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

РАБОТА ПРОВЕРЕНА

Рецензент
Директор ООО «Наполеон Айти»

_____ П.С. Подкорытов

“___”_____ 2017 г.

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

“___”_____ 2017 г.

РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ФОРМИРОВАНИЯ И ОФОРМЛЕНИЯ ЗАКАЗОВ НА ПРОДУКЦИЮ В РОЗНИЧНОЙ СЕТИ МАГАЗИНОВ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2017.11-032-1909.ВКР

Научный руководитель
доцент кафедры СП, к.п.н.
_____ О.Н. Иванова

Автор работы,
студент группы ВМИ-401
_____ А.Г. Михайлов

Ученый секретарь
(нормоконтролер)

_____ О.Н. Иванова

“___”_____ 2017 г.

Челябинск–2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5
1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 7
1.1. Постановка задачи 7
1.2. Обзор аналогов 8
1.3. Средства разработки 10
2. ПРОЕКТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ФОРМИРОВАНИЯ И ОФОРМЛЕНИЯ ЗАКАЗОВ НА ПРОДУКЦИЮ В РОЗНИЧНОЙ СЕТИ МАГАЗИНОВ 14
2.1. Определение требований к проектируемому приложению 14
2.2. Разработка диаграммы вариантов использования и диаграммы последовательности 15
2.3. Разработка диаграммы деятельности 19
3. РЕАЛИЗАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ФОРМИРОВАНИЯ И ОФОРМЛЕНИЯ ЗАКАЗОВ НА ПРОДУКЦИЮ В РОЗНИЧНОЙ СЕТИ МАГАЗИНОВ 21
3.1. Методология разработки и структура проекта 21
3.2. Архитектура системы 23
3.3. Модель внутреннего хранилища данных приложения 24
3.4. Особенности реализации приложения 25
4. ТЕСТИРОВАНИЕ 29
4.1. Выбор способов тестирования системы 29
4.2. Функциональное тестирование 30
4.3. Интегральное тестирование 35
ЗАКЛЮЧЕНИЕ 36
ЛИТЕРАТУРА 37
ПРИЛОЖЕНИЕ 40

ВВЕДЕНИЕ

Актуальность темы

Индустрия мобильных приложений занимает громадную часть электронного бизнеса в современных компьютерных сетях. В последние годы наблюдается существенный рост рынка мобильных приложений во всех его областях. Так, в 2012 году рынок мобильных приложений оценивался в 53 миллиарда долларов, а в 2016 году он составил уже около 100 миллиардов долларов [22]. Тем не менее, имеются перспективы создания новых мобильных приложений, т.к. доля пользователей именно такими устройствами постоянно растет, а также мобильные технологии постоянно развиваются и совершенствуются.

Частичный или полный перенос сферы продаж в интернет-пространство является логичным и целесообразным для любой компании, т.к. это позволяет сэкономить издержки на открытие новых магазинов, повышает узнаваемость бренда и расширяет географию продаж. Но в политику некоторых компаний не входит продажа через интернет, поэтому зачастую в интернете встречается лишь услуги бронирования товаров.

Цель и задачи

Целью работы является разработка мобильного приложения формирования и оформления заказов на продукцию в розничной сети магазинов. Для достижения цели должны быть решены следующие задачи:

1) произвести постановку задачи по созданию мобильного приложения;
2) произвести обзор существующих аналогов для разрабатываемого мобильного приложения;
3) изучить современные платформы и средства разработки для операционной системы iOS;
4) определить требования и спроектировать мобильное приложение;
5) реализовать мобильное приложение;
6) протестировать мобильное приложение.

Структура и объем работы

Работа состоит из введения, четырех глав, заключения, библиографического списка и одного приложения. Объем работы составляет 39 страниц, объем библиографии – 24 источника. Объем приложения – 3 страницы.

В главе «Теоретическая часть» описана постановка задачи, произведен обзор существующих мобильных приложений для розничных сетей магазинов и программных средств для разработки мобильных приложений для операционной системы iOS.

Глава «Проектирование мобильного приложения для формирования и оформления заказов на продукцию в розничной сети магазинов» посвящена определению требований к разрабатываемому мобильному приложению. В этой же главе описываются диаграммы прецедентов и последовательности.

В третьей главе, «Реализация мобильного приложения для формирования и оформления заказов на продукцию в розничной сети магазинов», рассмотрена методология разработки приложения, описана диаграмма компонентов реализованного мобильного приложения и приведены некоторые методы.

Глава «Тестирование» посвящена результатам тестирования мобильного приложения. Представлены результаты функционального тестирования, выполненные в работающем приложении, и интегрального тестирования в различных условиях окружения.

В заключении сделаны выводы о проделанной работе и сформулированы перспективы дальнейшей разработки.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Постановка задачи

Разрабатываемый продукт должен представлять собой мобильное приложение и предназначен для розничной сети магазинов. Он будет предоставлять информацию о магазинах: расположение магазинов и кратчайший маршрут до каждого, время работы. Также, используя приложение, можно будет получить информацию о товарах: список товаров сети, наличие и остаток в каждом магазине, проходящие акции с участием этого товара. В приложении будет возможность оформления заказа: выбор списка товаров и их количества, магазина и времени, к которому заказ должен быть сформирован в выбранном магазине.

Вся информация, которую можно будет получить в приложении, должна актуализироваться не реже, чем один раз в сутки. Если пользователь выберет магазин, ему автоматически должна показываться цена на товар в выбранном магазине, в противном же случае цена должна заменяться на среднюю по всей сети. Если пользователь включит у себя на устройстве определение геопозиции, то приложение должно автоматически определять город, в котором он находится, показывать магазины этого города и сортировать их по удаленности от текущего местоположения пользователя, а на карте будет возможность построить маршрут до любого магазина, не выходя за рамки приложения.

Для облегчения получения информации о товарах, в приложении должна будет внедрена система интеллектуального поиска и функция сканирования штрих-кода товаров. Интеллектуальный поиск должен включать в себя поиск по синонимам и исправление ошибок пользователя, а также формировать популярные поисковые запросы для дальнейшего отображения их в приложении. Функция сканирования должна позволять пользователю отсканировать штрих-код товара и перейти на экран подробной информации о товаре при условии, что такой товар есть в сети.

1.2. Обзор аналогов

В настоящее время рынок мобильных приложений для розничных сетей магазинов представлен следующим спектром приложений.

«Магнит» - мобильное приложение для розничной сети магазинов «Магнит». Позволяет просмотреть магазины выбранного города на карте, акции, проходящие в конкретном магазине, а также сформировать список покупок [20].

Достоинства:

- наличие обратной связи с разработчиками;
- возможность «поделиться» списком покупок через почту или социальные сети.

Недостатки:

- недружественный дизайн;
- сильно ограничен функционал приложения;
- отсутствие данных о наличии и стоимости товаров как в конкретном магазине, так и во всей сети;
- возможность просмотреть акции только добавив магазин в избранное или через карту; отсутствие информации об акциях, проходящих во всей сети;
- плохая оптимизация приложения, что вызывает множество сбоев и медленную работу;
- дизайн не адаптирован под различные разрешения различных устройств;
- невозможность изменить город после первой загрузки приложения;
- отсутствие функции бронирования заказа.

«Пятерочка» - мобильное приложение для розничной сети магазинов «Пятерочка». Также позволяет просмотреть магазины выбранного города на карте, акции, проходящие в конкретном магазине, а также сформировать список покупок [23].

Достоинства:

- наличие системы отзывов;
- возможность «поделиться» списком покупок через почту или социальные сети;
- возможность построить маршрут до магазина, используя различные карты.

Недостатки:

- обладает всеми недостатками, описанными выше.

«Семья» - мобильное приложение для розничной сети магазинов «Семья». Позволяет привязать скидочную карту в приложение для дальнейшего использования телефона вместо карты, просматривать магазины сети и проходящие акции, а также формировать список покупок [24].

Достоинства:

- при отсутствии скидочной карты есть возможность завести новую карту посредством приложения;
- возможность просмотра истории совершенных покупок;
- наличие обратной связи;
- возможность посмотреть магазины как списком, так и на карте.

Недостатки:

- обладает всеми недостатками, описанными выше;
- невозможность построить маршрут до магазина.

«Едадил» - мобильное приложение для поиска скидок на продовольственные товары в популярных розничных сетях. Позволяет просматривать магазины торговых сетей на карте, акции, проходящие в данный момент в этих сетях, а также составлять список покупок [16].

Достоинства:

- наличие поиска и по проходящим акциям и их фильтрация по различным критериям;
- наличие функции «Поделиться»;
- отсутствие функции бронирования заказа.

Недостатки:

- обладает многими недостатками, описанными выше.

Сведем наиболее важные параметры рассмотренных мобильных приложений в таблицу 1.

Табл. 1. Обзор аналогов

№ п/п	Критерий	**Магнит**	**Пятерочка**	**Семья**	**Едадил**
1.	Удобство использования	2/5	3/5	3/5	4/5
2.	Функциональность	1/5	2/5	3/5	3/5
3.	Наличие и полнота информации о товарах	2/5	2/5	2/5	2/5
4.	Дизайн приложения	2/5	3/5	3/5	3/5
5.	Оценка пользователей AppStore*	★★★☆☆	--	--	★★★★☆
6.	Дата последнего обновления*	17.10.2016	23.11.2016	06.11.2016	9.10.2016

* – по информации на 15.02.2017 из [1]

Как видно из таблицы, рассмотренные мобильные приложения почти по всем критериям не достигают до среднего уровня. В связи с этим было решено создать мобильное приложение с тем дизайном, который будет наиболее удобен и привычен для пользователя операционной системы iOS, а также обладающее всем основным функционалом, который может предоставить мобильное приложение для формирования и оформления заказов на продукцию в розничной сети магазинов.

1.3. Средства разработки

Компания Apple предоставляет собственную платформу для разработки под семейство iOS – Xcode, она включает в себя все необходимые инструменты для разработки, отладки и тестирования мобильного приложения [11]. Кроме стандартной платформы, существуют также платформы для

кроссплатформенной разработки мобильных приложений, например, Xamarin [10]. Но кроссплатформенные платформы не предоставляют той полноты инструментов, которые есть в Xcode, а именно: сложность написания автоматизированных тестов (чтобы реализовать тесты, необходимо создавать отдельное приложение, которое будет запускаться и тестировать необходимые участки нашего приложения), нестабильность симуляторов устройств (частые аварийные завершения работы, некорректность поведения в некоторых случаях). Нестабильность была проверена на собственном опыте при разработке проекта по предмету «Основы технологии программирования .NET», аварийный завершения работы происходили не реже чем каждый час работы симулятора. Кроме того, в таких платформах затрудняется верстка приложения и реализация его бизнес-логики, из-за возможности кроссплатформенной разработки, что повышает общую нестабильность мобильного приложения. Так, например, добавляя какой-либо элемент, мы не можем точно сказать, как он будет выглядеть на разных устройствах и разных системах, так как его настройка является общей для всех платформ, а тонкости реализации верстки для разных платформ различны.

Помимо недостатков, описанных выше, стоит учитывать своевременность обновлений приложения. Когда выходит новая версия операционной системы, необходимо поддерживать ее новые возможности для поддержания спроса на приложение. Xcode обновляется одновременно с обновлением операционной системы, что позволяет сразу начать использовать новые функции и возможности в разрабатываемых приложениях. В то время как при разработке на платформе Xamarin нет возможности оперативно выпустить обновление для приложения, так как необходимо ждать, пока в платформе появятся соответствующие инструменты. Именно поэтому для разработки продукта была выбрана платформа Xcode.

Для разработки мобильных приложений под семейство iOS существуют 2 языка программирования – Swift и Objective C. Для реализации продукта был выбран язык Objective C, так как его аналог – Swift – является

относительно молодым языком, поэтому он пока очень часто модифицируется, что затрудняет реализацию приложения и его поддержку в будущем. Objective C же, являясь более старым языком, меняется очень редко, в основном лишь добавляя новый функционал, что делает процессы реализации и поддержки более легкими.

Язык Objective C имеет ряд преимуществ:

1) язык является объектно-ориентированным, где даже встроенные типы данных представлены классами;

2) низкая сложность разработки и сопровождения (обширная документация языка, встроенная в среду разработки Xcode и широкое сообщество программистов);

3) высокая читаемость кода;

4) развитая система контроля ошибок.

Objective C построен на основе языка C, поддерживает статическую и динамическую типизацию, делегаты, события, итераторы, анонимные функции, исключения.

Для создания приложения будет использоваться фреймворк Cocoa Touch [2], который является частью iOS SDK – комплекта средств для разработки приложений для системы iOS. Данный фреймворк следует шаблону проектирования Model – View – Controller [8]. Использование фреймворка позволяет сократить объем работы, которую пришлось бы делать вручную.

В состав фреймворка входят следующие ключевые библиотеки:

1) Foundation Kit – библиотека, содержащая основные классы (классы с префиксом NS);

2) UIKit – библиотека, которая содержит GUI-классы (специфичные для iOS);

3) MapKit – библиотека для работы с картами и навигационными возможностями устройств;

4) GameKit – библиотека для взаимодействия с сервисом Game-Center;

5) iAd – библиотека для внедрения сервисов контекстной рекламы в приложении.

Также фреймворк Cocoa Touch включает в себя следующие технологии и механизмы: Core Animation, распознаватели жестов (в том числе мультитач-жестов), многозадачность.

Для работы с данными внутри приложения будет использоваться фреймворк Core Data [9]. Он предоставляет оболочку для работы с данными в виде объектного графа, сущностями и их связями, атрибутами. При этом Core Data скрывает множество деталей реализации взаимодействия с хранилищем данных, чем облегчает разработку приложения. Также Core Data очень хорошо сочетается с фреймворком Cocoa Touch, чем еще больше облегчает работу с ним. Работа с фреймворком Core Data легко осуществляется через библиотеку Magical Record [6].

Таким образом, нами была осуществлена постановка задачи, произведен анализ аналогичных программных продуктов, имеющихся на рынке, а также сделан выбор платформы и языка программирования для реализации мобильного приложения.

2. ПРОЕКТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ФОРМИРОВАНИЯ И ОФОРМЛЕНИЯ ЗАКАЗОВ НА ПРОДУКЦИЮ В РОЗНИЧНОЙ СЕТИ МАГАЗИНОВ

2.1. Определение требований к проектируемому приложению

В ходе проектирования мобильного приложения были определены функциональные возможности, предоставляемые пользователю.

Приложение должно показывать пользователю каталог товаров розничной сети. У каждого товара должна присутствовать следующая информация: название, характеристика, стоимость, изображение и другие полезные для пользователя данные.

Приложение должно автоматически определять город, в котором находится пользователь, а пользователь в свою очередь должен иметь возможность изменить выбранный приложением город или выбрать свой город в случае, если он определение своей геопозиции на устройстве.

Мобильное приложение должно показывать список магазинов выбранного города как списком, так и с помощью маркеров на карте, список городов должен быть отсортирован по удаленности от пользователя, если он включил определение своей геопозиции на устройстве.

Пользователь приложения должен иметь возможность выбрать магазин; после выбора магазина мобильное приложение должно обновить данные по товарам с учетом конкретного выбранного магазина, а именно стоимость и количество единиц товара в этом магазине.

Пользователь должен иметь возможность добавить товар в корзину для последующего оформления заказа. Если пользователь не выбрал магазин на момент добавления товара в корзину, приложение должно автоматически выбрать ближайший к пользователю магазин, в котором есть в наличии данный товар, либо вывести пользователю сообщение в случае, если данного товара нет в выбранном городе. Если в момент добавления товара в корзину у пользователя не выбран город, мобильное приложение должно показать список городов с возможностью выбора одного из них.

Пользователь должен иметь возможность оформить заказ, указав свои данные: ФИО, номер телефона, e-mail, время, когда он хочет забрать заказ и комментарий (по желанию), в случае если он выбрал товаров на сумму не менее 500 рублей. Приложение должно отправить оформленный пользователем заказ на сервер для последующей передачи этих данных в выбранный магазин.

Пользователь должен иметь возможность просмотреть отзывы к товарам и оставить свой, указав ФИО, e-mail и комментарий.

Пользователь должен иметь возможность отсканировать штрих-код товара, чтобы получить информацию о продукте, если он есть в сети или сообщение о его отсутствии.

Среди нефункциональных требований можно выделить следующие.

Приложение должно функционировать на устройствах iPhone пятого поколения (iPhone 4S) и выше, а также на устройствах iPad.

Приложение должно функционировать на ОС iOS версии не ниже 8.0.

Приложение должно быть доступно на iPhone только в портретной ориентации, а на iPad – только в ориентации пейзаж.

Приложение должно загружать информацию во внутреннюю базу данных из внешней базы, хранящей полное описание товаров, информацию о магазинах и городах.

2.2. Разработка диаграммы вариантов использования и диаграммы последовательности

Диаграмма прецедентов (вариантов использования) отражает отношения между актерами и прецедентами системы и позволяет описать систему на концептуальном уровне. Прецедент – возможность моделируемой системы, часть ее функциональности, благодаря которой актер может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов ее использования и описывает типичный способ взаимодействия пользователя с

системой. Диаграмма прецедентов обычно применяется для спецификации внешних требований к системе [14].

В ходе проектирования был выделен следующий актер.

Пользователь – пользователь приложения, которому доступна возможность использования всех его функций.

На основе функциональных требований к мобильному приложению была создана диаграмма прецедентов, которая представлена на рис. 1.

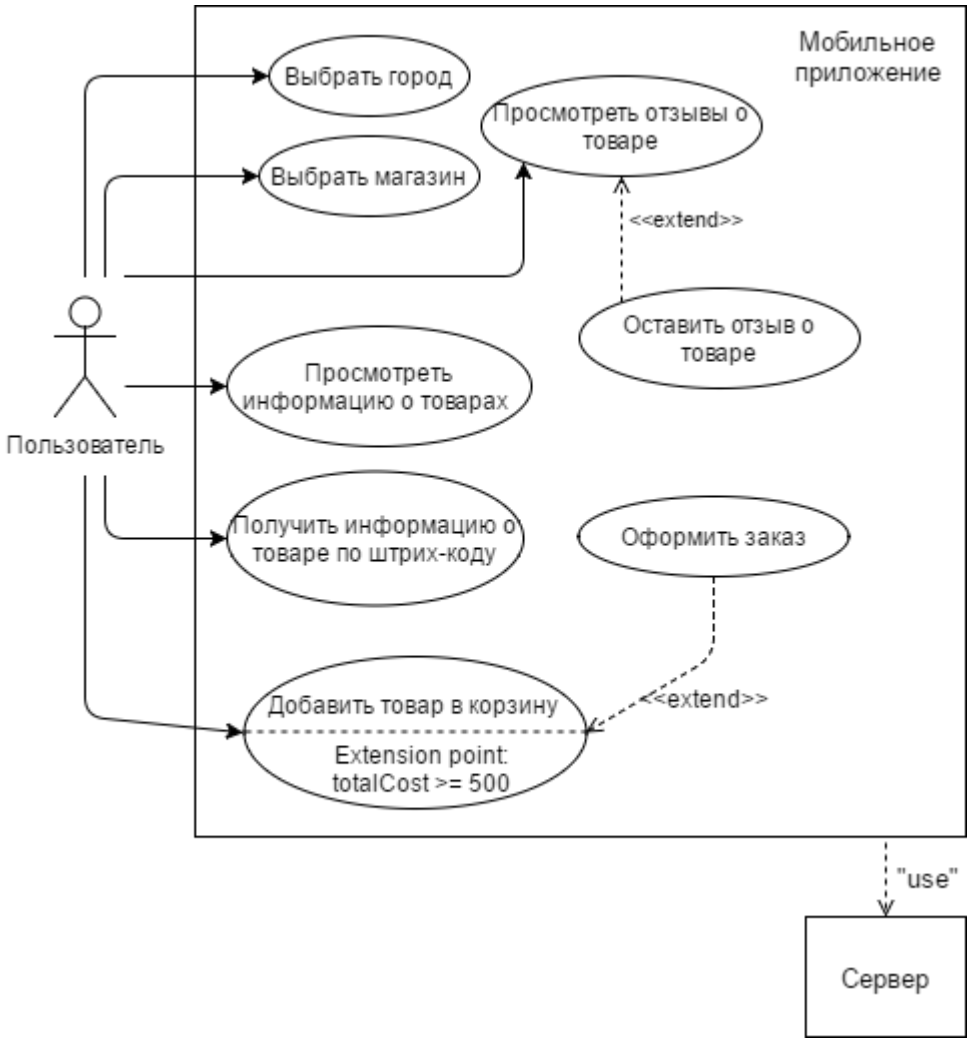

Рис. 1. Диаграмма прецедентов

Выбрать город – выбрать город для отображения магазинов данного города.

Выбрать магазин – выбрать магазин ранее выбранного города для отображения актуального для него информации о товарах – стоимость продукта, наличие и остаток в магазине.

Просмотреть информацию о товарах – просмотреть информацию о продуктах, которые предлагает розничная сеть, а именно название, характеристики, стоимость (средняя по сети, если магазин не выбран) и др.

Просмотреть отзывы о товаре – посмотреть отзывы о товаре, оставленные другими пользователями.

Оставить отзыв о товаре – поставить оценку о товаре по пятибалльной шкале и оставить комментарий к своей оценке с указанием своего ФИО и e-mail. Вариант использования доступен из интерфейса просмотра отзывов о товаре.

Добавить товар в корзину – добавить товар в корзину с последующей возможностью оформить заказ

Получить информацию о товаре по штрих-коду – отсканировать имеющийся штрих-код продукта и получить о нем информацию, при условии, что этот товар есть в сети.

Оформить заказ – оформить заказ, состоящий из ранее добавленных товаров в корзину с указанием своих данных: ФИО, e-mail, контактного номера телефона, временем, к которому заказ должен быть сформирован и комментария к заказу; заказ может быть оформлен при условии, что общая стоимость товаров в корзине не менее 500 рублей. Вариант использования доступен из интерфейса корзины.

Работа мобильного приложения зависит от сервера. Все данные, имеющееся в приложения – информация о товарах и магазинах – постоянно актуализируются с сервера. Также в ходе работы приложения на сервер отправляются данные об оставленных пользователем отзывах о товарах и оформленных пользователем заказов. Сервер сообщает мобильному приложению об успешности завершения этих действий и только потом, приложение показывает соответствующее уведомление пользователю.

Диаграмма последовательности (sequence diagram) – UML-диаграмма, которая представляет взаимодействие между линиями жизни как упорядо-

ченную последовательность событий [13]. На рис. 2 представлена диаграмма последовательности, описывающая процесс оформления заказа, согласно сформулированным в п. 2.2 требованиям к мобильному приложению. В данном процессе пользователь добавляет в корзину товары, которые он хочет включить в свой заказ. После добавления всех необходимых товаров, пользователь нажимает кнопку «Оформить заказ».

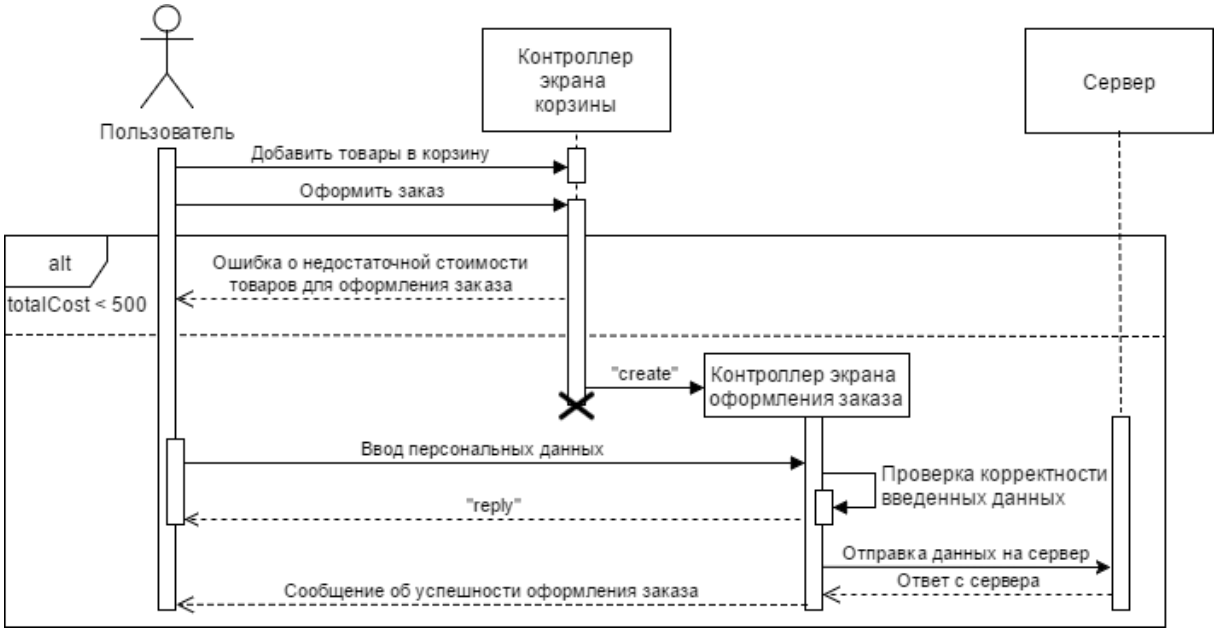

Рис. 2. Диаграмма последовательности оформления заказа

Если в момент нажатия кнопки «Оформить заказ» общая сумма товаров в корзине менее 500 рублей, приложение уведомляет пользователя, что оформить заказ можно, только добавив товаров на сумму не менее 500 рублей. Если в корзине находятся товары на достаточную сумму, осуществляется переход на экран оформления заказа, где пользователь вводит персональные данные – ФИО, e-mail, мобильный телефон, дату, к которой заказ должен быть сформирован, и, по желанию, комментарий. После ввода пользователем данных происходит их проверка на корректность: ФИО должно быть непустым, e-mail и номер телефона должны соответствовать стандартному формату, а указанное время должно быть в интервале ближайших двух дней в период работы магазина. Если хотя бы одно поле содержит некор-

ректные данные, приложение уведомит пользователя о необходимости исправить эту информацию. Если все введенные данные являются корректными, приложение отправляет информацию о заказе на сервер, получает ответ от сервера об успешности оформления заказа и показывает соответствующее уведомление пользователю.

2.3. Разработка диаграммы деятельности

На рис. 3 представлена диаграмма деятельности мобильного приложения в каталоге при добавлении товара в корзину.

Рис. 3. Диаграмма деятельности при добавлении товара в корзину

Начальным состоянием системы является запуск мобильного приложения. При запуске приложения пользователю показывается экран общего каталога. Нажав на любую категорию из списка на экране общего каталога, пользователь переходит на экран каталога категории, где может добавить

товар в корзину. При добавлении товара в корзину, в системе должен быть выбран город и магазин. Если на момент добавления в системе не выбран город, то она предложит выбрать город, а затем магазин, после чего произойдет добавление товара. Если город выбран, но не выбран магазин, то система при включенной геопозиции сама определит ближайший к пользователю магазин, в котором есть выбранный товар, иначе предложит пользователю выбрать магазин, после чего произойдет добавление товара в корзину. Система может перейти в конечное состояние в любой момент деятельности, если пользователь закрыл приложение.

Процесс оформления заказа описан диаграммой последовательности оформления заказа, которая представлена на рис. 2 в п. 2.2 работы.

Таким образом, нами были определены функциональные и нефункциональные требования к мобильному приложению, на их основе была построена диаграмма прецедентов. Кроме этого была построена диаграмма последовательности оформления заказа и диаграмма деятельности при добавлении товара в корзину.

3. РЕАЛИЗАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ФОРМИРОВАНИЯ И ОФОРМЛЕНИЯ ЗАКАЗОВ НА ПРОДУКЦИЮ В РОЗНИЧНОЙ СЕТИ МАГАЗИНОВ

3.1. Методология разработки и структура проекта

Для разработки мобильного приложения была выбрана инкрементальная модель разработки программного обеспечения.

В инкрементной модели полные требования к системе делятся на различные сборки. Терминология часто используется для описания поэтапной сборки ПО. Имеют место несколько циклов разработки, и вместе они составляют жизненный цикл «мульти-водопад». Цикл разделен на более мелкие легко создаваемые модули. Каждый модуль проходит через фазы определения требований, проектирования, кодирования, внедрения и тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не будет создана полная система [17, 18, 19].

Данная методология позволяет быстро выпустить первую версию мобильного приложения и легко добавлять новый функционал, а также адекватно рассчитывать трудозатраты и стоимость разработки.

При реализации проекта на первом этапе было выпущена первая версия, включавшая группу экранов «каталог», предназначенную для просмотра информации о товарах.

На последующих этапах последовательно добавлялись: группа экранов «магазины» для отображения информации о магазинах и возможности выбора одного из них, группа экранов «корзина» для возможности формирования списка товаров и оформления заказа, включающего эти товары, а также добавление небольших функциональных возможностей в уже существующие модули.

Проект мобильного приложения содержит 5 основных компонентов, взаимодействие которых представлено на рис. 4.

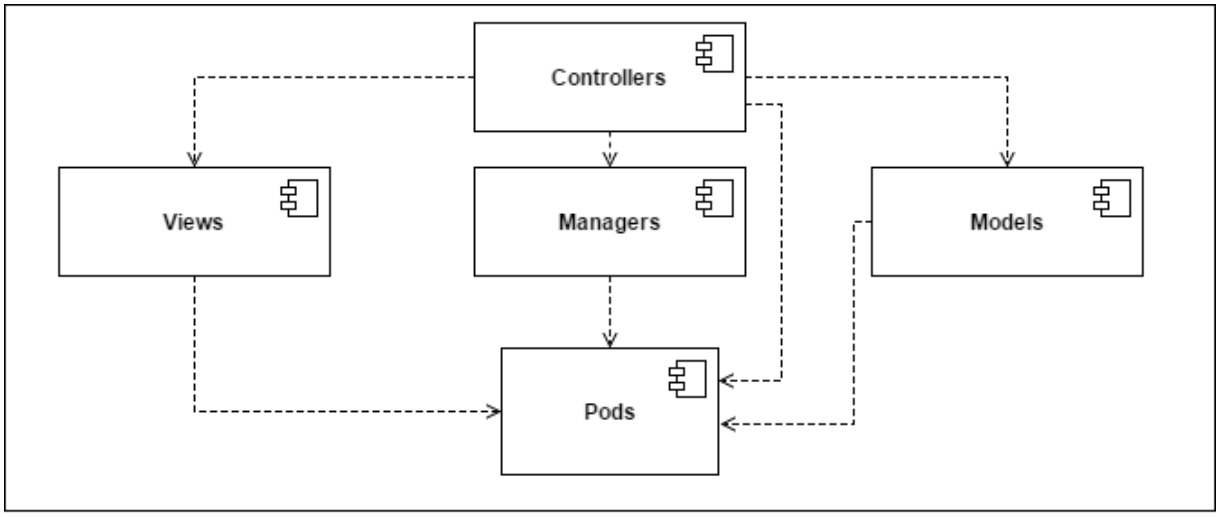

Рис. 4. Диаграмма компонентов

Проект содержит 5 компонентов – Models, Controllers, Views, Managers и Pods. Компонент Controllers содержит все классы контроллеров системы. Контроллеры реализуют логику экранов мобильного приложения.

Компонент Views содержит классы отдельных элементов экрана мобильного приложения (ячейка таблицы, поле поиска и т.д.). Часть логики, связанная с этими элементами, перенесена в их классы, чтобы разгрузить классы контроллеров.

Компонент Models включает в себя реализацию классов моделей. Каждый класс, хранимый в каталоге Models, реализует отдельную структуру данных.

Компонент Managers содержит классы для связи мобильного приложения с сервером. Для удобства связь отдельного модуля приложения реализована в классе отдельного менеджера.

Компонент Pods включает в себя все сторонние библиотеки, подключенные в проекте.

Также стоит отметить файлы, которые нельзя отнести к конкретному компоненту, но они являются неотъемлемой частью проекта. Файл

xcDataModel содержит описание структур данных, используемых приложением. Класс RequestManager содержит реализацию механики отправки запросов на сервер. К классу Request Manager обращаются все классы из компонента Managers. Файл storyboard содержит структуру экранов приложения и связь их друг с другом.

3.2. Архитектура системы

Все мобильное приложение можно разбить на 3 логические группы экранов приложений: каталог, корзина и магазины, навигация между которыми осуществляется посредством TabBarController – стандартным классом фреймворка Cocoa Touch, реализующим переключение между экранами с помощью нижнего меню – Tab Bar. Схема экранов системы и их взаимодействие представлена на рис. 5.

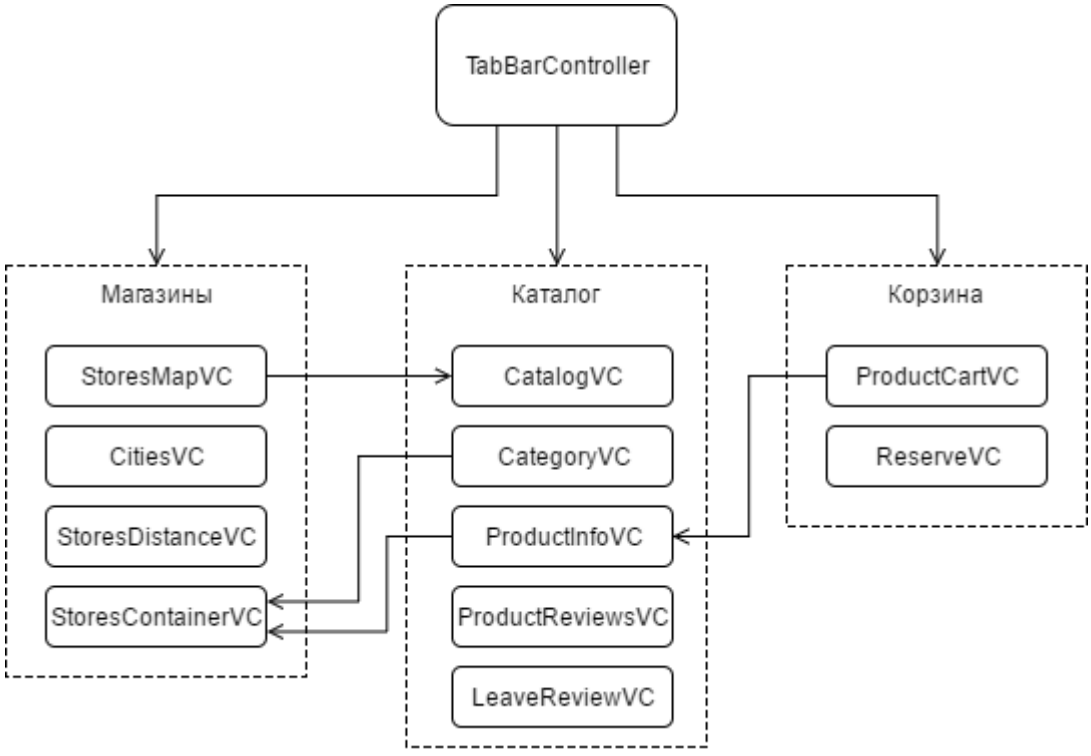

Рис. 5. Экраны системы и их взаимодействие

Каждый экран мобильного приложения реализуется отдельным классом UIViewController из компонента Controllers.

Модуль «Каталог» содержит экраны общего каталога (CatalogVC), каталога категории (CategoryVC), карточки товара (ProductInfoVC), просмотра

отзыва о товаре (ProductReviewsVC) и оставления отзыва о товаре (LeaveReviewVC). Модуль «Магазины» содержит экраны списка магазинов (StoresDistanceVC), карты с магазинами (StoresMapVC), списка городов (CitiesVC) и вспомогательный класс – StoresContainerVC, который реализует механику переключения экранов списка магазинов и карты с помощью специального элемента – Segmented Control. Модуль «Корзина» содержит экран корзины (ProductCartVC) и экран оформления заказа (ReserveVC).

Помимо основного класса-контроллера, каждый экран может быть дополнительно разбит на несколько вспомогательных классов из компонента Views, которые реализуют некоторую часть интерфейса экрана и бизнес-логику, связанную с этой частью. Так, например, экран списка магазинов помимо основного класса StoresDistanceVC также реализуется с помощью класса ячейки списка (StoresDistanceCell) и заголовка списка (StoresDistanceCell). Класс ячейки содержит настройку элементов отображения информации о магазине, а класс заголовка содержит настройку элементов отображения выбранного города и обработку нажатий на название города и кнопку отмены выбора города.

3.3. Модель внутреннего хранилища данных приложения

На рис. 6 представлена схема модели внутреннего хранилища данных в разрабатываемом мобильном приложении.

Система содержит 6 различных структур данных – User, Product, Category, Store, City и ProductCart.

User и ProductCart являются локальными структурами данных приложения, данные, хранящиеся в этих структурах, на сервере отсутствуют. Также локальным является поле countInCart структуры Product. Все остальные данные актуализируются с сервера каждый раз при попадании на экран, который отображает эту информацию. Поля products моделей Category и ProductCart представляют собой массив моделей Product. Поле cityId модели Store хранит лишь id города, но не саму модель. Вся структура хранилища

описана в файле xcdatamodel, каждый класс из компонента Models реализует отдельную модель данных.

Сформированный пользователем заказ отправляется на сервер и не хранится во внутреннем хранилище данных. Полученные с сервера и отправленные пользователем на сервер отзывы о товарах также не хранятся во внутреннем хранилище данных.

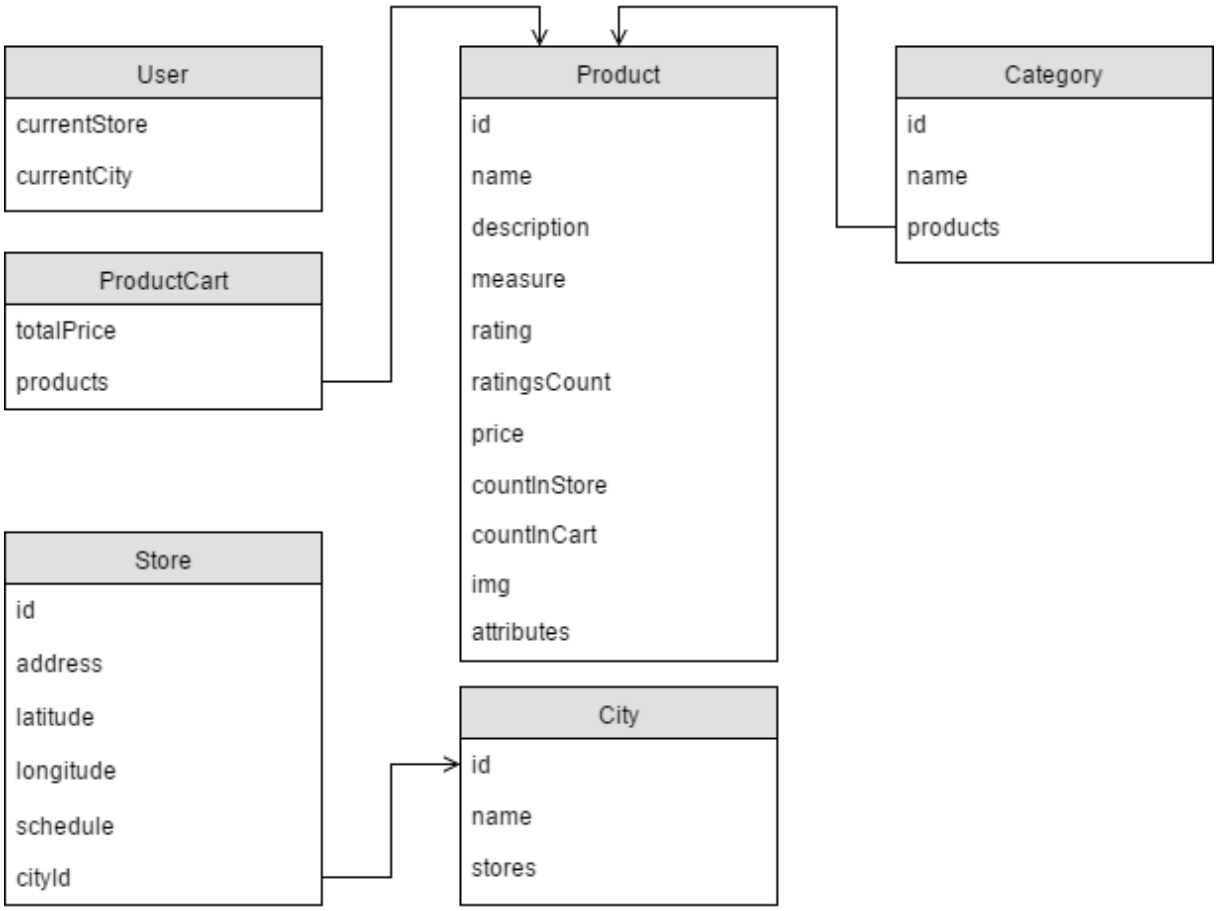

Рис. 6. Схема модели внутреннего хранилища данных приложения

3.4. Особенности реализации приложения

Мной были изучены особенности разработки iOS-приложений [3, 4, 21].

На рис. 7 представлен метод обработки результатов сканирования штрихкода товара. Метод начинает выполняться только когда камера устройства распознала штрихкод.

На рис. 8 изображен метод отправки запросов на сервер, получение ответа с сервера и его обработка. Связь с сервером реализована с помощью архитектурного стиля REST [7].

```
-(void)captureOutput:(AVCaptureOutput *)captureOutput didOutputMetadataObjects:(NSArray *)metadataObjects fromConnection:
    (AVCaptureConnection *)connection {
    if (metadataObjects == nil || [metadataObjects count] == 0) {
        return;
    }
    AVMetadataMachineReadableCodeObject *metadataObj = [metadataObjects objectAtIndex:0];
    if ([[metadataObj type] isEqualToString:AVMetadataObjectTypeEAN13Code]) {
        [ScannerManager getProductByBarcode:[metadataObj stringValue] completion:^(Product * _Nonnull product, NSError *error) {
            if (error) {
                [NotificationManager showLocalNotificationWithMessage: error.localizedDescription andNotificationType:
                    NotificationTypeError];
            } else {
                DDCustomLogInfo(@"product id = %@", product.id);
                [self stopReading];
                self.isScannedRecently = NO;
                if (kIsIPhone) {
                    [self performSegueWithIdentifier:@"goToProductVCFromScanBarCode" sender:product];
                } else {
                    ProductInfoVC *rootVC = [self.storyboard instantiateViewControlleithIdentifier:@"productInfoPopUp"];
                    rootVC.product = product;
                    self.product = product;
                    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(goToWhereBuyWithProduct)
                        name:@"WhereBuyPressedFromPopUp" object:nil];
                    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(goToProductReviewsWithProduct)
                        name:@"ReviewsPressedFromPopUp" object:nil];
                    _productInfPopupController = [[STPopupController alloc] initWithRootViewController:rootVC];
                    [_productInfPopupController setNavigationBarHidden:YES];
                    [_productInfPopupController.backgroundView setBackgroundColor:[UIColor coloithWhite:0 alpha:0.7]];
                    [_productInfPopupController presentInViewController:self];
                    [_productInfPopupController.backgroundView addGestureRecognizer:[[UITapGestureRecognizer alloc] initWithTarget:self
                        action:@selector(backgroundViewDidTapPopUp)]];
                }
            }
        }];
    }
}
```

Рис. 7. Метод делегата класса GMSMapView

request:entity:action:mainParameter:otherParameters:completion является основным методом: принимает на вход параметры запроса и его тип (GET, POST, PUT или DELETE) и возвращает 2 параметра (error и result). Метод выполняется асинхронно, по завершению его работы вызывается блок Completion, который определяется классом, вызвавшем исходный метод.

Для формирования параметров error и result вызывается вспомогательный метод – -parseResponseRest:withError:andResult, который обрабатывает полученный ответ с сервера и записывает в переменные соответствующие значения. Если с сервера пришла ошибка либо не пришло ничего, то в переменную error будет записан соответствующий объект класса NSError, который содержит код ошибки и ее описание. В этом случае переменная result будет пустой. Если с сервера пришел корректный ответ без ошибки, в параметр result запишутся запрошенные данные, переменная error будет пустой.

```
-(void) request:(HTTPRequestType)type entity:(NSString *)entity action:(NSString *)action mainParameter:(NSString *)mainParameter
    otherParams:(NSDictionary *)params completion:(void(^)(NSDictionary *result, NSError *error))completion {
    DEFAULT_BACKGROUND_THREAD_BLOCK
        __block NSError *err;
        NSMutableString *requestURL = [[NSMutableString alloc] initWithString:serverURL];
        [requestURL appendString:@"/api/v"];
        [requestURL appendString:[NSString stringWithFormat:@"%i", apiVersion]];
        [requestURL appendString:entity];
        if (action) {
            [requestURL appendString:action];
        }
        if (mainParameter) {
            [requestURL appendString:mainParameter];
        }

        AFHTTPSessionManager *session = [AFHTTPSessionManager manager];
        session.responseSerializer.acceptableContentTypes =[NSSet setWithObject:@"text/html"];
        session.responseSerializer = [AFHTTPResponseSerializer serializer];
        session.requestSerializer.timeoutInterval = 8;
        [session.requestSerializer setValue:[self getDeviceInfoForHeader] forHTTPHeaderField:@"User-Agent"];

        if (type == GET) {
            [session GET:requestURL
              parameters:params
                progress:nil
                 success:^(NSURLSessionDataTask * _Nonnull task, id  _Nullable responseObject) {...}
                 failure:^(NSURLSessionDataTask * _Nullable task, NSError * _Nonnull error) {...}];
        }
        if (type == POST) {...}
        if (type == PUT) {...}
        if (type == DELETE) {...}
    END_THREAD_BLOCK
}
```

Рис. 8. Методы класса RequestManager

На рис. 9 представлен метод записи данных о продуктах, которые пришли в ответе с сервера, в локальное хранилище приложения. С сервера приходит массив структур типа ключ:значение, для каждой структуры вызывается метод – -initWithDictionary:, изображенный на рисунке. Метод возвращает объект класса Product, содержащий данные, переданные методу на входе.

```
+(nullable instancetype)initWithDictionary:(NSDictionary *)serverResult {
    __block Product *product;
        [MagicalRecord saveWithBlockAndWait:^(NSManagedObjectContext * _Nonnull localContext) {
            product = [Product MR_findFirstByAttribute:@"id"
                                             withValue:@([serverResult[@"productId"] intValue])
                                             inContext:localContext];
            if (!product) {
                product = [Product MR_createEntityInContext:localContext];
            }
            [product MR_importValuesForKeysWithObject:serverResult];
            product.ratingsCount = serverResult[@"rating"][@"voteCount"];
            product.rating = serverResult[@"rating"][@"rating"];
            if (!serverResult[@"quantity"]) {
                product.countInStore = @(0);
            }
            if  (serverResult[@"category"]){
                [product addCategoriesObject:[serverResult[@"category"] MR_inContext:localContext]];
            }
            if (serverResult[@"color"]) {
                product.color = [serverResult[@"color"] firstObject][@"name"];
            }
            if (serverResult[@"sugar"]) {
                product.sugar = [serverResult[@"sugar"] firstObject][@"name"];
            }
            if  (serverResult [@"country"]){
                product.country = [serverResult[@"country"] firstObject][@"name"];
            }
        }];
    return [product MR_inContext:defaultContext];
}
```

Рис. 9. Метод -initWithDictionary: класса Product

На рис. 10 представлены методы конфигурации ячейки продукта на экране каталога категории товаров.

Метод – -configureWithProduct: публичен, и вызывается в методе делегата таблицы, который находится в контроллере экрана каталога категории. Метод вызывается для каждой ячейки таблицы.

```
-(void)configureWithProduct:(RWProduct *)product forIndexPath:(NSIndexPath*)indexPath{
    self.countryFlagImageWidth.constant = product.countryFlagImage ? 22 : 0;
    self.currentProduct = product;
    self.nameLabel.text = product.name;
    self.descriptionLabel.text = [RWProduct getShortDescritionForCategoryCell:product];
    self.flagImageLeftConstraint.constant = [self.descriptionLabel.text isEqualToString:@""] ? 0 : 7;
    [self.starsView setRating:product.rating withVotes:product.ratingsCount];
    [self setOfferAndNewLabels];
    [self setLabelsWhenStoreSelected:(BOOL)[RWUser getCurrentStore]];
    [self.addToCartButton setImage:imageWithName(([product.countInCart integerValue] == 0) ?
                                                 @"icon_bucket" :
                                                 @"icon_bucketRed")
                          forState:UIControlStateNormal];
    [self setPrice];
    [self setProductImage];
    [self setCountryFlagImg];
}
```

Рис. 10. Методы конфигурации ячейки таблицы с продуктами

На рис. 11 представлены методы делегата класса GMSMapView -mapView:didTapMarker: и -mapView:markerInfoWindow:. Первый метод содержит действия при нажатии на маркер, расположенный на карте, второй возвращает экземпляр класса UIView, который покажется на экране при нажатии на маркер в случае, если первый метод вернул NO для этого маркера.

```
-(BOOL)mapView:(GMSMapView *)_mapView didTapMarker:(nonnull GMSMarker *)marker{
    if(![self.mapView.unclusteredMarkers containsObject:marker]){
        float zoom = _cameraPosition.zoom < 15 ? _cameraPosition.zoom + 1 : _cameraPosition.zoom + 1.5f;
        GMSCameraPosition *markerPosition = [GMSCameraPosition cameraWithLatitude:marker.position.latitude
                                                                        longitude:marker.position.longitude
                                                                             zoom:zoom];

        [self.mapView animateToCameraPosition:markerPosition];
        return YES;
    }
    return NO;
}


-(UIView *)mapView:(GMSMapView *)mapView markerInfoWindow:(GMSMarker *)marker {
    if ([self.metroMarkersArray containsObject:marker]) {
        [self.routeButton setHidden:YES];
    } else {
        if (Database getCurrentLocation]) {
            [self.routeButton setHidden:NO];
        }
    }
    self.selectedStore = marker.userData[@"store"];
    StoresMapMarkerView *view = marker.userData[@"view"];
    return view;
}
```

Рис. 11. Методы делегата класса GMSMapView

4. ТЕСТИРОВАНИЕ

4.1. Выбор способов тестирования системы

Из-за тесной связи логики мобильного приложения с его интерфейсом, unit-тестирование является неэффективным способом тестирования системы, т.к. не может обеспечить достаточный процент покрытия исходного кода. Для автоматизации тестирования iOS-приложения в среду разработки Xcode интегрирован фреймворк XCTest [12], который позволяет, в частности, разрабатывать UI-тесты. Для реализации UI-тестов в фреймворке используется механизм записи действий пользователя в режиме реального времени, который реализован в инструменте UI-Recorder, который также интегрирован в среду разработки. В ходе использования этого средства были выявлены следующие недостатки:

- нестабильная работа Xcode в ходе использования инструмента, аварийные завершения среды разработки наблюдались не реже, чем каждый час работы;
- сгенерированный при использовании приложения код UI-тестов содержит большое количество синтаксических ошибок, которые приходится исправлять вручную для возможности запуска тестов;
- из-за наличия связи приложения с сервером и, как следствие, реализации асинхронных методов взаимодействия, сильно усложняется автоматизированное тестирование системы при нестабильном соединении, т.к. зачастую происходит обращение к объектам, которые еще не были получены с сервера;
- написание UI-тестов вручную с использованием фреймворка – нетривиальная задача, поэтому высока вероятность допустить ошибку при реализации тестов.

В связи с этими недостатками было принято решение не использовать механизм автоматизированного UI-тестирования, а ограничиться ручным функциональным и интеграционным тестированием системы.

4.2. Функциональное тестирование

Функциональное тестирование – это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения в определенных условиях решать задачи, нужные пользователям [5]. Функциональные требования определяют, что именно делает программное обеспечение, какие задачи оно решает. Используя методологию функционального тестирования, проверим работу мобильного приложения.

Тест № 1. Корректное отображение экрана общего каталога

Входные данные: -

Цель: Проверить корректность отображения экрана общего каталога. В верхней части каталога должны быть акции, которые получаются с сервера. Ниже должен располагаться список категорий, который также получается с сервера.

Результат: тест пройден.

Тест № 2. Корректное отображение экрана каталога категории.

Входные данные: Приложение открыто на экране общего каталога.

Цель: Проверить корректность перехода на экран каталога категории и отображение экрана каталога категории. В верхнем баре должно отображаться название категории, на основной части экрана должна показываться таблица с товарами данной категории.

Результат: тест пройден.

Тест № 3. Автоматическое определение города при включенной геопозиции.

Входные данные: -

Цель: Проверить автоматический определение и выбор города, в котором находится пользователь.

Ход проведения:

1) открыть приложение;
2) перейти на экран списка магазинов;

3) проверить, что приложение не показало экран выбора города и корректно определило город, показав список магазинов данного города.

Результат: тест пройден.

Тест № 4. Корректное отображение магазинов выбранного города.

Входные данные: Приложение открыто на экране общего каталога, город выбран автоматически либо пользователем вручную, определение геопозиции включено.

Цель: Проверить корректность отображение списка магазинов, отсортированных по удалению от пользователя, а также корректное отображение магазинов на карте.

Ход проведения:

1) открыть экран списка магазинов;
2) проверить корректность сортировки магазинов по удаленности;
3) перейти на экран карты;
4) проверить отображение магазинов на карте в виде маркеров;
5) нажать на маркер и убедиться в корректности отображения информации о магазине.

Результат: тест пройден.

Тест № 5. Корректность выбора магазина.

Входные данные: Приложение открыто на экране общего каталога, город выбран автоматически либо пользователем вручную, магазин не выбран.

Цель: Проверить обновление данных о товарах при выборе магазина.

Ход проведения:

1) открыть экран категории;
2) проверить, что у товаров стоит средняя цена и не показывается количество товаров в магазине;
3) перейти на экран магазинов;
4) выбрать магазин из списка;
5) перейти обратно на экран категории;

6) убедиться, что надпись «средняя цена» исчезла, появилось количество единиц товара в магазине.

Результат: тест пройден.

Тест № 6. Корректность добавления товара в корзину без выбранного магазина.

Входные данные: Приложение открыто на экране общего каталога, город выбран автоматически либо пользователем вручную, магазин не выбран, определение геопозиции включено.

Цель: Проверить корректность добавления товара в корзину в случае, когда магазин не выбран. Должен автоматически выбраться ближайший к пользователю магазин, товар должен добавиться в корзину.

Ход проведения:

1) открыть экран категории;
2) проверить, что у товаров стоит средняя цена и не показывается количество товаров в магазине;
3) нажать на кнопку добавления товара;
4) перейти на экран магазинов;
5) убедиться в том, что выбран ближайший магазин;
6) перейти на экран корзины;
7) убедиться, что выбранный товар был добавлен в корзину.

Результат: тест пройден.

Тест № 7. Невозможность оформления заказа при общей стоимости товаров в корзине менее 500 руб.

Входные данные: Приложение открыто на экране общего каталога, город выбран, магазин выбран.

Цель: Проверить невозможность оформления заказа при общей стоимости товаров в корзине менее 500 руб.

Ход проведения:

1) открыть экран категории;
2) добавить несколько товаров на сумму менее 500 рублей;

3) перейти на экран корзины;

4) убедиться, что посчитанная общая сумма товаров верна;

5) нажать на кнопку «оформить заказ»;

6) проверить появление уведомления о недостаточной общей стоимости товаров в корзине для оформления заказа.

Тест № 8. Возможность оформления заказа пи общей стоимости товаров в корзине от 500 руб.

Входные данные: Приложение открыто на экране общего каталога, город выбран, магазин выбран.

Цель: Проверить возможность оформления заказа пи общей стоимости товаров в корзине от 500 руб.

Ход проведения:

1) открыть экран категории;

2) добавить несколько товаров на сумму от 500 рублей;

3) перейти на экран корзины;

4) убедиться, что посчитанная общая сумма товаров верна;

5) нажать на кнопку «оформить заказ»;

6) проверить, что приложение показало экран оформления заказа.

Результат: тест пройден.

Тест № 9. Корректность проверки данных при оформлении заказа.

Входные данные: Приложение открыто на экране оформлении заказа, все поля пусты.

Цель: Проверить корректность проверки данных при оформлении заказа

Ход проведения:

1) нажать на кнопку «Отправить»;

2) убедиться, что красным подсвечены все поля, кроме комментария;

3) убедиться, что приложение не дает выбрать любую дату и время, кроме часов работы магазина на текущий и следующий день;

4) выбрать корректную дату;

5) ввести некорректный телефон (менее 11 цифр);

6) ввести некорректный email;

7) нажать на кнопку «Отправить»;

8) убедиться, что поля «номер телефона» и «email» подсвечены красным;

9) ввести корректные данные;

10) нажать на кнопку «Отправить»;

11) проверить, что приложение показало экран общего каталога с уведомлением, что заказ успешно сформирован.

Результат: тест пройден.

Тест № 10. Корректность работы отзывов о товаре.

Входные данные: Приложение открыто на карточке товара

Цель: Проверить корректность работы отзывов о товаре

Ход проведения:

1) нажать на кнопку «Отзывы»;

2) убедиться, что появился экран с отзывами о товаре или экран с надписью «нет отзывов», если ранее никто не оставлял отзывы об этом товаре;

3) нажать на кнопку «Оставить отзыв»;

4) ввести некорректные данные;

5) нажать на кнопку «Отправить»;

6) убедиться, что все поля подсвечены красным;

7) ввести корректные данные;

8) нажать на кнопку «Отправить» повторно;

9) убедиться, что приложение вновь показало экран отзывов с уведомлением о том, что ваш отзыв успешно принят;

10) убедиться в том, что кнопка «Оставить отзыв» исчезла для этого товара;

11) через несколько дней убедиться, что отзыв о товаре появился в списке после прохождения модерации.

Результат: тест пройден.

Тест № 10. Корректность работы сканера штрих-кода.

Входные данные: Приложение открыто на экране сканера штрих-кода товара

Цель: Проверить корректность работы сканера штрих-кода

Ход проведения:

1) отсканировать штрихкод товара, которого нет в сети магазинов;
2) получить ошибку о том, что такого товара нет в сети магазинах;
3) нажать на кнопку «Оставить отзыв»;
4) отсканировать штрихкод товара, который есть в сети магазинов;
5) убедиться, что приложение открыл экран карточки отсканированного товара.

Результат: тест пройден.

4.3. Интеграционное тестирование

Интеграционное тестирование – полная проверка программного продукта после его сборки с целью выявления ошибок, возникающих в процессе интеграции программных модулей или компонентов [5].

Собранный проект был проверен на всех возможных различных разрешениях экранов iOS-устройств, а именно: 640x960px (iPhone 4), 640x1136px (iPhone 5S), 750x1334px (iPhone 6), 1080x1920px (iPhone 6S+) и 2048x1536px (iPad), на предмет некорректности отображения элементов интерфейса. Ошибок не было выявлено, все элементы на всех разрешениях отобразились корректно. Также была проведена проверка проекта на всех поддерживаемых приложением версиях операционной системы iOS, а именно: iOS 8, iOS 9, iOS 10. В ходе проверки на различных версиях операционной системы также не было выявлено ошибок, мобильное приложение работает корректно на всех поддерживаемых версиях ОС.

Все скриншоты приложения представлены в приложении.

ЗАКЛЮЧЕНИЕ

Роль мобильных приложений в повседневной жизни растет постоянно. Многие люди пользуются несколькими десятками приложений ежедневно. Практически каждый день выпускаются новые мобильные приложения и обновления для старых. Скачивание приложений не требует долгого времени и специальных навыков, установка также проста и понятна [15].

Целью работы являлась разработка мобильного приложения для формирования и оформления заказов на продукцию в розничной сети магазинов. Для достижения данной цели были решены следующие задачи:

1) произведена постановка задачи;
2) произведен обзор существующих аналогов;
3) изучены современные платформы и средства разработки для операционной системы iOS;
4) определены требования и спроектировано мобильное приложение;
5) реализовано мобильное приложение;
6) протестировано мобильное приложение.

Все поставленные задачи были решены, цель достигнута.

Разработанное приложение имеет перспективы дальнейшего развития. С учетом усложнения бизнес-процесса и ростом требований заказчика, возникает потребность в расширении функционала системы.

В перспективе планируется реализовать следующие возможности:

- внедрить историю заказов в приложении;
- разработать интеллектуальный алгоритм выбора магазина при добавлении нескольких товаров в корзину;
- внедрить рубрику «товары-аналоги» на экране информации о товаре.

ЛИТЕРАТУРА

1. App Store. [Электронный ресурс] URL: https://itunes.apple.com/ru/genre/ios/id36?mt=8 (дата обращения: 01.05.2017).
2. Cocoa Touch. [Электронный ресурс] URL: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html (дата обращения: 08.08.2016).
3. Hegarty P. Standford University Lecture – Developing iOS7 App. [Электронный ресурс] URL: https://www.youtube.com/watch?v=ZqKbN_C4Yvg&list=PLnOdYr35FyvhDUAIW17vo7nGfHJAyikUp (дата обращения: 11.03.2017).
4. iOS Developer Library. [Электронный ресурс] URL: https://developer.apple.com/library/ios/navigation/ (дата обращения: 11.03.2017).
5. Kaner Cem, Falk Jack, Nguyen Hung Quoc. Testing Computer Software. – USA: Wiley Computer Publishing, 1999. – 479 p.
6. Magical Record. [Электронный ресурс] URL: https://github.com/magicalpanda/MagicalRecord (дата обращения: 02.04.2017).
7. Masse Mark. Rest API Design Rulebook. – США: O'Reilly, 2012. – 116 p.
8. Model-View-Controller. [Электронный ресурс] URL: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html (дата обращения: 03.02.2017).
9. What Is Core Data? [Электронный ресурс] URL: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/ (дата обращения: 07.09.2016).
10. Xamarin. [Электронный ресурс] URL: https://www.xamarin.com/ (дата обращения: 23.02.2017).
11. Xcode в AppStore для Mac. [Электронный ресурс] URL: https://itunes.apple.com/ru/app/xcode/id497799835?mt=12 (дата обращения: 09.02.2017).

12. XCTest. [Электронный ресурс] URL: https://developer.apple.com/reference/xctest (дата обращения: 15.05.2017).

13. Арлоу Дж., Нейштадт А. UML 2 и унифицированный процесс. – М.: Символ-Плюс, 2007. – 624 р.

14. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. – СПб.: Издательство «Питер», 2003. – 432 с.

15. Для чего нужны мобильные приложения? [Электронный ресурс] URL: http://for-gsm.ru/publ/igry_i_soft/dlja_chego_nuzhny_mobil-nye_prilozhenija/4-1-0-514 (дата обращения: 03.03.2017).

16. Едадил: акции, скидки на продукты, детские товары. [Электронный ресурс] URL: https://itunes.apple.com/ru/app/едадил-акции-скидки-на-продукты-детские-товары/id700569948?mt=8 (дата обращения: 15.02.2017).

17. Закис А. RUP и другие методологии разработки ПО. Ч. 1. Сравнение методологий разработки ПО. // КомпьютерПресс, 2006. – № 3. – С. 174-175.

18. Закис А. RUP и другие методологии разработки ПО. Ч. 2. Сравнение методологий разработки ПО. // КомпьютерПресс, 2006. – № 8. – С. 158-159.

19. Закис А. RUP и другие методологии разработки ПО. Ч. 3. Сравнение методологий разработки ПО. // КомпьютерПресс, 2006. – № 11. – С. 170-173.

20. Магнит: акции и скидки. [Электронный ресурс] URL: https://itunes.apple.com/ru/app/магнит-акции-и-скидки/id881463973?mt=8 (дата обращения: 15.02.2017).

21. Марк Д., Наттинг Дж. iOS 6 SDK Разработка приложений для iPhone, iPad и iPod touch. – США: APress, 2013. – 672 р.

22. Обзор рынка мобильных приложений. [Электронный ресурс] URL: https://megamozg.ru/post/16122/ (дата обращения: 03.03.2016).

23. Пятерочка. [Электронный ресурс] URL: https://itunes.apple.com/ru/app/пятёрочка/id1174271758?mt=8 (дата обращения: 15.02.2017).

24. Сеть магазинов Семья. [Электронный ресурс] URL: https://itunes.apple.com/ru/app/сеть-магазинов-семья/id893221963?mt=8 (дата обращения: 15.02.2017).

ПРИЛОЖЕНИЕ

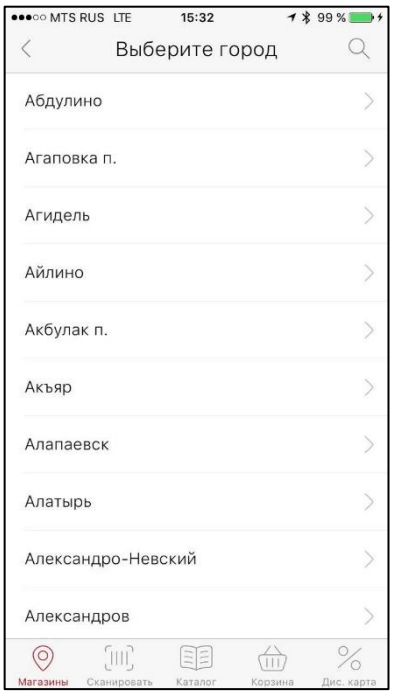

Рис. 1. Экран выбора города

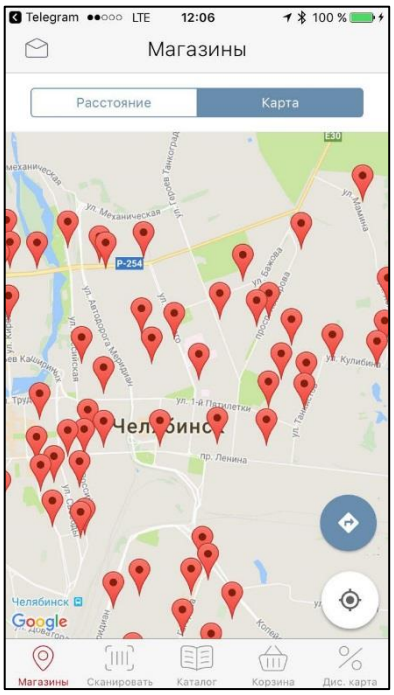

Рис. 2. Экран карты с магазинами

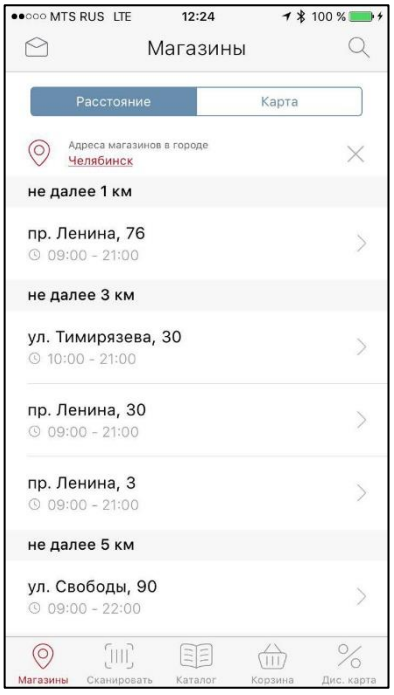

Рис. 3. Экран списка магазинов

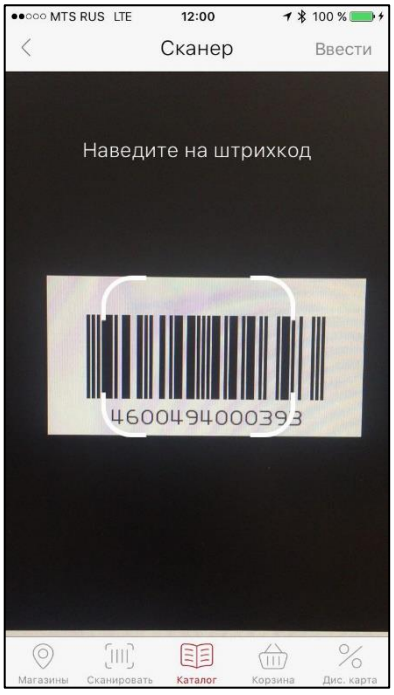

Рис. 4. Экран сканирования штрихкода товара

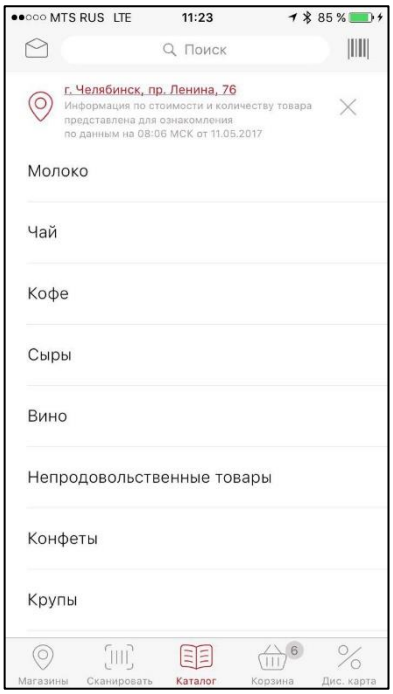

Рис. 5. Экран общего каталога

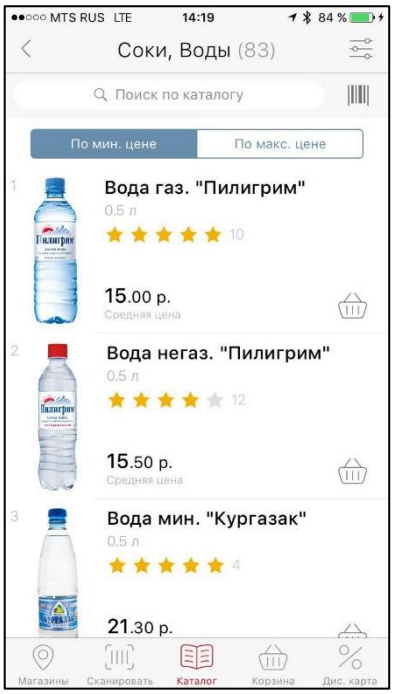

Рис. 6. Экран каталога категории

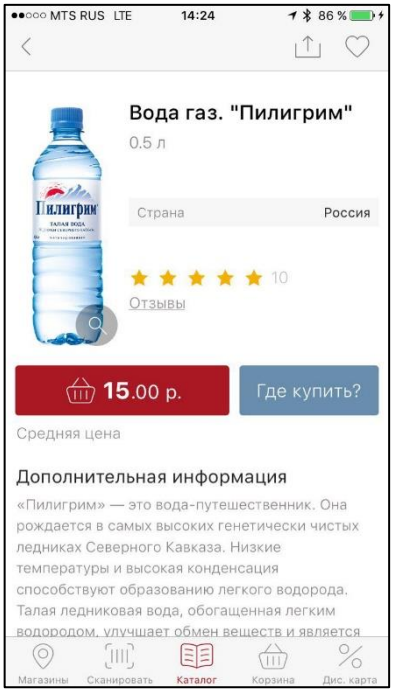

Рис. 7. Экран информации о товаре

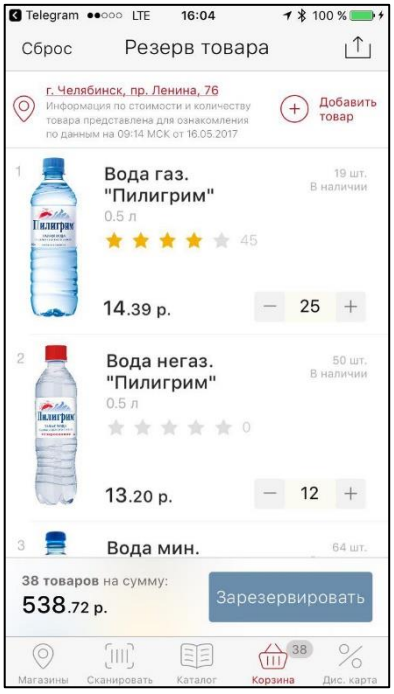

Рис. 8. Экран корзины

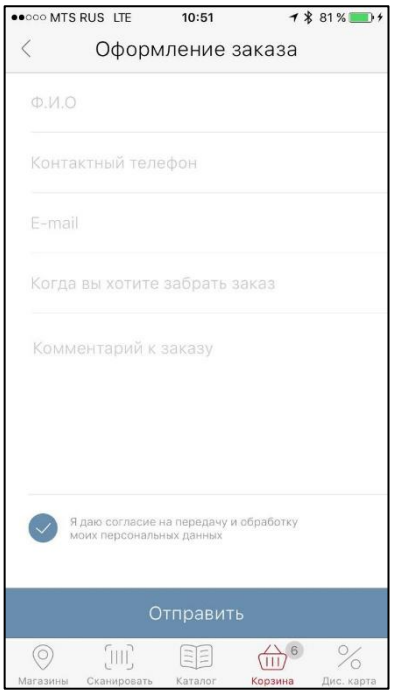

Рис. 9. Экран оформления заказа

Рис. 10. Экран отзывов о товаре

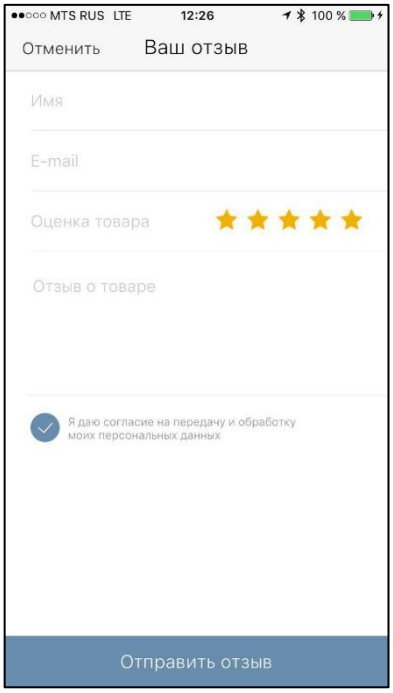

Рис. 11. Экран оставления отзыва