataObject.ObjectType.qr]
  video = AVCaptureVideoPreviewLayer(session: session)
  video.videoGravity = AVLayerVideoGravity.resizeAspectFill
  viewQRContainer.layer.addSublayer(video)
  session.startRunning()
```

Конец листинга.

В данном листинге мы для начала создаем переменную «session» указываем тип хранения данных «AVCaptureSession», дальше создаем проверяемую переменную «captureDevice» в которой будет хранится используемое оборудование, а именно камера типа видео. Следующим шагом является попытка вызвать обработчик событий при помощи камеры устройства и записать в переменную «input». Дальше в переменную «output» записываем обработчик выходных данных, добавляем последний к раннее созданной сессии и добавляем очередь выходных данных. Определяем тип данных которые будет принимать обработчик выходных данных, присеваем тип «qr». Последними действиями идут создание переменной «video» в нее передаем обработчик вывода картинки с камеры на экран приложения, присваиваем обработчик «video» раннее созданному контейнеру «viewQRContainer» и запускаем сессию обработки «QR» кода с камеры.

Рассмотрим фрагмент кода выполняющий функцию отображения и заполнения таблицы с созданными пользователем магазинами (листинг 3.2)

Листинг 3.2. Вывод данных в режиме динамической таблицы.

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
return shops.count
   }
   override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
      let cell = tableView.dequeueReusableCell(withIdentifier: "cell") as!
ShopCustomTableCell
      let shop = shops[indexPath.row]

      let testDate = UserDefaults.standard.object(forKey: (shop.value(forKey:
"shop_logo") as! String)) as! NSData
cell.shopImage.image = UIImage(data: testDate as Data)
```

```
        cell.shopLabel.text = shop.value(forKey: "shop_name") as? String
        return cell
```

Конец листинга.

Первая функция «tableView» принимающая «numberOfRowInSection» вызывается для отображения определенного количества строк данной таблицы, в данном случае функция возвращает значение «shops.count», количество объектов в ранее созданной переменной «shops».

Вторая функция «tableView» принимающая «cellForRowAt» выполняет функцию циклического заполнения ранее определенного числа строк таблицы. Изначально мы инициализируем опциональную переменную «cell» в которую записываем обработчик поочередного вызова ячеек с идентификатором «withIdentifier» равным «cell», вызов происходит с помощью созданного класса «ShopCustomTableCell» в котором мы создали интерфейс представления для ячеек данной таблицы. Потом инициализируем опциональную переменную «shop» хранящую объект массива «shops» с индексом «indexPath.row» равным индексу данной строки таблицы. Следующим шагом мы создаем объект класса «UserDefault» который в свою очередь обращается в локальное хранилище приложение и вызывает объект с названием «shop.value(forKey: "shop_logo")» что в свою очередь является записью из базы данных с названием загруженного логотипа магазина, вызов из базы данных происходит как объект типа «String», затем происходит представление объекта как «NSData». Дальше происходит запись ранее созданной ссылки на логотип в контейнер изображений «imageView» приписанный в классе представлений «cell», запись происходит обязательно с типом «UIImage». Таким же образом присваивается и название логотипа «shop.value(forKey: "shop_name")» из базы данных как тип «String» в переменную «shopLabel» и идентификатор «text» из класса представлений «cell». В самом конце возвращается объект «cell» с полным представлением ячейки.

Участок кода отвечающий за динамическое появление окна с выбором типа добавления логотипа в создаваемый пользователем магазин (листинг 3.3).

Листинг 3.3. «Alert» окно с кнопками выбора.

```
let imagePickerController = UIImagePickerController()
imagePickerController.delegate = self
let actionAlert = UIAlertController(title: "Выберите действие", message: nil,
preferredStyle: .actionSheet)
actionAlert.addAction(UIAlertAction(title: "Камера", style: .default, handler:
{(action:UIAlertAction) in
   imagePickerController.sourceType = .camera
   self.present(imagePickerController, animated: true, completion: nil)
}))
actionAlert.addAction(UIAlertAction(title: "Галерея", style: .default, handler:
{(action:UIAlertAction) in
    imagePickerController.sourceType = .photoLibrary
    self.present(imagePickerController, animated: true, completion: nil)
}))
actionAlert.addAction(UIAlertAction(title: "Отмена", style: .cancel, handler: nil))
self.present(actionAlert, animated: true, completion: nil)
```

Конец листинга.

В начале листинга мы создаем опциональную переменную «imagePickerController» которая хранит в себе обработчик «UIImagePickerController» который дает возможность совершать манипуляции с изображениями. Следующим шагом является создание опциональной переменной «actionAlert» типа «UIAlertController» которая позволяет создавать окна оповещений, добавляем кнопку с названием «Камера», к переменной «imagePickerController» добавляем обрабатываемый

тип «Camera», и запускаем презентацию метода «imagePickerController». Таким же образом добавляем кнопку с названием «Галерея» и присваиваем переменной «imagePickerController» тип «photoLibrary» создаем презентацию. Следующим шагом создаем действие кнопку с именем «Отмена», для отмены.

Функция добавления выбранного изображения в контейнер на экране для отображения пользователю представлена в листинге 3.4.

Листинг 3.4. Функция отображения выбранного изображения.

```
func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [String : Any]) {
   let image = info[UIImagePickerControllerOriginalImage] as! UIImage
   self.ImageControllerOutlet.image = image
      picker.dismiss(animated: true, completion: nil)
}
```

Конец листинга.

Функция «imagePickerController» с входным типом «didFinishPickingMediaWithInfo» реализует использование изображения после его выбора в обработчике выбора изображений «imagePickerController». Дальше происходит создание опциональной переменной «image» которая содержит метод «info» с типом «UIImagePickerControllerOriginalImage» который возвращает ссылку на выбранной изображение. Таким образом предыдущая строчка реализует метод окончательно выбора изображения из галереи или предварительной съемки с помощью камеры устройства. Добавление ранее выбранного изображения хранящегося в переменной «image» в контейнер изображений «ImageControllerOutlet» через идентификатор типа «image». В конце происходит завершение обработчика «ImagePickerController» созданного в (листинге 3.3.), через класс «picker» и метод «dissmiss».

ГЛАВА 4. ИСПЫТАНИЯ

4.1. Программа и методика испытаний

Рассматриваемым в данной главе объектом, является ранее разработанное «мобильное приложение учета расходов физических лиц».

Целью испытаний приложения являются проверка его работоспособности, а так же полное соответствие требованиям описанным в первой главе.

Полный перечень требований к автоматизированной системе приведен в третьем разделе первой главы, однако ключевыми пунктами, требующими установки соответствия, являются следующие:

- предоставлять пользователю возможность ввода основных данных о проделанной ранее покупке. Ввод должен быть как автоматизированный с использованием камеры телефона так и ручной;
- предоставлять возможность типизации своей покупки. Покупка типизируется минимум двумя параметрами: магазин и тип покупки;
- предоставлять полный отчет о потраченных средствах за любой временной период. Отчет можно составить по магазинам и типам товара в убывающем и возрастающем порядке по цене или по дате;
- реализовывать возможность продолжительного хранения данных о чеках с последующей возможностью использования данной информации в отчетах;
- реализовать возможность составления графиков, инфографики или диаграмм для более удобного и понятного восприятия данных отчетов о расходах;

Исходя и приведенных выше требований – был разработан следующий алгоритм для проведения испытаний приложения.

1. Тестирование ввода данных.

1.1. Нажать на кнопку «Добавить чек» расположенную в нижней левой части экрана.

1.2. Убедится в наличии возможности сканирования «QR» кода с чека.

1.3. Отсканировать «QR» код с чека.

1.4. Нажать на кнопку «Ручной ввод» в верхней правой части экрана.

1.5. Ввести данные о чеке в ручную.

1.6. Нажать на кнопку «Магазин» в центре экрана.

1.7. Убедится в наличии списка созданных магазинов.

1.8. Выбрать магазин из предложенного списка.

1.9. Повторно выбрать магазин из предложенного списка.

1.10. Нажать на кнопку «Выбрать» в нижней левой части предложенного списка.

1.11. Убедится в корректности выбора магазина.

1.12. Нажать на копку «Типы товара» в нижней центральной части экрана.

1.13. Убедится в наличии списка типов товара созданных ранее.

1.14. Выбрать один или несколько типов товара из предложенного списка.

1.15. Выбрать несколько одинаковых типов из списка.

1.16. Выбрать больше пяти типов товаров.

1.17. Нажать на кнопку «Выбрать» в нижней части предложенного списка.

1.18. Убедится в корректности выбора типов товара.

1.19. Нажать на кнопку «Сохранить чек» в нижней части экрана.

1.20. Сохранить чек без введенных данных о цене чека.

1.21. Сохранить чек без введенных данных о дате чека.

1.22. Сохранить чек без введенных данных о времени чека.

1.23. Сохранить чек без выбора магазина.

1.24. Сохранить чек без выбора типов товара.

1.25. Нажать на кнопку «Магазины» в верхнем меню главного экрана.
1.26. Убедится в наличии экрана со списком созданных ранее магазинов.
1.27. Нажать на кнопку «Добавить магазин» в верхней части экрана.
1.28. Убедится в открытии экрана добавления магазина.
1.29. Нажать на кнопку «Добавить логотип» в центре экрана.
1.30. Убедится в открытии меню выбора типа действия.
1.31. Выбрать действие «Камера».
1.32. Убедится в отображении картинки с камеры устройства.
1.33. Сделать снимок, нажатием на круглую кнопку в нижней части экрана.
1.34. Убедится в создании фотографии.
1.35. Нажать на кнопку «Use Photo» в правой нижней части экрана.
1.36. Убедится в корректности отображении сделанной ранее фотографии на экране.
1.37. Нажать на кнопку «Добавить логотип» в центре экрана.
1.38. Выбрать действие «Галерея» в появившемся меню.
1.39. Убедится в отображении галереи устройства.
1.40. Выбрать фотографию из галереи.
1.41. Убедится в корректности отображения выбранного изображения на экране.
1.42. Нажать на текстовое поле «Название» в нижней части экрана.
1.43. Убедится в корректном отображении клавиатуры и удобстве ввода.
1.44. Ввести желаемое название магазина.
1.45. Нажать на кнопку «return» в нижней правой части клавиатуры.
1.46. Убедится в корректном отображении ранее введённого названия.
1.47. Нажать на кнопку «Добавить магазин» в нижней части экрана.
1.48. Убедится в переходе на предыдущий экран со списком созданных магазинов.
1.49. Нажать на кнопку «Добавить магазин» без выбранного логотипа.

1.50. Нажать на кнопку «Добавить магазин» без введенного названия магазина.
1.51. Убедится в появлении ранее созданного магазина в списке.
1.52. Убедится в корректности сохранения раннее выбранного логотипа и названия магазина.
1.53. Нажать на кнопку «Типы» в верхнем меню главного экрана.
1.54. Убедится в переходе на экран со списком ранее созданных типов товара.
1.55. Нажать на кнопку «Добавить тип товара» в верхней части экрана.
1.56. Убедится в переходе на экран добавления типа товара.
1.57. Убедится в отображения списка возможных иконок.
1.58. Нажать на любую иконку из списка.
1.59. Убедится в отображении выбранной иконки в нижней части экрана.
1.60. Повторно нажать на любую иконку из списка.
1.61. Повторно нажать на выбранную иконку их списка.
1.62. Нажать на текстовое поле «Название».
1.63. Убедится в корректном отображении клавиатуры и удобстве ввода.
1.64. Ввести название магазина.
1.65. Нажать на кнопку «return» в правой нижней части экрана.
1.66. Убедится в скрытии клавиатуры.
1.67. Убедится в корректности ранее введенного названия.
1.68. Нажать на кнопку «Добавить тип товара» в нижней части экрана.
1.69. Убедится в переходе на предыдущий экран со списком ранее созданных типов товара.
1.70. Нажать на кнопку «Добавить тип товара» без выбранной иконки.
1.71. Нажать на кнопку «Добавить тип товара» без введенных данных о названии типа товара.
1.72. Убедится в появлении ранее созданного типа товара в списке.
1.73. Убедится в корректном отображении ранее выбранных иконки и названии.

2. Тестирования отображения данных.

2.1. Убедится в корректности вывода десяти последних чеков на кланом экране.

2.2. Убедится в корректности отображения выбранного магазина для каждого чека.

2.3. Убедится в корректности и католичестве типов товара в каждом чеке на главном экране.

2.4. Убедится в Корректном отображении основных данных на чеке.

2.5. Убедится в корректности отображения иконки типа ввода информации чека.

2.6. Нажать на кнопку «Список/диаграмма» нижней правой части экрана.

2.7. Убедится в отображении списка всех чеков.

2.8. Убедится в корректности отображаемых данных.

2.9. Нажать на кнопку «Фильтр» в верхнем меню.

2.10. Выбрать тип отображения по возрастанию или по убыванию.

2.11. Нажать на кнопку «Дата».

2.12. Ввести дату в текстовое поле «От» и «До».

2.13. Нажать на кнопку «Применить».

2.14. Убедится в отображении чеков по заданным показателям.

2.15. Нажать на кнопку диаграмма в верхней части экрана.

2.16. Убедится в корректном отображении диаграммы типов товара.

2.17. Нажать на кнопку «Фильтр».

2.18. Нажать на кнопку «Цена».

2.19. Ввести цену в текстовое поле «От» и «До».

2.20. Нажать кнопку «Применить».

2.21. Убедится в корректном отображении чеков по заданным данным.

2.22. Нажать на кнопку «Диаграмма».

2.23. Убедится в корректном отображении диаграммы.

4.2. Тестирование приложения

С начала запускаем приложение, путем нажатия на иконку приложения с именем «InWallet» в меню вашего устройства. После того как приложение загрузилось, мы увидим главный экран с десятью последними добавленными чеками. Главный экран приложения изображен на рисунке 4.1.

Рис 4.1. Главный экран приложения.

Каждый из представленных чеков имеет обязательные для отображения поля:

- Картинка магазина.
- От одной до пяти иконок типов товара.

- Сумму чека.
- Дату чека.
- Тип добавления чека.

Для добавление нового чека, требуется нажать на кнопку «Добавить чек» в виде иконки чека со знаком плюс в нижней части экрана. Экран добавления чека на рисунке 4.2.

Рис. 4.2. Сканирование QR кода при добавлении чека

При переходе на экран добавления нового чека сразу же предлагается отсканировать «QR» код при помощи камеры устройства. Сканирование

происходит мгновенно по этому для примера был взят «QR» не с чека, таким образом можно увидеть предупреждение о не верности кода. Если у вас нет на руках чека с кодом, то можно нажать на кнопку «Ручной ввод» или отсканировать правильный «QR» код. Следующий шаг после сканирования «QR» кода показан на рисунке 4.3.

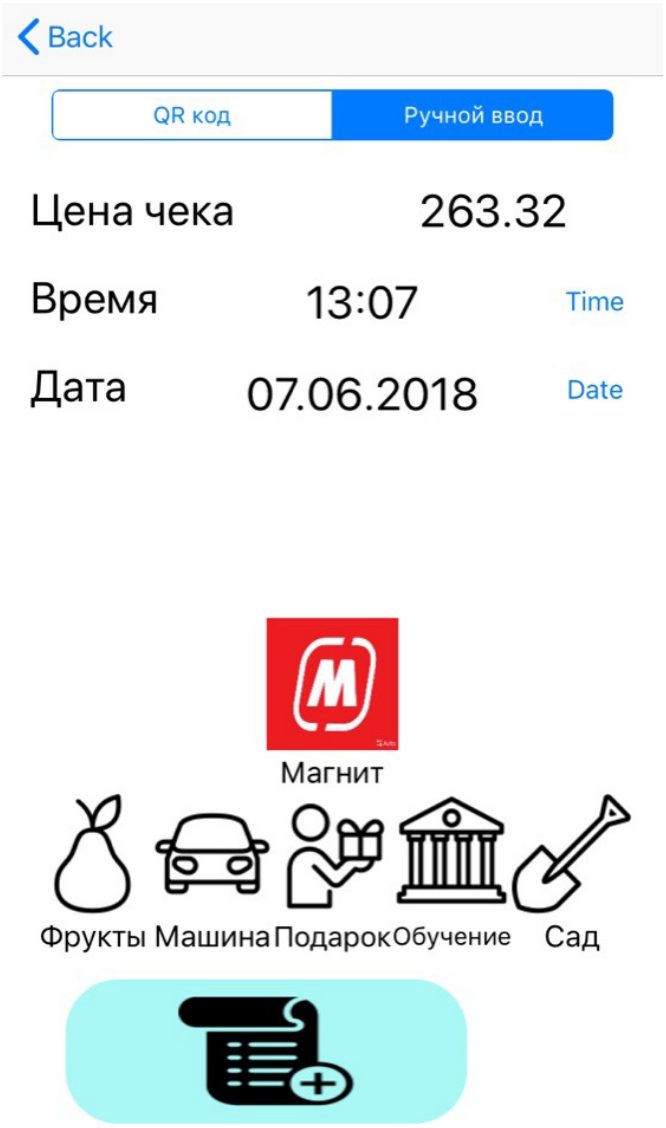

Рис. 4.3. Шаг после сканирования «QR» кода.

После верно отсканированного «QR» кода или при нажатии на кнопку «Ручной ввод» окно с изображением с камеры скрывается. Если был отсканирован код, то в полях «Цена чека», «Время», «Дата» появятся данные с вашего чека, если была нажата кнопка «Ручной ввод», то данные поля будут

пустыми. При добавлении чека существует 2 обязательных поля магазин и тип товара. При нажатии на кнопку добавления магазина появляется окно с ранее созданными магазинами. Ниже находится поля с типами товара. Обязательным является указать один тип товара, дополнительные четыре по желанию. Экран с ранее созданными магазинами изображен на рисунке 4.4.

Рис. 4.4. Созданные магазина.

При добавлении нового чека в обязательном порядке требуется указать»» магазин в котором была произведена покупка или получена услуга. Для перехода на данный экран требуется нажать на кнопку «Магазины» в верхнем меню главного экрана приложения. На данном экране отображены

все доступные магазины которые были ранее созданы. Для создания нового магазина нужно нажать на кнопку «Добавить магазин». Процесс добавления магазина показан на рисунке 4.5.

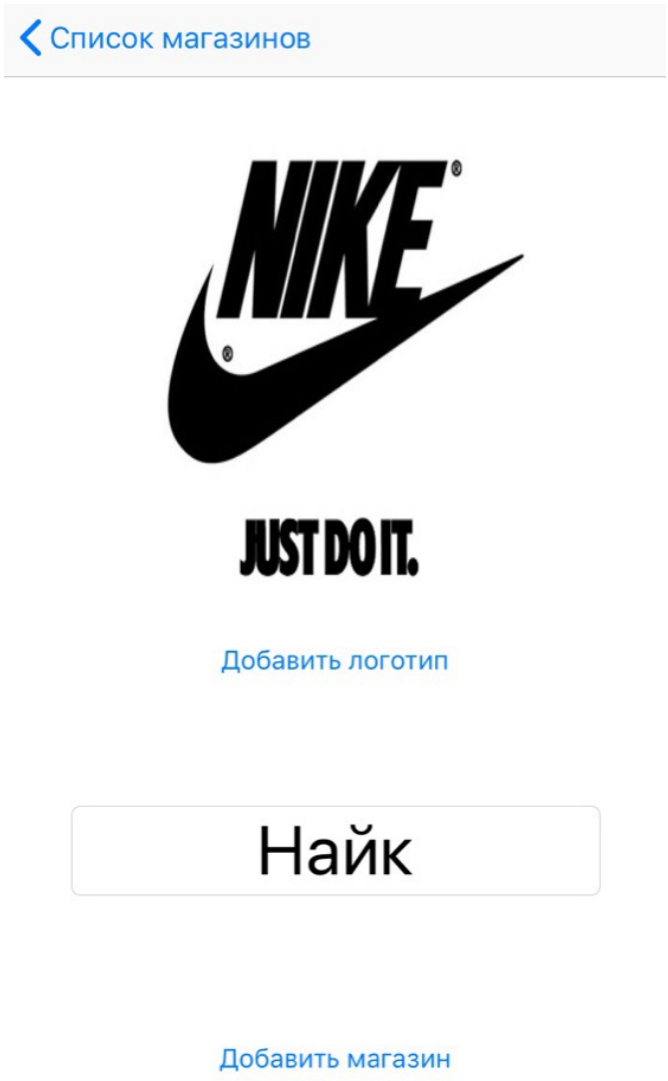

Рис. 4.5. Процесс создания магазина.

Для создания магазина обязательными являются логотип и название. Для добавления логотипа магазина есть два варианта добавления, снимок с камеры устройства или выбор из галереи. После ввода названия будущего магазина, нажимаем на кнопку «Добавить магазин». На экране отобразится список созданных магазинов изображенный на рисунке 4.5. с добавленным магазином который создавался. Если нажать на кнопку «Добавить магазин»

без выбранного логотипа или веденного названия, то на экране появится уведомление об ошибке у указанием поля которое было пропущено при заполнении. Экран с созданными типа товара изображен на рисунке 4.6.

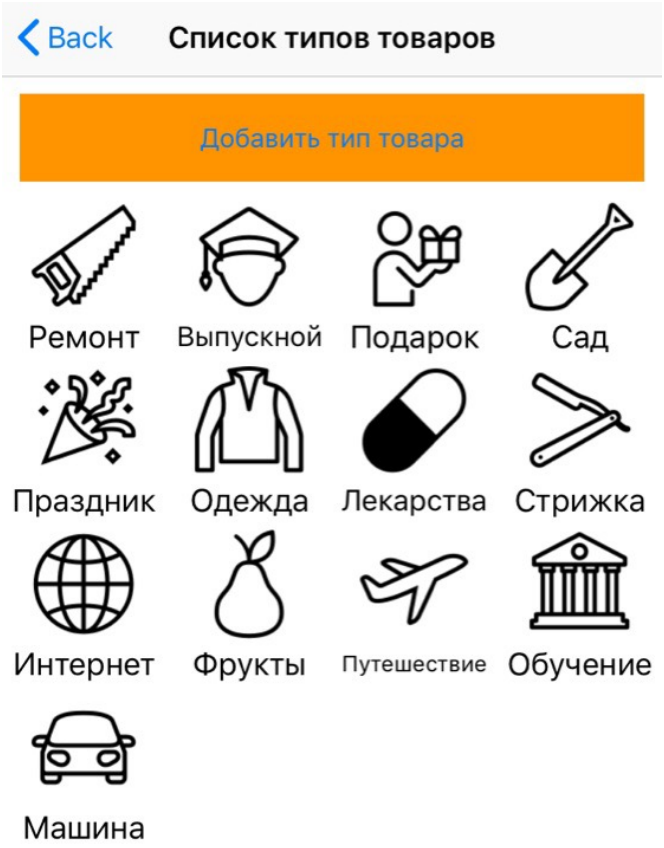

Рис. 4.6. Экран с ранее созданными типами товара.

Для создания чека обязательным условием так же является наличие хотя бы одного типа товара или услуги. Для того чтобы перейти к экрану со списком ранее созданных типов товара требуется нажать на кнопку «Типы» в верхнем меню главного экрана приложения. Для создания нового типа товара нужно нажать на кнопку «Добавить тип товара» в верхней части экрана со

списком типов товара. Добавление нового типа товара изображено на рисунке 4.7.

Рис. 4.7. Процесс создания тип товара.

Для создания типа товара требуется выбрать один из двух ста разнообразных иконок и ввести название будущего типа. Иконка и название являются обязательными критериями для создания типа товара. Для создания типа с выбранной иконкой и введенным названием нужно нажать на кнопку «Добавить тип товара» в нижней части экрана. Если нажать на кнопку «Добавить тип товара» без введенного названия или не выбранной заранее иконки, то на экране появится сообщение с ошибкой указывающей какое

именно из обязательных к заполнению полей является пустым. Экран со списком всех ранее созданных чеков изображен на рисунке 4.8.

Рис.4.8. Экран со списком всех ранее созданных чеков.

Для того чтобы перейти на экран со списком всех чеков, нужно нажать на кнопку «Список и диаграмма» на в нижней правой части главного экрана приложения. На данном экране мы видим список всех чеков отсортированных по по возрастанию относительно даты в чеке, главным отличием от главного экрана является количество отображаемых чеков. На данном экране есть возможность сортировки чеков по нескольким критериям. Для того что бы

отсортировать чеки нужно нажать на кнопку «Фильтр» в верхнем меню экрана. Окно фильтра изображено на рисунке 4.9.

Рис. 4.9. Окно фильтра.

В окне фильтра представлены кнопки для настройки фильтрации чеков из списка всех чеков изображённого на рисунке 4.8. Фильтр можно настраивать как в группе так и по отдельности. Существует возможность сортировки по возрастанию, убыванию. Фильтрация поддерживает условия «От» и «До», можно задать границы отображения чеков полученных в период между заданными датами или сумма которых входит в границы. Так же можно выбрать магазин из списка в котором вам дали чек. Для применения фильтра

нужно выбрать требуемые условия, задать границы и нажать кнопку «Применить». Результаты фильтрации чеков изображены на рисунке 4.10.

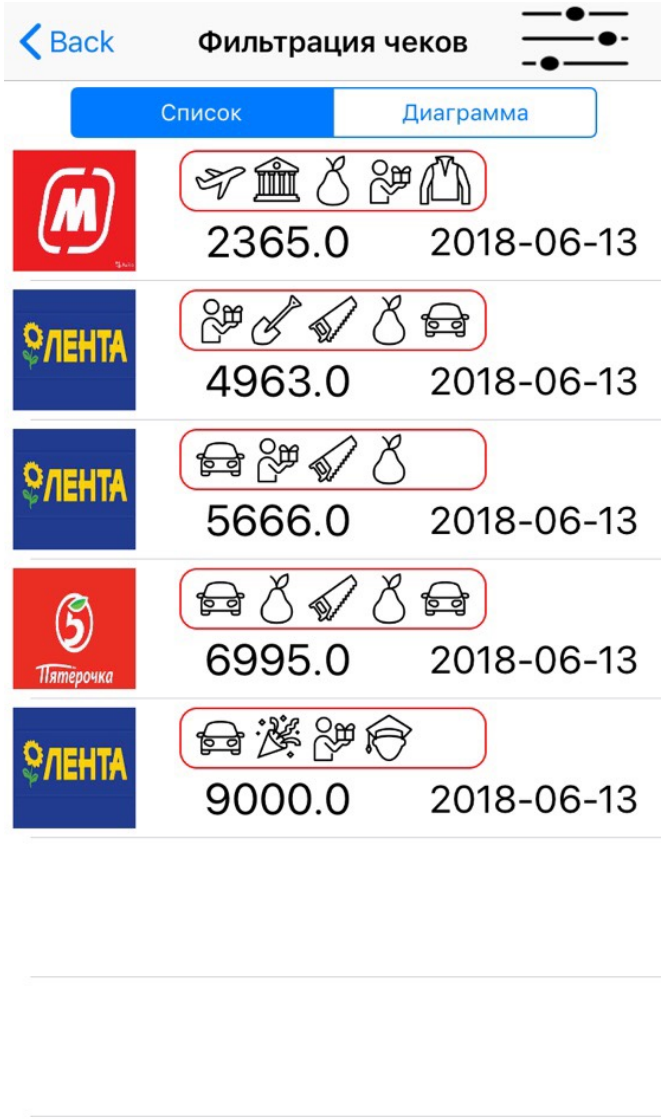

Рис. 4.10. Результаты фильтрации.

На изображении 4.9. были заданы критерии фильтрации. Порядком отображения чеков задано по возрастанию, так же нажата кнопка «Сумма чека» которая фильтрует по данным из двух полей «От» и «До», которые были заданы в пределах от двух тысяч и до девяти тысяч. Результат фильтрации постностью корректен и удовлетворяет заданным ранее условиям. По списку отфильтрованных чеков можно просмотреть построенную круговую диаграмму. Для отображения диаграммы требуется нажать на кнопку

«Диаграмма» в верней части экрана. Окно диаграммы отфильтрованных чеков отображено на рисунке 4.11.

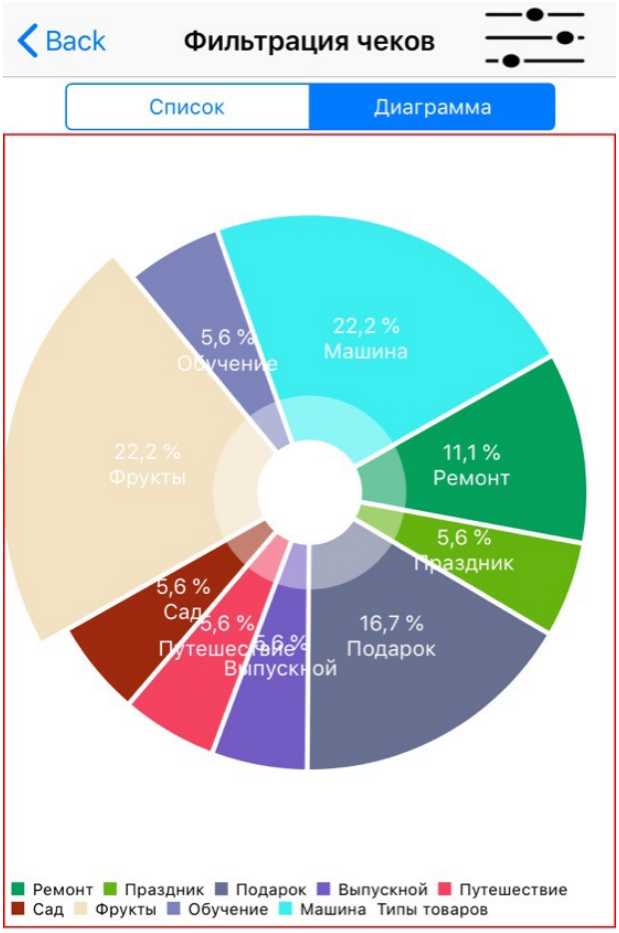

Рис. 4.11. Диаграмма отфильтрованных чеков.

Круговая диаграмма строится только из чеков находящимся в списке всех ранее созданных чеков. Если отобразить диаграмму до фильтрации чеков, то диаграмма будет содержать данные по всем чекам. Диаграмма строится исходя из количества разных типов товара отмеченных в списке чеков.

Если создать чек, магазин или тип товара без заполнения обязательных для этого полей, каждый раз будет отображаться уведомление об ошибке с информацией о поле которое требует обязательного заполнения для продолжения работы. Уведомление об ошибке изображено на рисунке 4.12.

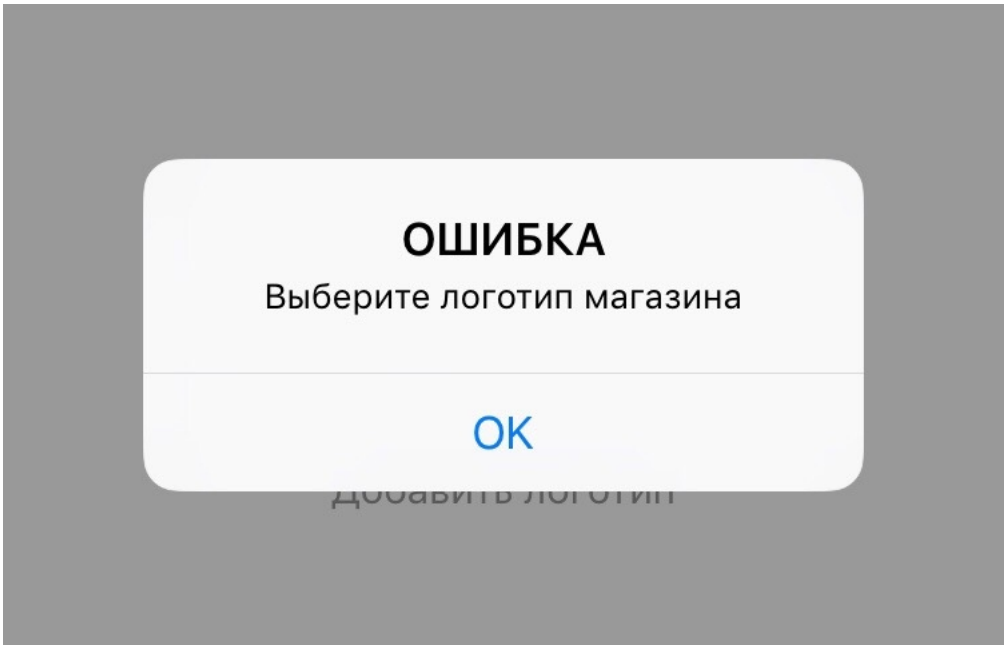

Рис. 4.12. Уведомление об ошибке.

По результатам проведения испытаний разработанного приложения было установлено, что приложение полостью работоспособно и готово к эксплуатации. Все функциональные возможности разработанного приложения полностью удовлетворяют требования к приложению, поставленные в начале проектирования по всем пунктам.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы было спроектировано и разработано мобильное приложение учета расходов физических лиц.

Данное приложение выполняет функции учета расходов пользователя и позволяет любому пользователю держать все свои расходы в одном месте. Архитектура приложения предполагает минимальное участие человека и максимальную информативность и гибкость отчетности. Типизация расходов поможет пользователям лучше оценивать траты для более умного и продуманного планирования своего будущего бюджета. Фильтрация предоставляет возможность для более детального изучения расходов опираясь на один или несколько важных для пользователя критериев. Помимо этого составление диаграмм позволяет более точно отсеивать ненужные типы товара на которые пользователь не задумываясь тратит свои личные средства.

После собранных данных о расходах, изучения похожих приложений были составлены требования, на основе которых спроектировано и разработано приложение. После проведения тестирования, опираясь на их результат, следует сделать вывод – разработанное приложение полностью удовлетворяет всем требованиям, предъявленным к приложению на этапе постановки задачи.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Новый кассовый чек по закону № 54-ФЗ: путеводитель по всем реквизитам: [Электронный ресурс] // URL: https://www.klerk.ru/buh/articles/463067/. - (дата обращения: 18.02.2018).
2. Проектирование баз данных : [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Проектирование_баз_данных. - (дата обращения: 28.03.2018).
3. Инфологическое проектирование : [Электронный ресурс] // URL: http://wiki.mvtom.ru/index.php/Инфологическое_проектирование. - (дата обращения: 03.03.2018).
4. ER-модель : [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/ER-модель. - (дата обращения: 26.03.2018).
5. Процедура проектирования: [Электронный ресурс] // URL: http://citforum.ru/database/dbguide/4-7.shtml. - (дата обращения: 27.03.2018).
6. Логическое проектирование бд : [Электронный ресурс] // URL: https://studfiles.net/preview/719824/page:4/. - (дата обращения: 27.03.2018).
7. Swift(язык программирования): [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Swift_(язык_программирования) - (дата обращения: 08.04.2018).
8. Язык программирования Swift. Русская версия : [Электронный ресурс] // URL: https://habr.com/post/225841/. - (дата обращения: 15.04.2018).
9. Xcode : [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Xcode - (дата обращения: 03.05.2018).
10. Стоит ли изучать Swift? : [Электронный ресурс] // URL: https://proglib.io/p/why-swift - (дата обращения: 09.04.2018).
11. Apple Xcode IDE от Apple. Русская версия : [Электронный ресурс] // URL: http://wnfx.ru/apple-xcode-ide-ot-apple/. - (дата обращения: 29.04.2018).

12. Как научится программировать iOS: [Электронный ресурс] // URL: http://maxmikheev.ru/blog/2016/03/05/how-to-learn-ios-development-from-scratch/ - (дата обращения: 01.05.2018).
13. Кассовый чек : [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Кассовый_чек - (дата обращения: 04.03.2018).
14. На что тратят деньги россияне : [Электронный ресурс] // URL: https://dengi.161.ru/text/business/573531.html - (дата обращения: 22.02.2018).
15. Экономический кризис 2017 года в России: хроника падения : [Электронный ресурс] // URL: http://bs-life.ru/makroekonomika/krizis2017.html - (дата обращения: 23.02.2018).
16. Чек онлайн-кассы: все о реквизитах : [Электронный ресурс] // URL: http://konturalco.ru/base/chek-onlajn-kassy-vse-o-rekvizitax - (дата обращения: 01.03.2018).
17. Порядок регистрации онлайн-кассы в налоговой инспекции реквизитах : [Электронный ресурс] // URL: http://online-kassa.pro/zakon/registratsiya-kkt-v-nalogovoj.html - (дата обращения: 09.03.2018).

ПРИЛОЖЕНИЕ

```
import UIKit
import CoreData
import Charts

class ListFilterVC: UIViewController, UITableViewDataSource {

   @IBOutlet var diagramView: UIView!
   @IBOutlet var pieChart: PieChartView!

   @IBOutlet var segmentListDiagOutlet: UISegmentedControl!

   @IBAction func segmentListDiag(_ sender: Any) {
      if(segmentListDiagOutlet.selectedSegmentIndex == 0){
         if(self.diagramView.alpha == 1){
            UIView.animate(withDuration: 0.1, delay: 0.0, options: .curveEaseOut,
animations: {
               self.diagramView.frame.origin.y = 0
               self.diagramView.alpha = 0
               self.viewTable.alpha = 1
            })
         }
      }
      if(segmentListDiagOutlet.selectedSegmentIndex == 1){
         if(diagramView.alpha == 0){
            UIView.animate(withDuration: 0.1, delay: 0.0, options: .curveEaseOut,
animations: {
               self.diagramView.frame.origin.y = (UIScreen.main.bounds.height -
self.diagramView.frame.size.height) - 82
```

```
                self.diagramView.alpha = 1
                self.viewTable.alpha = 0.5
            })
        }
        setChart(values: dicChart)
    }


}
@IBOutlet var viewFilter: UIView!
@IBOutlet var ascendingOutletButton: UIButton!
@IBOutlet var waningOutletButton: UIButton!
@IBOutlet var dateOutletButton: UIButton!
@IBOutlet var priceOutletButton: UIButton!
@IBOutlet var shopOutletButton: UIButton!


@IBOutlet var applyButtonOutlet: UIButton!
@IBOutlet var cancerButtonOutlet: UIButton!

@IBOutlet var textFieldFrom: UITextField!
@IBOutlet var textFieldBefore: UITextField!

@IBOutlet var viewTable: UITableView!

var checks: [NSManagedObject] = []
var types: [NSManagedObject] = []
var shops: [NSManagedObject] = []
var properies: [NSManagedObject] = []
var dicProperty = [Int: Dictionary<String, String>]()
```

```
    var dicShop = [Int: Dictionary<String, String>]()
    var dicType = [Int: Dictionary<String, String>]()
    var dicChart = [Int: Dictionary<String, String>]()

    let dateFormatter = DateFormatter()

    let picker = UIDatePicker()
    var ascendingFilter = 0;
    var waningFilter = 0;
    var dateFilter = 0;
    var priceFilter = 0;
    var shopFilter = 0;

@IBAction func barGetFilter(_ sender: Any) {
        if self.viewFilter.alpha == 0 {
            UIView.animate(withDuration: 0.1, delay: 0.0, options: .curveEaseOut,
animations: {
                self.viewFilter.frame.origin.y = ( (UIScreen.main.bounds.height -
self.viewFilter.frame.size.height) / 3)
                self.viewFilter.alpha = 1
            })
        } else {
            UIView.animate(withDuration: 0.3, delay: 0.0, options: .curveEaseIn,
animations: {
                self.viewFilter.frame.origin.y = 0
                self.viewFilter.alpha = 0
            })
        }
    }
```

```
    @IBAction func enableFilter(_ sender: Any) {
        var dateFrom = Date()
        var dateBefore = Date()
        var priceFrom = Double()
        var priceBefore = Double()

        if(dateFilter == 1){
            let calendar = Calendar.current
            var dateComponent = DateComponents()

            let from = textFieldFrom.text?.split(separator: ".")
            let before = textFieldBefore.text?.split(separator: ".")
            dateComponent.year = Int(from![2])
            dateComponent.month = Int(from![1])
            dateComponent.day = Int(from![0])
            dateFrom = calendar.date(from: dateComponent)!

            dateComponent.year = Int(before![2])
            dateComponent.month = Int(before![1])
            dateComponent.day = Int(before![0])
            dateBefore = calendar.date(from: dateComponent)!
        }


        let appDelegate = UIApplication.shared.delegate as? AppDelegate
        let managetContext = appDelegate?.persistentContainer.viewContext
        let fetchRequest = NSFetchRequest<NSManagedObject>(entityName:
"Check")

        if(dateFilter == 1){
```

```
            let predicateDate = NSPredicate(format: "check_date >= %@ AND
check_date <= %@", (dateFrom as NSDate), (dateBefore as NSDate))
                fetchRequest.predicate = predicateDate
            if(ascendingFilter == 1){
                let predicateAscendingFilter = NSSortDescriptor(key: "check_date",
ascending: true)
                fetchRequest.sortDescriptors = [predicateAscendingFilter]
            }
            if(waningFilter == 1){
                let predicateAscendingFilter = NSSortDescriptor(key: "check_date",
ascending: false)
                fetchRequest.sortDescriptors = [predicateAscendingFilter]
            }
}
        if(priceFilter == 1){
            priceFrom = Double(textFieldFrom.text!)!
            priceBefore = Double(textFieldBefore.text!)!
            let predicatePrice = NSPredicate(format: "check_price >= %@ AND
check_price <= %@", (priceFrom as NSNumber), (priceBefore as NSNumber))
                fetchRequest.predicate = predicatePrice
            if(ascendingFilter == 1){
                let predicateAscendingFilter = NSSortDescriptor(key: "check_price",
ascending: true)
                fetchRequest.sortDescriptors = [predicateAscendingFilter]
            }
            if(waningFilter == 1){
                let predicateAscendingFilter = NSSortDescriptor(key: "check_price",
ascending: false)
                fetchRequest.sortDescriptors = [predicateAscendingFilter]
            }
```

```
        }
do {
            checks = (try managetContext?.fetch(fetchRequest))!
            print("Я Взял Данные и бд")
        } catch {
            let fetchError = error as NSError
            print(fetchError)
            print("НЕ ПОЛУЧИЛСОЬ ВЗЯТЬ ИЗ БД")
        }
        dicChart = [:]
        viewTable.reloadData()

        UIView.animate(withDuration: 0.3, delay: 0.0, options: .curveEaseIn,
animations: {
            self.viewFilter.frame.origin.y = 0
            self.viewFilter.alpha = 0
        })
    }



    func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return checks.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
```

```
        var timed = [String: String]()
        let cell = viewTable.dequeueReusableCell(withIdentifier: "cell") as!
FilterCustomTVC
        let check = checks[indexPath.row]

        let id_shop = Int((dicProperty[check.value(forKey: "id_check") as!
Int]?["id_shop"])!)
        let id_check_property = Int64((dicProperty[check.value(forKey: "id_check")
as! Int]?["id_check_property"])!)
        //let create_date = dicProperty[check.value(forKey: "id_check") as!
Int]?["create_date"]
        let shop_logo = dicShop[id_shop!]!["shop_logo"]
        let typeIndex = getTypeInCheck(id: id_check_property!)
        let imageDate = UserDefaults.standard.object(forKey: (shop_logo)!) as!
NSData

        cell.shopImage.image = UIImage(data: imageDate as Data)
        cell.dateText.text = dateFormatter.string(from:check.value(forKey:
"check_date") as! Date)
        cell.priceText.text = String((check.value(forKey: "check_price") as?
Double)!)

        for i in typeIndex {

            if(dicChart[i.value(forKey: "id_product_type") as! Int] == nil){
                timed = [:]
                timed["type_logo"] = dicType[i.value(forKey: "id_product_type") as!
Int]!["type_logo"]!
                timed["type_name"] = dicType[i.value(forKey: "id_product_type") as!
Int]!["type_name"]!
```

```
            timed["value"] = String(1)
            dicChart[i.value(forKey: "id_product_type") as! Int] = timed
        } else {
            dicChart[i.value(forKey: "id_product_type") as! Int]!["value"] =
                String(Int(dicChart[i.value(forKey: "id_product_type") as!
Int]!["value"]!)! + 1)
        }
    }

    if(typeIndex.count >= 1){
        cell.productType_1.image = UIImage(named:
dicType[typeIndex[0].value(forKey: "id_product_type") as! Int]!["type_logo"]!)!
        if(typeIndex.count >= 2){
            cell.productType_2.image = UIImage(named:
dicType[typeIndex[1].value(forKey: "id_product_type") as! Int]!["type_logo"]!)!
            if(typeIndex.count >= 3){
                cell.productType_3.image = UIImage(named:
dicType[typeIndex[2].value(forKey: "id_product_type") as! Int]!["type_logo"]!)!
                if(typeIndex.count >= 4){
                    cell.productType_4.image = UIImage(named:
dicType[typeIndex[3].value(forKey: "id_product_type") as! Int]!["type_logo"]!)!
                    if(typeIndex.count == 5){
                        cell.productType_5.image = UIImage(named:
dicType[typeIndex[4].value(forKey: "id_product_type") as! Int]!["type_logo"]!)!

                    }
                }
            }
        }
    }
```

```
        return cell
    }

    @IBAction func cancelFilter(_ sender: Any) {
        UIView.animate(withDuration: 0.3, delay: 0.0, options: .curveEaseIn,
animations: {
            self.viewFilter.frame.origin.y = 0
            self.viewFilter.alpha = 0
        })
    }

    @IBAction func waningButton(_ sender: Any) {
        if(waningOutletButton.layer.borderWidth == 0){
            if(ascendingOutletButton.layer.borderWidth >= 1){
                ascendingOutletButton.layer.borderWidth = 0
                ascendingOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.0).cgColor
                ascendingFilter = 0
            }
            waningOutletButton.layer.borderWidth = 2
            waningOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.7).cgColor
            waningFilter = 1
        } else {
            waningOutletButton.layer.borderWidth = 0
            waningOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.0).cgColor
            waningFilter = 0
        }
    }
```

```
    @IBAction func ascendingButton(_ sender: Any) {
        if(ascendingOutletButton.layer.borderWidth == 0){
            if(waningOutletButton.layer.borderWidth >= 1){
                waningOutletButton.layer.borderWidth = 0
                waningOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.0).cgColor
                waningFilter = 0
            }
            ascendingOutletButton.layer.borderWidth = 2
            ascendingOutletButton.layer.borderColor = UIColor.blue.cgColor
            ascendingOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.7).cgColor
            ascendingFilter = 1
        } else {
            ascendingOutletButton.layer.borderWidth = 0
            ascendingOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.0).cgColor
            ascendingFilter = 0
        }
    }

    @IBAction func dateButton(_ sender: Any) {
        if(dateOutletButton.layer.borderWidth == 0){
            if(priceOutletButton.layer.borderWidth >= 1){
                priceOutletButton.layer.borderWidth = 0
                priceOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.0).cgColor
                priceFilter = 0
            }
```

```
            dateOutletButton.layer.borderWidth = 2
            dateOutletButton.layer.borderColor = UIColor.blue.cgColor
            dateOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.7).cgColor
            dateFilter = 1
            enabledTextField()
            getTadeTicker()
        } else {
            dateOutletButton.layer.borderWidth = 0
            dateOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.0).cgColor
            disabledTextField()
            dateFilter = 0
        }
    }

    @IBAction func priceButton(_ sender: Any) {
        if(priceOutletButton.layer.borderWidth == 0){
            if(dateOutletButton.layer.borderWidth >= 1){
                dateOutletButton.layer.borderWidth = 0
                dateOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.0).cgColor
                dateFilter = 0
            }
            priceOutletButton.layer.borderWidth = 2
            priceOutletButton.layer.borderColor = UIColor.blue.cgColor
            priceOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.7).cgColor
            priceFilter = 1
            enabledTextField()
```

```
            getTextField()
        } else {
            priceOutletButton.layer.borderWidth = 0
            priceOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.0).cgColor
            disabledTextField()
            priceFilter = 0
        }
    }

    @IBAction func shopButton(_ sender: Any) {
        if(shopOutletButton.layer.borderWidth == 0){
            shopOutletButton.layer.borderWidth = 2
            shopOutletButton.layer.borderColor = UIColor.blue.cgColor
            shopOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.7).cgColor
        } else {
            shopOutletButton.layer.borderWidth = 0
            shopOutletButton.layer.backgroundColor =
UIColor.gray.withAlphaComponent(0.0).cgColor
        }
    }

    func getTextField(){
        textFieldBefore.inputView = nil
        textFieldFrom.inputView = nil
        textFieldFrom.inputAccessoryView = nil
        textFieldBefore.inputAccessoryView = nil
        textFieldFrom.keyboardType = UIKeyboardType.decimalPad
        textFieldBefore.keyboardType = UIKeyboardType.decimalPad
```

```
    }

    func getTadeTicker() {

        let toolbarFrom = UIToolbar()
        let toolbarBefore = UIToolbar()

        let doneFrom = UIBarButtonItem(barButtonSystemItem: .done, target: nil,
action: #selector(donePressedFrom(dataPicker:)))
        let doneBefore = UIBarButtonItem(barButtonSystemItem: .done, target: nil,
action: #selector(donePressedBefore(dataPicker:)))
        toolbarFrom.sizeToFit()
        toolbarBefore.sizeToFit()
        toolbarFrom.setItems([doneFrom], animated: true)
        toolbarBefore.setItems([doneBefore], animated: true)

        textFieldFrom.inputAccessoryView = toolbarFrom
        textFieldBefore.inputAccessoryView = toolbarBefore
        textFieldFrom.inputView = picker
        textFieldBefore.inputView = picker
        picker.datePickerMode = .date
    }

    func disabledTextField() {
        textFieldBefore.text = nil
        textFieldFrom.text = nil
        textFieldFrom.placeholder = "От"
        textFieldBefore.placeholder = "До"
        textFieldBefore.isEnabled = false
        textFieldFrom.isEnabled = false
```

```
}

func enabledTextField() {
    textFieldBefore.text = nil
    textFieldFrom.text = nil
    textFieldFrom.placeholder = "От"
    textFieldBefore.placeholder = "До"
    textFieldBefore.isEnabled = true
    textFieldFrom.isEnabled = true
}

@objc func donePressedFrom(dataPicker: UIDatePicker){
    let dateFormater = DateFormatter()
    dateFormater.dateFormat = "dd.MM.yyyy"

    textFieldFrom.text = dateFormater.string(from: picker.date)
    self.view.endEditing(true)
}
@objc func donePressedBefore(dataPicker: UIDatePicker){
    let dateFormater = DateFormatter()
    dateFormater.dateFormat = "dd.MM.yyyy"

    textFieldBefore.text = dateFormater.string(from: picker.date)
    self.view.endEditing(true)
}

var numberOfDownloadsEntries:[ChartDataEntry] = []


override func viewDidLoad() {
```

```
super.viewDidLoad()

diagramView.alpha = 0
diagramView.frame.origin.y = 0
segmentListDiagOutlet.selectedSegmentIndex = 0

pieChart.drawEntryLabelsEnabled = true
pieChart.clearsContextBeforeDrawing = true
pieChart.holeRadiusPercent = 0.19
pieChart.holeColor = NSUIColor.white.withAlphaComponent(0.4)
pieChart.layer.borderWidth = 1
pieChart.layer.borderColor = UIColor.red.cgColor
pieChart.usePercentValuesEnabled = true
pieChart.chartDescription?.text = ""
pieChart.transparentCircleRadiusPercent = 0.35



self.viewFilter.alpha = 0
self.viewFilter.frame.origin.y = 0
self.viewFilter.layer.cornerRadius = 30

ascendingOutletButton.layer.cornerRadius = 40
waningOutletButton.layer.cornerRadius = 40
dateOutletButton.layer.cornerRadius = 40
priceOutletButton.layer.cornerRadius = 40
shopOutletButton.layer.cornerRadius = 40
applyButtonOutlet.layer.cornerRadius = 20
cancerButtonOutlet.layer.cornerRadius = 20
```

```
        disabledTextField()

        getShops()
        getTypes()
        getProperties()


        let appDelegate = UIApplication.shared.delegate as? AppDelegate
        let managetContext = appDelegate?.persistentContainer.viewContext
        let fetchRequest = NSFetchRequest<NSManagedObject>(entityName:
"Check")
        do {
            let predicateAscendingFilter = NSSortDescriptor(key: "check_date",
ascending: false)
            fetchRequest.sortDescriptors = [predicateAscendingFilter]
            checks = try managetContext!.fetch(fetchRequest)
            print("Я Взял Данные и бд")
        } catch {
            let fetchError = error as NSError
            print(fetchError)
            print("НЕ ПОЛУЧИЛСОЬ ВЗЯТЬ ИЗ БД")
        }
    }


    func resizeImage(image: UIImage, targetSize: CGSize) -> UIImage {
        let size = image.size

        let widthRatio  = targetSize.width  / image.size.width
        let heightRatio = targetSize.height / image.size.height
```

```
        // Figure out what our orientation is, and use that to form the rectangle
        var newSize: CGSize
        if(widthRatio > heightRatio) {
            newSize = CGSize(width: size.width * heightRatio, height: size.height *
heightRatio)
        } else {
            newSize = CGSize(width: size.width * widthRatio,  height: size.height *
widthRatio)
        }

        // This is the rect that we've calculated out and this is what is actually used
below
        let rect = CGRect(x: 0, y: 0, width: newSize.width, height: newSize.height)

        // Actually do the resizing to the rect using the ImageContext stuff
        UIGraphicsBeginImageContextWithOptions(newSize, false, 1.0)
        image.draw(in: rect)
        let newImage = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()
        return newImage!
    }

    func setChart(values: [Int: Dictionary<String, String>]) {

        var dataEntries: [PieChartDataEntry] = []
        var colors: [UIColor] = []
        var piece = PieChartDataEntry(value: 0)

        for i in 0..<values.count {
```

```
            let red = Double(arc4random_uniform(256))
            let green = Double(arc4random_uniform(256))
            let blue = Double(arc4random_uniform(256))

            let color = UIColor(red: CGFloat(red/255), green: CGFloat(green/255),
blue: CGFloat(blue/255), alpha: 1)
            colors.append(color)
        }

        for i in values {
            piece = PieChartDataEntry(value: 0)
            piece.value = Double(i.value["value"] as! String)!
            piece.label = i.value["type_name"]
            dataEntries.append(piece)
        }

        let formatter = NumberFormatter()
        formatter.numberStyle = .percent
        formatter.maximumFractionDigits = 1
        formatter.multiplier = 1.0


        let pieChartDataSet = PieChartDataSet(values: dataEntries, label: "Типы
товаров")
        let pieChartData = PieChartData(dataSet: pieChartDataSet)
        pieChartDataSet.sliceSpace = 3.0

pieChartData.setValueFormatter(DefaultValueFormatter(formatter:formatter))

        pieChartDataSet.colors = colors
```

```
        pieChart.data = pieChartData

    }

    func getTypes() {
        let appDelegate = UIApplication.shared.delegate as? AppDelegate
        let managetContext = appDelegate?.persistentContainer.viewContext
        let fetchRequestTypeInCheck =
NSFetchRequest<NSManagedObject>(entityName: "ProductType")
        do {
            types = try managetContext!.fetch(fetchRequestTypeInCheck)
            print("Я Взял Данные и бд")
            var timed = [String: String]()
            for i in types {
                timed = [:]
                timed["type_logo"] = (i.value(forKey: "type_logo") as! String)
                timed["type_name"] = (i.value(forKey: "type_name") as! String)
                dicType[(i.value(forKey: "id_product_type") as! Int)] = timed
            }
        } catch {
            let fetchError = error as NSError
            print(fetchError)
            print("НЕ ПОЛУЧИЛСОЬ ВЗЯТЬ ИЗ БД")
        }
    }

    func getShops() {
        let appDelegate = UIApplication.shared.delegate as? AppDelegate
        let managetContext = appDelegate?.persistentContainer.viewContext
        let fetchRequestShop = NSFetchRequest<NSManagedObject>(entityName:
```

```
"Shop")
        do {
            shops = try managetContext!.fetch(fetchRequestShop)
            print("Я Взял Данные и бд")
            var timed = [String: String]()
            for i in shops {
                timed = [:]
                timed["shop_name"] = (i.value(forKey: "shop_name") as! String)
                timed["shop_logo"] = (i.value(forKey: "shop_logo") as! String)
                dicShop[(i.value(forKey: "id_shop") as! Int)] = timed
            }
        } catch {
            let fetchError = error as NSError
            print(fetchError)
            print("НЕ ПОЛУЧИЛСОЬ ВЗЯТЬ ИЗ БД")
        }
    }

    func getProperties() {
        let appDelegate = UIApplication.shared.delegate as? AppDelegate
        let managetContext = appDelegate?.persistentContainer.viewContext
        let fetchRequest2 = NSFetchRequest<NSManagedObject>(entityName:
"CheckProperties")
        do {
            properies = try managetContext!.fetch(fetchRequest2)
            print("Я Взял Данные и бд")
            var timed = [String: String]()

            dateFormatter.dateFormat = "yyyy-MM-dd"
            for i in properies {
```

```
            let dateString = dateFormatter.string(from:i.value(forKey: "create_date")
as! Date)
            timed = [:]
            timed["id_shop"] = (String(i.value(forKey: "id_shop") as! Int64))
            timed["id_check_property"] = (String(i.value(forKey:
"id_check_property") as! Int64))
            timed["create_date"] = (dateString)
            timed["input_type"] = (String(i.value(forKey: "input_type") as! Bool))
            dicProperty[(i.value(forKey: "id_check") as! Int)] = timed
          }
      } catch {
          let fetchError = error as NSError
          print(fetchError)
          print("НЕ ПОЛУЧИЛСОЬ ВЗЯТЬ ИЗ БД")
      }

  }

  func getTypeInCheck(id:Int64) -> [NSManagedObject] {
      var types = [NSManagedObject]()
      let appDelegate = UIApplication.shared.delegate as? AppDelegate
      let managetContext = appDelegate?.persistentContainer.viewContext
      let fetchRequest = NSFetchRequest<NSManagedObject>(entityName:
"TypeInCheck")
      let predicatePrice = NSPredicate(format: "id_check_property == %i", id)
      fetchRequest.predicate = predicatePrice
      do {
          types = try managetContext!.fetch(fetchRequest)
          print("Я Взял Данные и бд")
```

```
        } catch {
            let fetchError = error as NSError
            print(fetchError)
            print("НЕ ПОЛУЧИЛСОЬ ВЗЯТЬ ИЗ БД")
        }
        print("Boот\(types.count)")
        return types
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.

    }

    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        if (touches.first) != nil {
            view.endEditing(true)
        }
        super.touchesBegan(touches, with: event)
    }
}
```

Выпускная квалификационная работа выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«07» июня 2018 г.

Попов Дмитрий Вадимович

(подпись) *(Ф.И.О.)*