МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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
Технический директор
ООО «Наполеон АйТи»
_____ Е.А. Жорницкий
“___” _____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., профессор
_____ Л.Б. Соколинский
“___” _____ 2020 г.

Разработка мобильного iOS приложения для учета продуктов – MyFridge

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2020.308-048.КР

Научный руководитель,
ст. преподаватель кафедры СП
_____ К.Ю. Никольская

Автор работы,
студент группы КЭ-403
_____ М.И. Витовицкий

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2020 г.

Челябинск - 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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

УТВЕРЖДАЮ
Зав. кафедрой СП
_____
Л.Б. Соколинский
«03» февраля 2020 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Витовицкому Максиму Игоревичу
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. Тема работы (утверждена приказом ректора от 24.04.2020 № 627)
Разработка мобильного iOS приложения для учета продуктов - MyFridge
2. Срок сдачи студентом законченной работы: 02.06.2020 г.
3 .Исходные данные к работе
3.1. Усов В. Swift. Основы разработки приложений под iOS и macOS — СПб.: Издательство «Питер», 2018. — 448 с.
3.2. Харазян А. Язык Swift. Самоучитель. — СПб.: БХВ-Петербург, 2016. — 176 с.
3.3. Курс: Application Programming Interface. [Электронный ресурс]. URL: https://swiftbook.ru/courses/ (дата обращения: 15.05.2020)
4. Перечень подлежащих разработке вопросов
4.1. Произвести обзор приложений и литературы по предметной области.
4.2. Спроектировать мобильное приложение.
4.3. Реализовать мобильное приложение.
4.4. Произвести тестирование мобильного приложения.
5. Дата выдачи задания: «03» февраля 2020 г.

Научный руководитель,
ст. преподаватель кафедры СП К.Ю. Никольская

Задание принял к исполнению М.И. Витовицкий

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

АКТУАЛЬНОСТЬ ТЕМЫ

Важнейшей составляющей нашей жизни является еда.

Довольно часто мы не помним, что есть в холодильнике дома. При современном темпе жизни сложно уследить за всем, поэтому приходя в магазин мы неосознанно совершаем ненужные покупки. Нередко в любом холодильнике остаются продукты с истекшим сроком годности, который не всегда проверяется перед употреблением. Вследствие этого могут возникнуть следующие проблемы, приведённые ниже.

1. Пищевое отравление.

2. Потеря финансовых средств и др.

Данные проблемы можно избежать, если вести учет продуктов, постоянно его обновлять и расширять. Для этого можно использовать различные инструменты: обычный блокнот, заметки в электронном формате, устанавливать напоминания в телефоне с помощью различных приложений. Но все эти варианты удобны, так как все это делается вручную, занимает значительное количество времени и совершенно не подходит для ритма жизни современного человека.

Для экономии времени и удобства нам необходим «карманный менеджер», с помощью которого можно быстро и комфортно вести учет своих продуктов.

Самым быстрым и удобным для пользователя вариантом добавления продуктов является сканирование штрих-кода. Для его считывания магазины использует специальное оборудование, но зная стандарты [9], достаточно будет камеры мобильного устройства.

ЦЕЛЬ И ЗАДАЧИ

Целью работы является создание мобильного приложения на платформе iOS для ведения учета продуктов. Данное приложение должно позволять добавлять продукты с помощью сканирования штрих-кода, редактировать информацию о продукте, добавлять изображение для продукта, а также

автоматически планировать напоминания об истечении срока годности продуктов. Также для удобства использования данного приложения должна поддерживаться функция создания общих семейных групп, с помощью которых будет вестись общий список продуктов, для редактирования его всеми членами семьи. Для достижения данной цели должны быть выполнены следующие задачи, приведенные ниже.

1. Произвести обзор аналогов и предметной области.
2. Изучить работу с базой данных Firebase.
3. Спроектировать мобильное приложение.
4. Спроектировать базу данных.
5. Протестировать мобильное приложение.

СТРУКТУРА И ОБЪЕМ РАБОТЫ

Работа состоит из введения, четырех глав, заключения, библиографического списка и двух приложений. Объем работы составляет 42 страницы, объем библиографии – 15 источников, объем приложения – 2 страницы.

В главе «Теоретические основы проектирования мобильного приложения для учета продуктов» была произведена постановка задачи, проведен обзор существующих аналогов мобильных приложения для учета продуктов, а также рассмотрены современные платформы для разработки мобильных приложений.

Глава «Проектирование мобильного iOS приложения для учета продуктов» посвящена определению требований к разрабатываемому мобильному приложению, описания архитектуры и организации хранения данных на устройстве. В этой же главе описываются диаграмма прецедентов, приведена архитектура мобильного приложения.

В третьей главе, «Реализация мобильного приложения», рассмотрена реализация хранения данных, а также реализация основной функциональности мобильного приложения.

Глава «Тестирование мобильного приложения» посвящена результатам тестирования мобильного приложения. Представлены результаты функционального тестирования, выполненные в работающем приложении, и интеграционного тестирования мобильного приложения. В заключении сделаны выводы о проделанной работе.

1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПРОЕКТИРОВАНИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ УЧЕТА ПРОДУКТОВ

1.1. Постановка задачи

Мы всегда заботимся о своем бюджете и здоровье. Одна из вещей, объединяющая эти понятия – продукты, которые мы потребляем. Самостоятельно следить за наличием продуктов в холодильнике – задача несложная, но требующая внимания времени. Запомнить все продукты очень сложно, а записывать – неудобно. Но мало запомнить все продукты, нужно запомнить и их срок годности.

Часто мы попадаем в ситуации, когда за кучей продуктов обнаруживаем давно залежавшийся йогурт или пачку открытого сока. Найдя такие продукты следует немедленно избавиться от них.

Также нередко в магазине мы не можем вспомнить, что именно из продуктов у нас уже есть, а что необходимо купить. Забывая что-то купить, мы тратим свое время на повторный поход в магазин, а купив что-то лишнее – излишне тратим свои средства.

Таких ситуаций можно было бы избежать. Получив вовремя напоминание о необходимости использовать продукт мы бы не забыли его употребить. А находясь в магазине, удобно было бы всегда иметь под рукой актуальный список продуктов, которые находятся в нашем холодильнике.

Исходя из этих проблем появляется потребность в реализации приложения, которое позволит удобно и без лишних временных затрат вести учет наших продуктов.

1.2. Обзор аналогичных приложений

Обзор научной литературы по выбранной теме дал отрицательные результаты. Далее представлен результат обзора аналогичных приложений.

При анализе магазина приложений было выявлено сравнительное малое количество аналогичных приложений, что является преимуществом в виде малой конкуренции.

Холодильник в кармане от LuckyRocky [1]. Холодильник в кармане – приложения для составления покупок и эмулятор вашего холодильника. В нем вы можете создать список покупок на дверце холодильника, использовав поиск по продуктам, а также добавлять купленные товары в холодильник (рисунок 1).

Достоинства:

1. большая база продуктов;
2. красивая анимация открытия холодильника.

Недостатки:

1. нет возможности указать срок хранения продукта;
2. отсутствует адаптация под современные устройства;
3. большинство полезных функций – платные;
4. много рекламы.

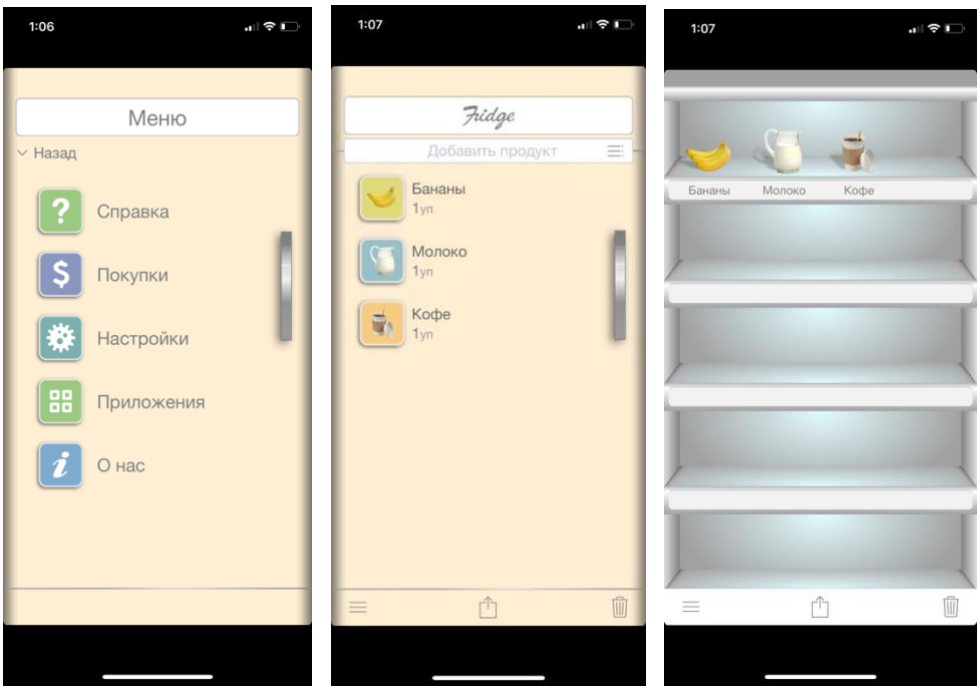

Рис. 1. Экраны мобильного приложения «Холодильник в кармане»

Мой Холодильник – приложения для учета продуктов в вашем холодильнике от разработчика OptiLife Apps [2]. Данное приложение помогает не забыть, что лежит в вашем холодильнике, легко и эффективно управлять вашими запасами (рисунок 2).

Достоинства:

1. интерфейс адаптирован под все современные устройства;

2. присутствует возможность получать уведомления о продуктах, срок годности которых подходит к концу;

3. есть возможность добавить фотографию к продукту.

Недостатки:

1. в бесплатной версии есть возможность добавить не более десяти продуктов;

2. нельзя добавить несколько одинаковых продуктов с разным сроком годности;

3. нельзя сохранить продукты, чтобы в будущем быстро добавлять их в холодильник;

4. после добавления продукта отображается реклама.

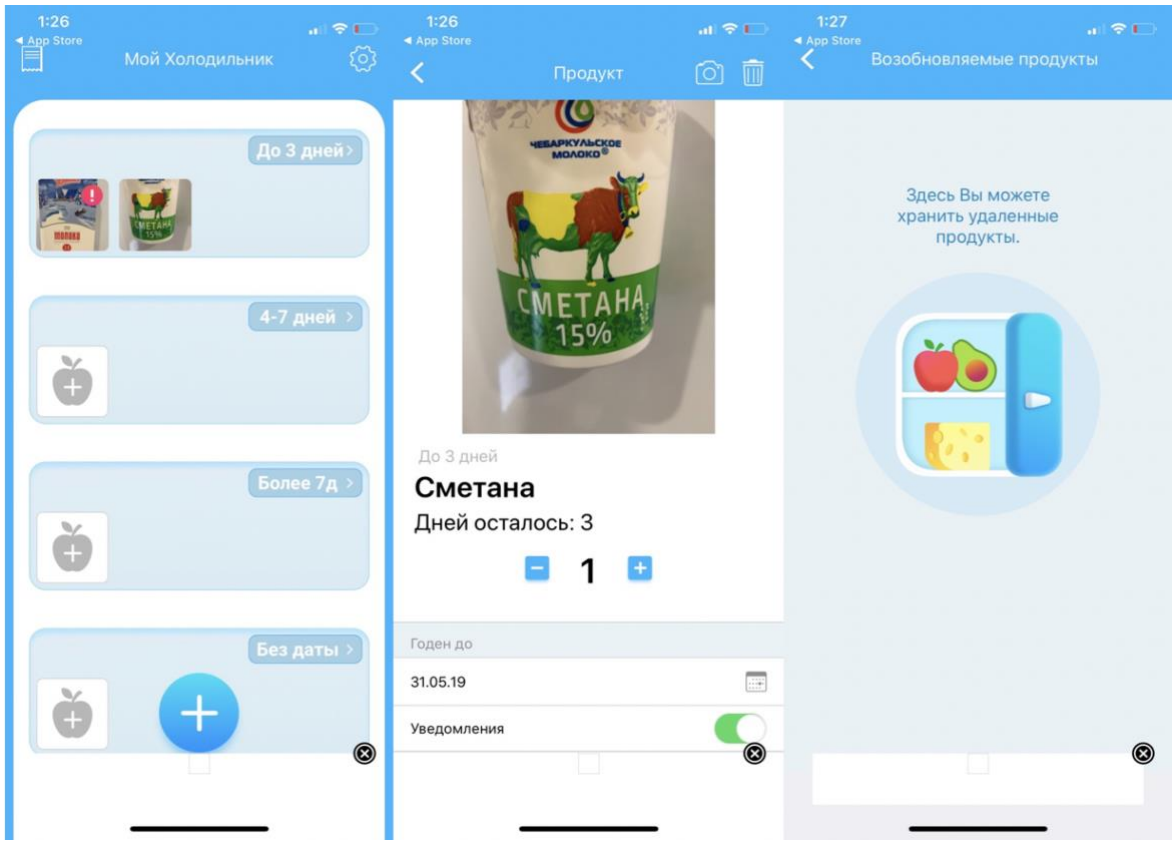

Рис. 2. Экраны мобильного приложения «Мой холодильник»

1.3. Анализ решений для реализации мобильного iOS приложения

В настоящее время существует множество подходов для разработки мобильных приложений. Все эти решения можно разделить на две большие категории: кроссплатформенные и нативные.

Кроссплатформенные решения – средства, позволяющие вести разработку одновременно для нескольких платформ. Самые популярные решения позволяют одновременно разрабатывать приложение для Android и iOS. Помимо очевидного преимущества в виде возможности одновременной разработки приложения под несколько операционных систем, данные инструменты для всех платформ используют один язык программирования, что избавляет от необходимости изучать нативные языки под каждую из платформ. Но данные решения имеют и свои минусы. Одним из самых значительных является малая гибкость в создании пользовательского интерфейса и организации навигации по приложению. К тому же, кроссплатформенные решения менее производительны, в сравнении с нативными средствами разработки, которые будут являться более практичным и удобным вариантом.

Нативные решения – официально и бесплатно распространяемыми, поддерживаемыми и актуальными средствами разработки компанией Apple являются интегрированная среда разработки Xcode [3], версии 11.5 и язык программирования Swift [4], версии 5.1. Данные решения позволяют добиться максимальной производительности, как при разработке приложения, так и при использовании приложения конечным потребителем. Для разработки пользовательского интерфейса используется система Interface Builder [5], позволяющая выстраивать пользовательский интерфейс с помощью графического редактора и последующей связи интерфейса с программной реализацией приложения, при помощи утилит IBOutlets и IBActions. Тестирование и отладка разработанных приложений возможна с помощью iPhone Simulator – решение для симуляции запуска мобильных приложений на реальном устройстве.

Также, Xcode включает в себя комплекс средств разработки приложений iOS SDK. Одной из важнейших его составляющих является фреймворк Cocoa Touch [6], основанный на архитектуре MVC – Model-View-Controller. Помимо этого, фреймворк Cocoa Touch включает в себя многие другие важные библиотеки, перечисленные ниже.

1. UIKit [11] – библиотека, предоставляющая классы для составления графического интерфейса мобильного приложения. Данная библиотека включает в себя базовые элементы для взаимодействия с пользователем: UILabel – элемент для отображения форматированного текста, UITextField – элемент для ввода пользователем текстовых данных, UIButton – элемент для взаимодействия с пользователем, по средством нажатия на данный элемент, а также множество других элементов.

2. UserNotifications – библиотека, предоставляющая возможность управлять локальными и push-уведомлениями. С помощью данного решения могут запланированы уведомления для пользователя, с возможностью их повторения.

3. Foundation Kit [12] – данная библиотека предоставляет базовые типы, такие как: целые числа, числа с плавающей точкой, строки, массивы, списки и др. Помимо этого присутствует реализация более сложных элементов: даты, файловое хранилище, обработка URL адресов, многопоточность, таймеры и др.

Вывод по главе 1

Проведя сравнение и анализ различных средств разработки было принято решение использовать нативную IDE Xcode и язык Swift. Данные решения полностью удовлетворяют требованиям удобства использования и производительности как со стороны разработки, так и со стороны конечного продукта.

Также для полноценной работы функции семейного доступа в приложении необходима реализация облачной базы данных. В качестве

реализации серверной части может быть использовано множество средств, каждое из которых обладает как своими положительными качествами, так и отрицательными. В данной работе был выбран сервис Firebase [7], предоставляющий готовые решения как для удаленного файлового хранилища, так и облачной базы данных, обновление и редактирование данных в которой происходит в режиме реального времени.

2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ УЧЕТА ПРОДУКТОВ

2.1. Функциональные и нефункциональные требования системы

Функциональные требования системы описывает функционал программного обеспечения и описывает поведение, которое должна предоставлять данная система.

1. Мобильное приложение должно иметь возможность добавлять продукты с помощью сканирования штрих-кода продукта.

2. Мобильное приложение должно иметь возможность редактировать срок годности продуктов.

3. Мобильное приложение должно иметь возможность добавлять изображение для продукта.

4. Мобильное приложение должно уведомлять пользователя о продуктах, срок годности которых подходит к концу.

5. Мобильное приложение должно иметь возможность создавать шаблоны продуктов. При последующем добавлении продукта с тем же штрих-кодом, что и у шаблона, информация о продукте должна автоматически заполняться информацией из шаблона.

6. Мобильное приложение должно иметь возможность создавать общие семейные группы, для ведения одно общего списка продуктов для всех членов семьи.

7. Мобильное приложение должно иметь возможность просмотра актуального списка продуктов.

8. Мобильное приложение должно иметь возможность просмотра подробной информации о продукте.

9. Мобильное приложение должно иметь возможность удалить продукт

Нефункциональные требования описывают свойства и ограничения, накладываемые на систему.

1. Приложение должно быть написано под систему iOS.

2. Приложение должно быть реализовано с помощью языка программирования Swift, а также интегрированной среды разработки Xcode.

3. Для создания пользовательского интерфейса приложение должно использовать Interface Builder.

4. Приложение должно иметь возможность отправки уведомлений.

2.2. Варианты использования системы

Для построения модели взаимодействия внешнего актера с мобильным приложением и проектирования мобильного приложения был использован язык графического описания UML [8, 14].

В ходе проектирования был выявлен единственный актер, взаимодействующий с системой – пользователь. Ему доступны все варианты использования приложения. На рисунке 3 приведены варианты использования приложения для пользователя.

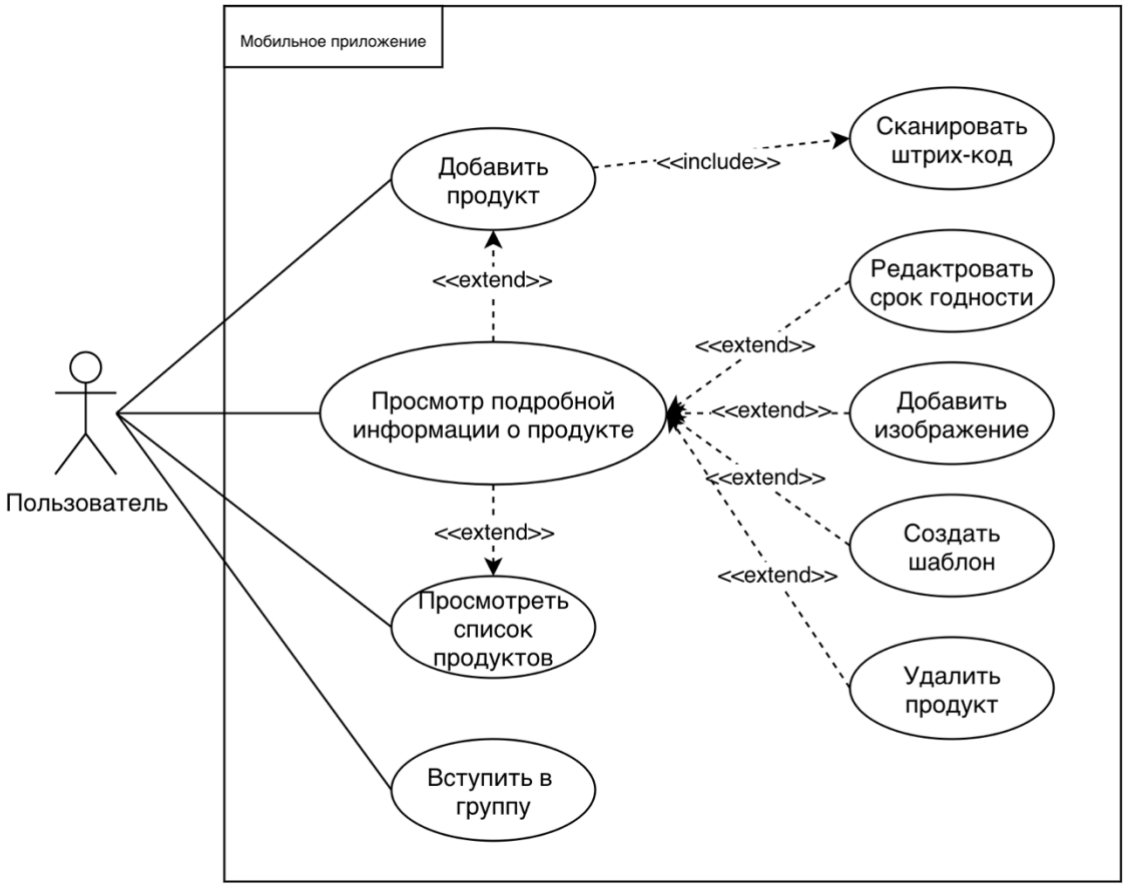

Рис. 3. Диаграмма вариантов использования мобильного приложения

Данный пользователь может совершать действия, описанные ниже.

1. Добавить продукт – с помощью сканера штрих-кодов пользователь может отсканировать новый продукт и добавить его в список продуктов для сохранения. После того, как пользователь добавил и заполнил информацию обо всех продуктах, пользователь их сохраняет, и они попадают в общий список продуктов.

2. Сканировать штрих-код – данное действие выполняется как часть действия добавления продукта. Штрих-код сканируется с помощью камеры мобильного устройства, пользователю достаточно навести на штрих-код, и он автоматически считается. После успешного считывания штрих-кода экран сканирования сообщает своему делегату отсканированный штрих-код и закрывается.

3. Просмотр подробной информации о продукте – пользователь может просмотреть информацию о продукте, нажав на продукт в списке. После нажатия откроется карточка продукта с информацией о продукте.

4. Просмотреть список продуктов – открывая приложение, пользователь видит список продуктов на главном экране и может их просмотреть. В списке продуктов отображается основная информация: изображение, название и дата истечения срока годности.

5. Вступить в группу – пользователь может ввести код-приглашение для подключения к семейной группе. После этого его список продуктов будет синхронизирован с другими пользователями, состоящий в этой группе. Так же пользователь имеет возможность сам пригласить к себе в группу других пользователей. На экране вступления в группу отображается код-приглашение, нажав на который он может его скопировать, а нажав на кнопку рядом с кодом – поделиться, с помощью стандартного меню.

6. Редактировать срок годности – просматривая карточку продукта, пользователь имеет возможность отредактировать его срок годности. Срок годности состоит из двух полей: дата производства и дата истечения срока

годности. Нажимая на одно из полей, для пользователя отображается экран с выбором даты, после чего выбранная дата сохраняется.

7. Добавить изображение – просматривая карточку продукта, пользователь имеет возможность добавить изображение к продукту. После нажатия на кнопку добавления изображения отображается камера. Пользователь делает фотографию продукта и может кадрировать. После нажатия на кнопку сохранения, изображение для продукта загружается на сервер и становится общим для всех пользователей в группе.

8. Создать шаблон – просматривая карточку продукта, пользователь имеет возможность сохранить продукт как шаблон. Шаблон необходим для автоматического заполнения информации о продуктах, которые будут добавляться в будущем.

3. РЕАЛИЗАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

3.1. Средства разработки

Для реализации приложения учета продуктов был использован высокоуровневый язык программирования Swift за свою интуитивность в понимании и множества удобных синтаксических решений.

В качестве среды разработки был выбрал Xcode. На данный момент это единственный способ использовать Interface Builder для визуального создания пользовательского интерфейса.

3.2. Компоненты системы

Для реализации приложения был выбран архитектурный паттерн MVC. Данная архитектура предлагает разделять приложения на три обособленных слоя, приведённых ниже.

1. Model – сущность, отвечающая за формирование моделей данных в приложении. Так же данная структура отвечает за способы работы с данными, может содержать логику их валидации или форматирования. Такие модели могут храниться как в локальной базе данных, так и в памяти приложения.

2. View – представление в мобильном приложении. Представляет собой слой пользовательского интерфейса для взаимодействия с пользователем. Представление получает данных из контроллера и отображает данные. При правильном подходе оно никак не должно влиять на измерения данных и другие слои в приложении, а лишь передавать контроллеру пользовательские события, такие как ввод текста, нажатие на кнопки и другие действия пользователя, который в свою очередь взаимодействует с моделью.

3. Controller – связующее звено между представлением и модельным слоем. Контроллер отвечает за обработку пользовательских событий, полученных из слоя отображения и маршрутизацию пользовательских запросов. Как пример, в данном слое может находиться валидация данных, введённых пользователем, перед передачей этих данных в модель. В случае, если данные из слоя пришли в неверном формате или в недостаточном

количестве, контроллер должен обратиться к представлению с запросом на отображение визуальной ошибки.

Диаграмма связей слоёв архитектуры MVC приведена на рисунке 4.

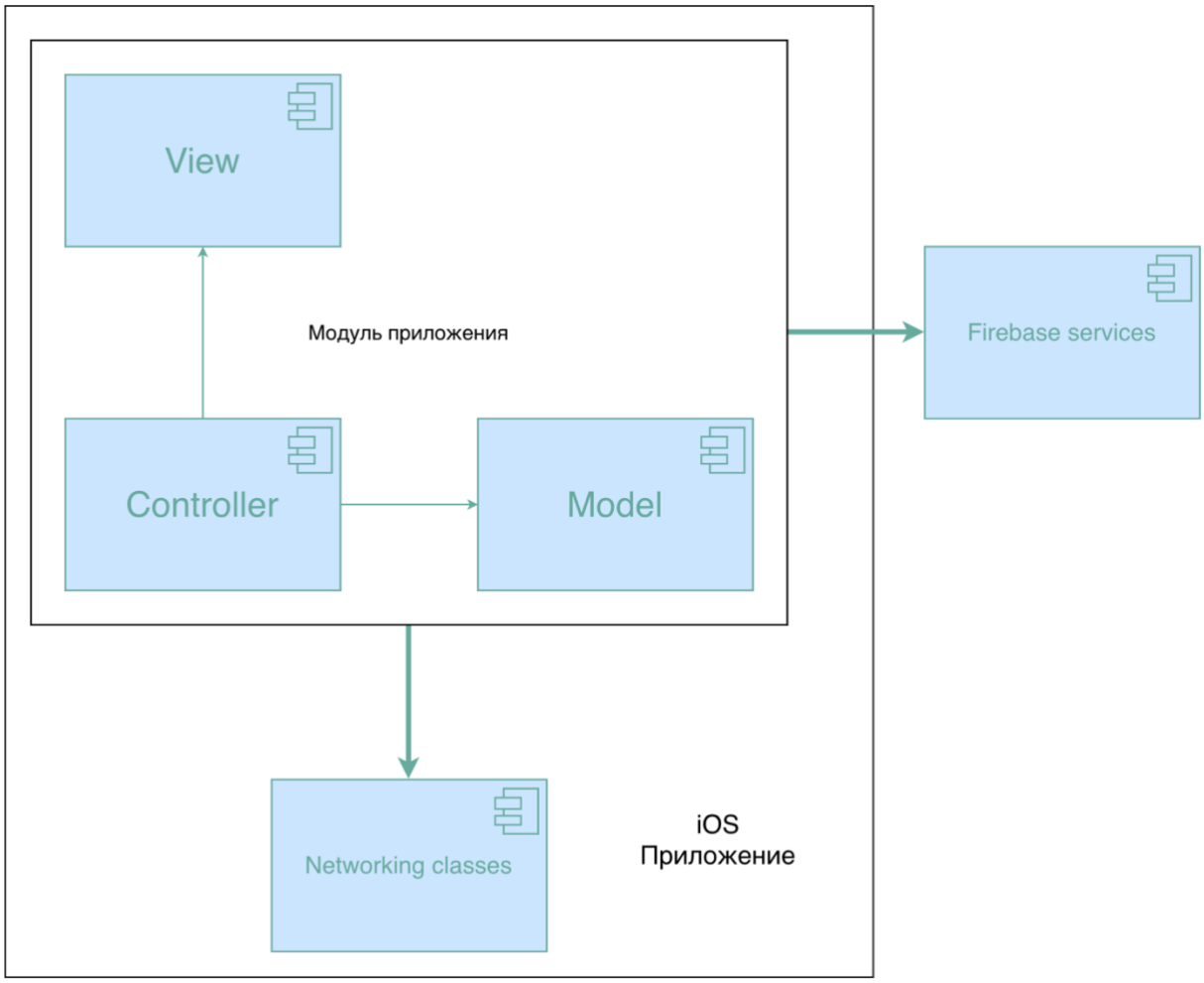

Рис. 4. Диаграмма слоёв архитектуры MVC

Одним из главных преимуществ архитектуры MVC является четкое разделение приложение на несколько слоёв со своей зоной ответственности. Каждый слой знает только про свои обязанности и связан с другими слоями с помощью интерфейсов. Это позволяет изолировать слои друг от друга и в случае необходимости бесшовно подменять реализации того или иного слоя.

Помимо разделения ответственности между слоями, данный подход упрощает и тестирование приложения. Можно подменить модельный слой и получать данные не с сервера или базы данных, а создавать тестовые модели

и тестировать отображение моделей для различных вариаций составление данных моделей. Основные контроллеры из разработанного мобильного приложения приведены ниже.

1. ProductsListVC – данный контроллер является стартовым экраном в приложении. На данном экране пользователь имеет возможность просмотреть все продукты, добавленные в список на текущий момент. В случае, если продуктов в списке нет, для пользователя отображается небольшое информационное окно, где предлагается начать пользоваться приложением и добавить продукты в список. Так же, с данного экрана пользователь может открыть контроллер с карточкой продукта.

2. AddProductVC – контроллер, позволяющий добавлять новые продукты в приложение. На данном экране пользователь может нажать на иконку добавления продукта, после чего откроется контроллер сканирования штрих-кодов. После успешного сканирования продукт добавляется в список продуктов для сохранения. С данного контроллера так же имеется возможность открыть контроллер с карточкой продукта.

3. BarcodeScannerVC – контроллер, выполняющей функцию сканирования штрих-кодов. Имеет входящий параметр, в который по итогу сканирования передаётся информация из штрих-кода в виде строки.

4. ProductDetailsVC – контроллер, с помощью которого пользователь может просмотреть детальную информацию о продукте, а также отредактировать данную информацию и загрузить изображение для продукта.

5. GroupController – контроллер, содержащий информацию о текущей пользовательской группе. На данном экране расположен код-приглашение в семейную группу, который будет скопирован в буфер обмена по нажатию. Помимо возможности поделиться кодом-приглашением присутствует возможность ввести код, полученный от другого пользователя, вступив при этом с ним в общую группу.

Помимо основных контроллеров присутствуют несколько дополнительных, для организации навигации по приложению. Корневым

контроллером приложения является UITabBarController. Он служит контейнером для всех основных контроллеров приложения и создаёт базовую навигацию. Для каждого контроллера создаётся отдельная вкладка в нижнем навигационном меню приложения, состоящая из иконки и текста, по нажатию на которую пользователь перемещается к соответствующему контроллеру.

3.3. Реализация моделей данных

Слой моделей данных представляет собой классы для сущностей, с которыми работает приложение. Модели хранятся в памяти приложения и обновляются в реальном времени. При запуске приложения происходит загрузка данных и формирование моделей из данных. Каждая из сущностей реализует протокол Codable, встроенный в стандартный фреймворк Foundation. Данный протокол позволяет автоматически декодировать ответ, полученный от сервера и преобразовывать словари в модели классов. Так же данный протокол позволяет выполнить обратное преобразование – из моделей создать словарь, состоящий из пары ключ и значения, для передачи данных на сервер. Ниже приведено описание и реализация моделей, с которыми взаимодействует приложение.

Product – модель продукта. Содержит в себе все необходимые поля для описания продукта, а также функции, реализующие возможность редактирования информации о продукте. Соответствует протоколу Codable, что позволяет инициализировать данную сущность из полученных данных с сервера в формате JSON, а также преобразовывать модель в формат JSON для передачи данной модели на сервер.

Помимо инициализации модели продукта из данных, полученных с сервера, продукт возможно инициализировать с помощью штрих-кода в формате строки. Тогда создастся продукт с данным штрих-кодом, а все остальные поля продукта будут заполнены стандартными данными для этих полей. Реализация модели Product приведена на рисунке 5.

```
class Product: Codable {

    var id: String { "\(barcode)_\(timestamp)" }

    var barcode: String
    var timestamp: Int
    var name: String
    var productionDate: String
    var shelfLifeEnd: String

    init(barcode: String) {
        let date = Date()

        self.barcode = barcode
        self.timestamp = Int(date.timeIntervalSince1970)
        self.name = "# Новый продукт"
        self.productionDate = date.string(.general)
        self.shelfLifeEnd = date.byAddingDays(3).string(.general)
    }

    init(template: ProductTemplate) {
        let date = Date()

        self.barcode = template.barcode
        self.timestamp = Int(date.timeIntervalSince1970)
        self.name = template.name
        self.productionDate = date.string(.general)
        self.shelfLifeEnd =
date.byAddingDays(template.storageDays).string(.general)
    }
}
```

Рис. 5. Реализация модели Product

Основные функции для редактирования модели Product приведены ниже.

1. AddImage – принимает в качестве параметра изображение и выполняет сохранение данного изображения на сервере.

2. GetImage – загружает изображение с сервера. Внутри данного метода реализовано кеширование изображения, во избежание излишней траты пакета

интернета пользователя, ускорения загрузки и уменьшения нагрузки на сервер.

3. SetName – устанавливает имя для товара и сохраняет товар с изменённым именем на сервере.

4. SetProductionDate – устанавливает дату изготовления продукта. В данном методе присутствует логика валидации срока годности: если пользователь ввёл дату изготовления товара, которая позже даты конца срока годности, то дата конца срока годности автоматически изменится. К тому же, для уменьшения действий, которые необходимо выполнить пользователю, дата окончания срока годности увеличивается пропорционально изменению даты срока изготовления.

5. SetShelfLIfeEndDate – устанавливает дату окончания срока годности продукта.

Реализация функций редактирования продукта приведена на рисунке 6.

Для удобства сравнения и сортировки модель Product реализует протокол Comparable, входящий в состав фреймворка Foundation. Реализация данного протокола приведена на рисунке 7.

ProductTemplate – модель шаблона продукта. Необходим для автоматического заполнения информации в продукте при его добавлении. При добавлении продукта выполняется поиск по всем шаблонам и в случае совпадения штрих-кодов, информация из шаблона переносится в сам продукт.

Шаблон продукта создаётся на основе модели продукта и в инициализаторе заполняется вся основная информация.

```
extension Product {
    func addImage(_ image: UIImage, completion: @escaping (Bool) -> Void) {
        FirebaseStorage.shared.saveImage(image, for: self, completion:
completion)
    }
    func getImage(completion: @escaping (UIImage?) -> Void) {
        FirebaseStorage.shared.getImage(for: self, completion: completion)
    }
    func setName(_ name: String) {
        self.name = name
        ProductsManager.shared.synchronizeProduct(self)
    }
    func setProductionDate(_ date: Date) {
        productionDate = date.string(.general)

        if date > shelfLifeEnd.date() {
            shelfLifeEnd = date.byAddingDays(1).string(.general)
        }
        if let template = TemplatesManager.shared.template(for: self.barcode)
{
            shelfLifeEnd =
date.byAddingDays(template.storageDays).string(.general)
        }
        ProductsManager.shared.synchronizeProduct(self)
    }
    func setShelfLifeEndDate(_ date: Date) {
        shelfLifeEnd = date.string(.general)
        if date < productionDate.date() {
            if let template = TemplatesManager.shared.template(for:
self.barcode) {
                productionDate = date.byAddingDays(-
template.storageDays).string(.general)
            } else {
                productionDate = date.byAddingDays(-1).string(.general)
            }
        }
        ProductsManager.shared.synchronizeProduct(self)
    }
}
```

Рис. 6. Реализация функций модели Product

```
extension Product: Comparable {

    static func < (lhs: Product, rhs: Product) -> Bool {
        lhs.timestamp < rhs.timestamp
    }

    static func == (lhs: Product, rhs: Product) -> Bool {
        lhs.barcode == rhs.barcode && lhs.timestamp == rhs.timestamp
    }
}
```

Рис. 7. Реализация протокола Comparable моделью Product

Ниже приведён список полей, содержащихся в шаблоне продукта.

1. Name – имя продукта, текстовая строка, не может быть пустым, не имеет ограничений по размеру.

2. Barcode – штрих-код продукта, не может быть пустым. Является идентификатором шаблона и уникальным ключом.

3. StorageDays – срок хранения продукта в днях, не может быть пустым. Рассчитывается, исходя из разницы даты производства и окончания срока годности продукта.

Реализация шаблона продукта приведена на рисунке 8.

```
class ProductTemplate: Codable {
    var name: String
    var barcode: String
    var storageDays: Day = 7
    init(product: Product) {
        let producationDate = product.productionDate.date()
        let shelfLifeEnd = product.shelfLifeEnd.date()
        let components = Calendar.current.dateComponents([.day], from:
producationDate, to: shelfLifeEnd)
        name = product.name
        barcode = product.barcode
        storageDays = components.day!
    }
}
```

Рис. 8. Реализация ProductTemplate

Для удобства поиска шаблона продукта и сравнения с моделью продукта реализован протокол Equtable, входящий в состав фреймворка Foundation, который позволяет сравнивать шаблоны между собой, а также сравнивать шаблон продукта и модель продукта. Реализация протокола Equtable приведена на рисунке 9.

```
extension ProductTemplate: Equatable {
    static func == (lhs: ProductTemplate, rhs: ProductTemplate) -> Bool {
        lhs.barcode == rhs.barcode
    }
    static func == (lhs: ProductTemplate, rhs: Product) -> Bool {
        lhs.barcode == rhs.barcode
    }
    static func == (lhs: Product, rhs: ProductTemplate) -> Bool {
        lhs.barcode == rhs.barcode
    }
}
```

Рис. 9. Реализация протокола Equtable классом ProductTemplate

3.4. Реализация контроллеров

В контроллерах содержится вся основная логика навигации в приложении и реакции мобильного приложения на действия пользователя. Контроллер получает от представления различные события: вводимые пользователем данные, нажатия и жесты. После получения определённых действий с отображения, контроллер запускает соответствующую бизнес логику приложения.

Основным контроллером и первым котроллером, который пользователь видит в приложении является ProductsVC. На данном контроллере пользователь видит добавленные им продукты в список продуктов. Если же список продуктов пользователя пустой, пользователь увидит информационное сообщение, призывающего его начать пользоваться приложением и добавить продукты.

Основным элементом отображения данного контроллера является коллекция – UICollectionView. Взаимодействие с данным видом отображения реализовано по средством делегирования – отображение делегирует контроллеру обязанности источника данных и обработчика действий пользователя. Для возможности принять на себя данную ответственность, контроллер ProductsVC расширяется и реализует два протокола – UICollectionViewDelegate и UICollectionViewDataSource.

Реализация данных протоколов представлена на рисунке 10.

```
extension ProductsVC: UICollectionViewDataSource {


    func collectionView(_ collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int {
        return products.count
    }


    func collectionView(_ collectionView: UICollectionView, cellForItemAt
indexPath: IndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
ProductCollectionCell.reuseID, for: indexPath) as! ProductCollectionCell
        let product = products[indexPath.row]


        cell.setup(withProduct: product)


        return cell
    }


}
extension ProductsVC: UICollectionViewDelegate {
    func collectionView(_ collectionView: UICollectionView, didSelectItemAt
indexPath: IndexPath) {
        collectionView.deselectItem(at: indexPath, animated: true)


        let product = products[indexPath.row]
        let controller = ProductDetailsVC.instance(product: product)
        Router.show(controller: controller)
    }
}
```

Рис. 10. Реализация делегатов отображения UICollectionView

Для корректного отображения списка товаров в виде коллекции контроллер ProductsVC реализует дополнительный протокол – UICollectionViewDelegateFlowLayout. Реализуя данный протокол, контроллер получает возможность управления размером каждой ячейки в коллекции, в которой отображается продукт.

Реализация протокола UICollectionViewDelegateFlowLayout представлена на рисунке 11.

```
extension ProductsVC: UICollectionViewDelegateFlowLayout {


    func collectionView(_ collectionView: UICollectionView, layout
collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath:
IndexPath) -> CGSize {
        let side = (collectionView.frame.width - 30) / 2
        return CGSize(width: side, height: side * 1.5)
    }


}
```

Рис. 11. Реализация протокола UICollectionViewDelegateFlowLayout

Для добавления новых товаров пользователь использует AddProductsVC. Пользователь переключается на него нажав на соответствующую кнопку в основном навигационном контроллере.

Для добавления товара пользователю необходимо нажать на кнопку добавления на представлении в контроллере AddProductsVC. После чего событие нажатия будет обработано контроллером и откроется экран сканирования штрих-кодов, за который отвечает ScannerViewController.

После открытия представления для сканирования штрих-кодов пользователь увидит изображение с камеры, отображенное на весь экран мобильного устройства.

При наведении камеры на штрих-код, система автоматически его распознает и будет произведена его обработка.

Обработка отсканированного штрих-кода представлена на рисунке 12.

Список товаров для сохранения реализован с помощью представления UITableView. Данное представление отображает все элементы для пользователя в виде единого списка.

Обязанности источника данных и обработчика действий пользователя берёт на себя AddProductsVC. Для этого данный контроллер реализует соответствующие протоколы отображения UITableView – UITableViewDelegate и UITableViewDataSource.

```
func metadataOutput(_ output: AVCaptureMetadataOutput, didOutput
metadataObjects: [AVMetadataObject], from connection: AVCaptureConnection) {
        captureSession.stopRunning()

        if let metadataObject = metadataObjects.first {
            guard let readableObject = metadataObject as?
AVMetadataMachineReadableCodeObject else { return }
            guard let stringValue = readableObject.stringValue else { return
}

AudioServicesPlaySystemSound(SystemSoundID(kSystemSoundID_Vibrate))
            found(code: stringValue)
        }

        dismiss(animated: true)
}
```

Рис. 12. Обработка отсканированного штрих-кода

Реализация протоколов отображения UITableView приведена на рисунке 13.

У пользователя имеется возможность просмотра детальной информации о продукте, а также возможность её редактирования. Для реализации данного функционала был разработан ProductDetailsVC.

Название продукта отображается в элементе UITextField. Данное отображение позволяет как показывать пользователю текстовую информацию, так и получать от пользователя текст по средством ввода с клавиатуры.

```
extension AddProductsVC: UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return unsavedProducts.count
    }
    func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
NewProductTableCell.reuseID) as! NewProductTableCell
        let product = unsavedProducts[indexPath.row]
        cell.setup(withProduct: product)
        return cell
    }
    func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
        if editingStyle == .delete {
            unsavedProducts.remove(at: indexPath.row)
            tableView.deleteRows(at: [indexPath], with: .fade)
            setPlaceholderHidden(unsavedProducts.count != 0)
        }
    }
}
extension AddProductsVC: UITableViewDelegate {
    func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        tableView.deselectRow(at: indexPath, animated: true)
        let product = unsavedProducts[indexPath.row]
        Router.show(controller: ProductDetailsVC.instance(product: product))
    }
}
```

Рис. 13. Реализация делегатов отображения UITableView

Для обновления название в модели продукта реализован протокол UITextFieldDelegate. Реализуя данный протокол, контроллер ProductDetailsVC становится делегатом элемента UITextField, получая при этом возможность реагировать на изменение состояния текста и поля для его ввода.

При изменении текста в поле для его ввода, название обновляется и сохраняется у модели продукта.

Реализация протокола UITextFieldDelegate представлена на рисунке 14.

```
extension ProductDetailsVC: UITextFieldDelegate {
    func textFieldDidEndEditing(_ textField: UITextField, reason:
UITextField.DidEndEditingReason) {
        guard let name = textField.text, name.count > 0 else {
            textField.text = product.name
            return
        }
        product.setName(name)
    }
    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return true
    }
}
```

Рис. 14. Реализация протокола UITextFieldDelegate

Редактирование большинства других данных о товаре происходит аналогичным образом, с помощью отображений, соответствующих типу свойства продукта и делегированием обязанностей на контроллер, для получения данных с отображения и изменения модели.

Важнейшей особенностью приложения является поддержка общих пользовательских групп, в которых ведётся общий список товаров и обновляется в режиме реального времени. Это помогает всем членам семьи вести общий, синхронизированный список продуктов.

Реализует данный функционал контроллер GroupVC. Он относительно простой, так как вся логика получения общего списка продуктов находится в сетевом слое.

3.5. Реализация сетевого слоя

В качестве серверного решения был выбран сервис Firebase за свою простоту в использовании и большое количество функционала.

В качестве базы данных был использован инструмент Realtime Database. Данный инструмент представляет собой нереляционную базу данных с поддержкой обновления данных в реальном времени.

Взаимодействие с облачной базой данный осуществляется с помощью REST-запросов по протоколу HTTPS.

Сетевой слой реализован в виде протокола APIRequesting, интерфейс которого представлен на рисунке 15. Данный протокол реализуется структурами, описывающими основные поля запроса, такие как: метод передачи данных протокола HTTPS, путь запроса, а также параметры запроса. Так же протокол APIRequesting имеет дополнительное поле для хранения адреса сервера.

```
protocol APIRequesting: ConnectionChecking {
    associatedtype ResponseModel: Codable

    var httpMethod: HTTPMethod { get }
    var baseURL: String { get }
    var requestURL: String { get }
    var parameters: Any? { get }
}
extension APIRequesting {
    var baseURL: String { "https://appid.firebaseio.com/groups/" }
    var headers: HTTPHeaders { [:] }
}
```

Рис. 15. Интерфейс протокола APIRequesting

Помимо полей, данный протокол требует указания типа возвращаемой модели ResponseModel, которая инициализируется из декодированного ответа сервера. Данный функционал реализован с помощью ассоциативных типов.

Реализуя протокол APIRequesting модель запроса указывает все необходимые поля, описанные в протоколе. Пример реализации запроса с помощью протокола APIRequsting можно увидеть на рисунке 16.

```
extension API {
    enum Products {
        struct All: APIRequesting {
            var httpMethod: HTTPMethod { .get }
            var requestURL: String {
"\(GroupManager.shared.groupID)/products/.json" }
            var parameters: Any? { nil }
            typealias ResponseModel = [Product]

        }
    }
}
```

Рис. 16. Пример реализации протокола APIRequesting

Для выполнения у объекта структуры, реализующей протокол APIRequesting, необходимо вызвать метод execute, выполняющий запрос по данным из описанной структуры и передать в данный метод замыкающую функцию, для получения ответа.

3.6. Реализация пользовательского интерфейса

Для реализации пользовательского интерфейса в мобильном приложении был использован инструмент Interface Builder, предоставляющий множество инструментов для визуальной разработки интерфейса приложения.

Одним из главных преимуществ использования Interface Builder является система Auto Layout, просчитывающая расположение визуальных элементов на экране пользователя во времени выполнения приложения на основе ограничений (Constraints).

Демонстрация процесса создания пользовательского интерфейса с помощью Interface Builder представлена на рисунке 17.

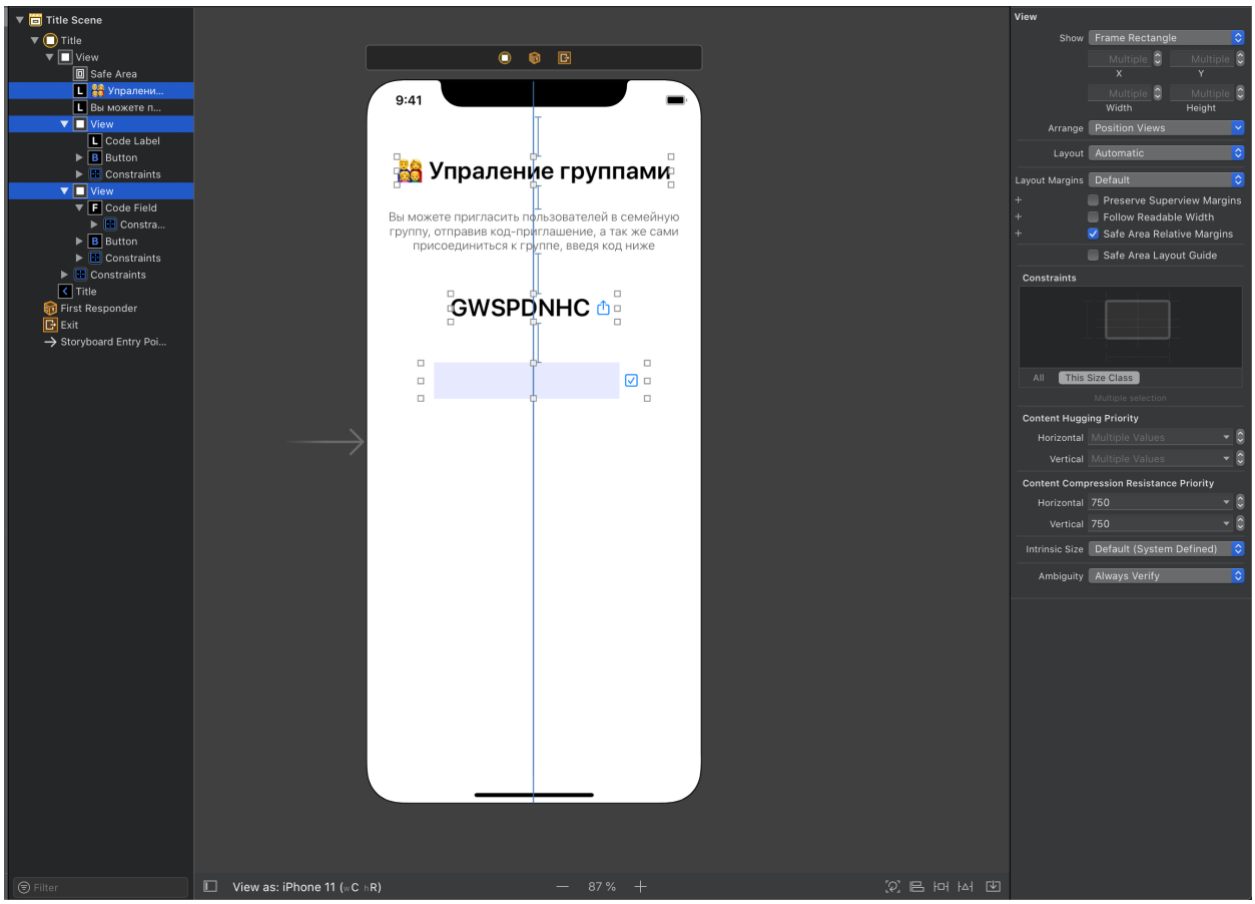

Рис. 17. Создание интерфейса при помощи Interface Builder

Interface Builder применяется внутри файлов Storyboard, предназначенных для создания пользовательского пути, состоящего из нескольких ViewController. Взаимодействие между контроллерами осуществляется с помощью определённого типа перехода между экранами приложения – segeues. Помимо файлов Storyboard, Interface Builder применяется в файлах XIB. Данный тип файлов предназначен для создания отдельных, самостоятельных представлений и последующего их использования и переиспользования.

Реализовав пользовательский интерфейс и имплементировав бизнес логику приложения, получаем полноценный продукт, готовый для тестирования и использования.

Главный экран отображён на рисунке 18. Он представлен в двух состояниях: обучение и список продуктов. Если пользовательский список

продуктов пуст, ему отображается информационное окно с предложением добавить новые продукты.

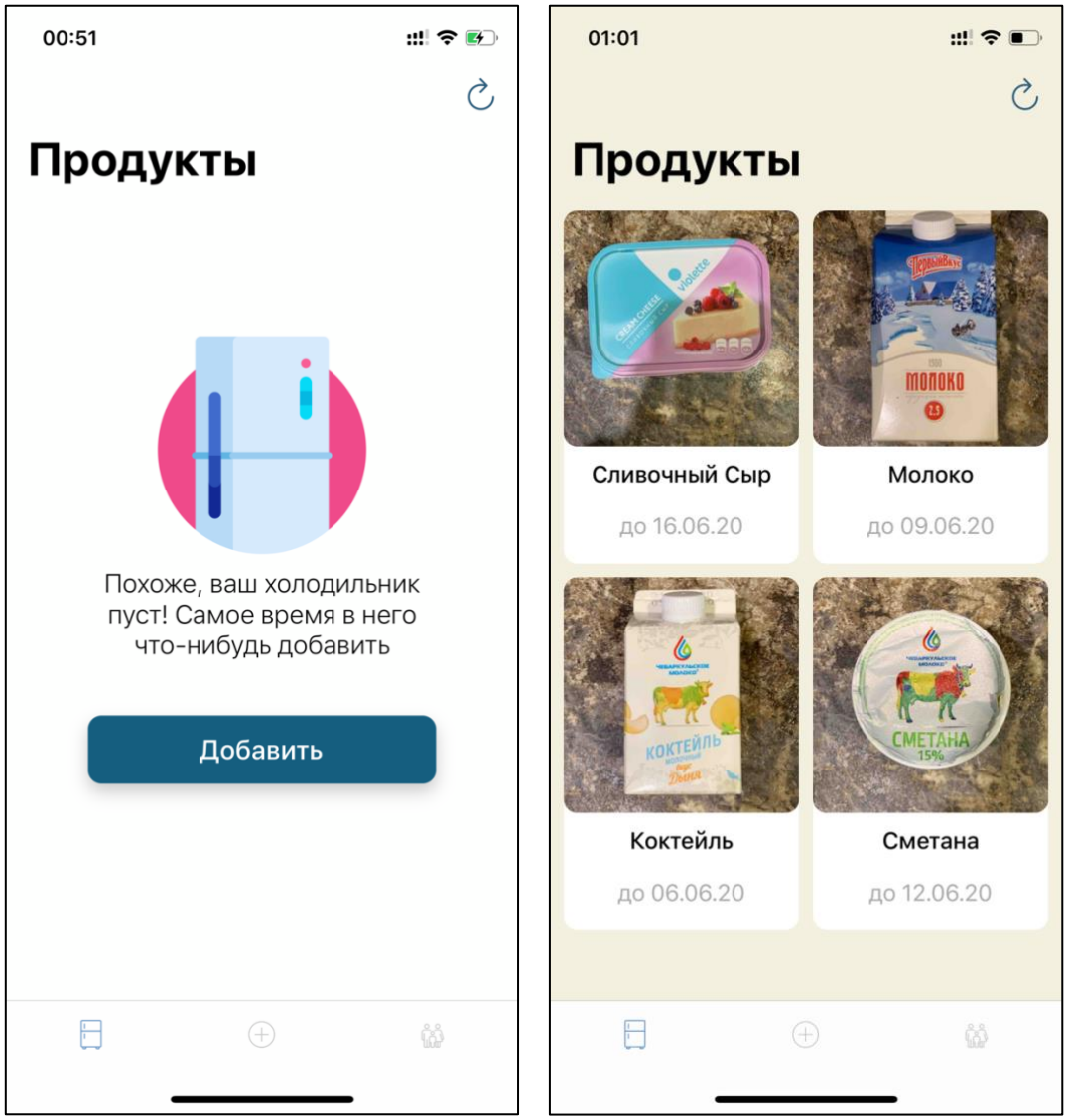

Рис. 18. Главный экран приложения

Для добавления новый продуктов пользователю необходимо перейти во вторую вкладку приложения, с помощью нижнего навигационного меню. На данном экране пользователь добавляет новый продукты к себе в холодильник, сканируя штрих-коды продуктов. На рисунке 19 отображён экран добавления продуктов в двух состояниях: список пустой и в списке присутствуют товары для сохранения, а также отображён экран сканирования штрих-кода. После заполнения списка и нажатия кнопки сохранения, продукты добавятся в основной список продуктов, пример которого отображён на рисунке 18.

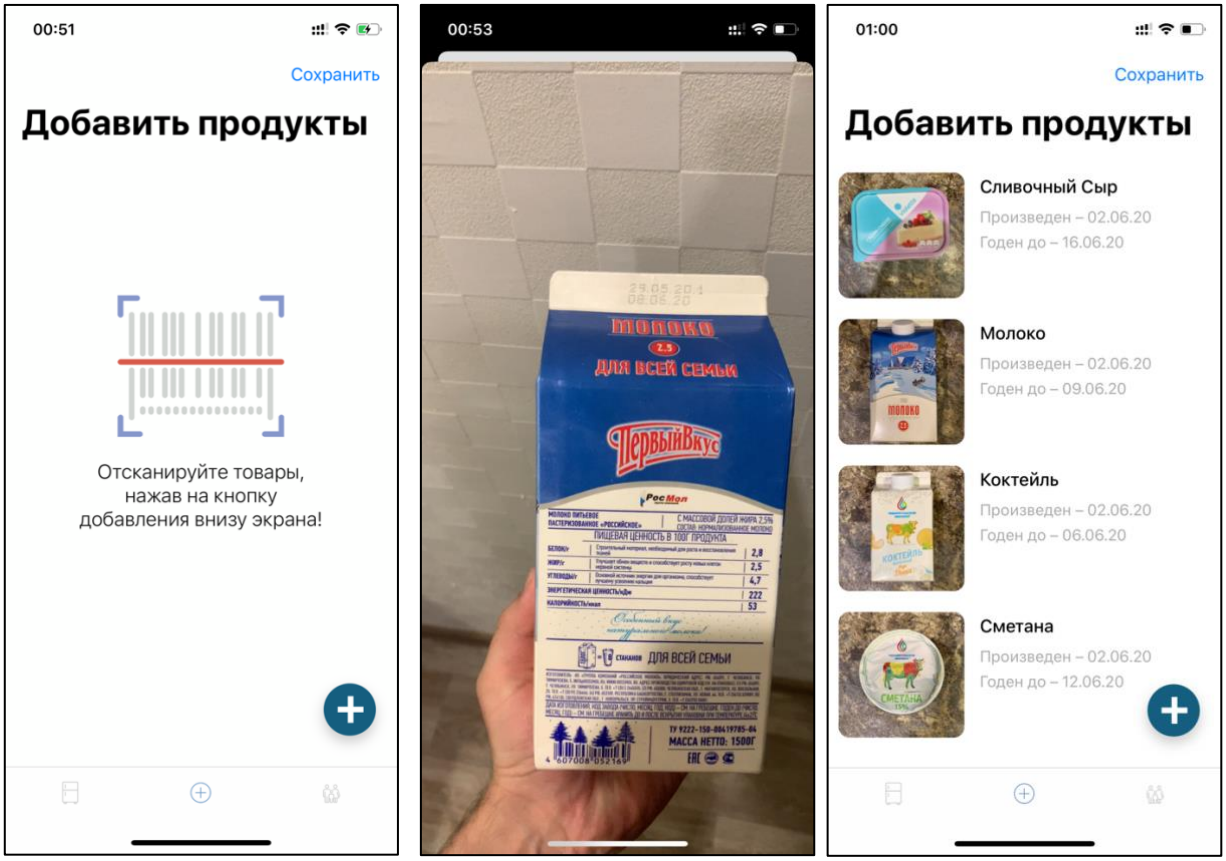

Рис. 19. Процесс добавления новых продуктов

Просмотр и редактирование информации и продукте доступно на экране детальной информации о продукте. На данном экране пользователь может отредактировать всю информацию о продукте, просмотреть штрих-код, а также сохранить продукт как шаблон, для автоматического заполнения информации при сканировании продуктов с совпадающим штрих-кодом. На рисунке 20 отображён экран просмотра информации о продукте, а также меню действий, которые можно совершить над продуктом.

Несколько устройств можно объединить в одну общую группу, синхронизировав при этом список продуктов. Эта функция удобна для нескольких членов одной семьи, пользователи могут с разных устройств управлять общим списком продуктов. Помимо информации о продуктах, синхронизируются ещё и фотографии, так как находятся на общем файловом хранилище и сохраняются на нём при добавлении фотографии к продукту.

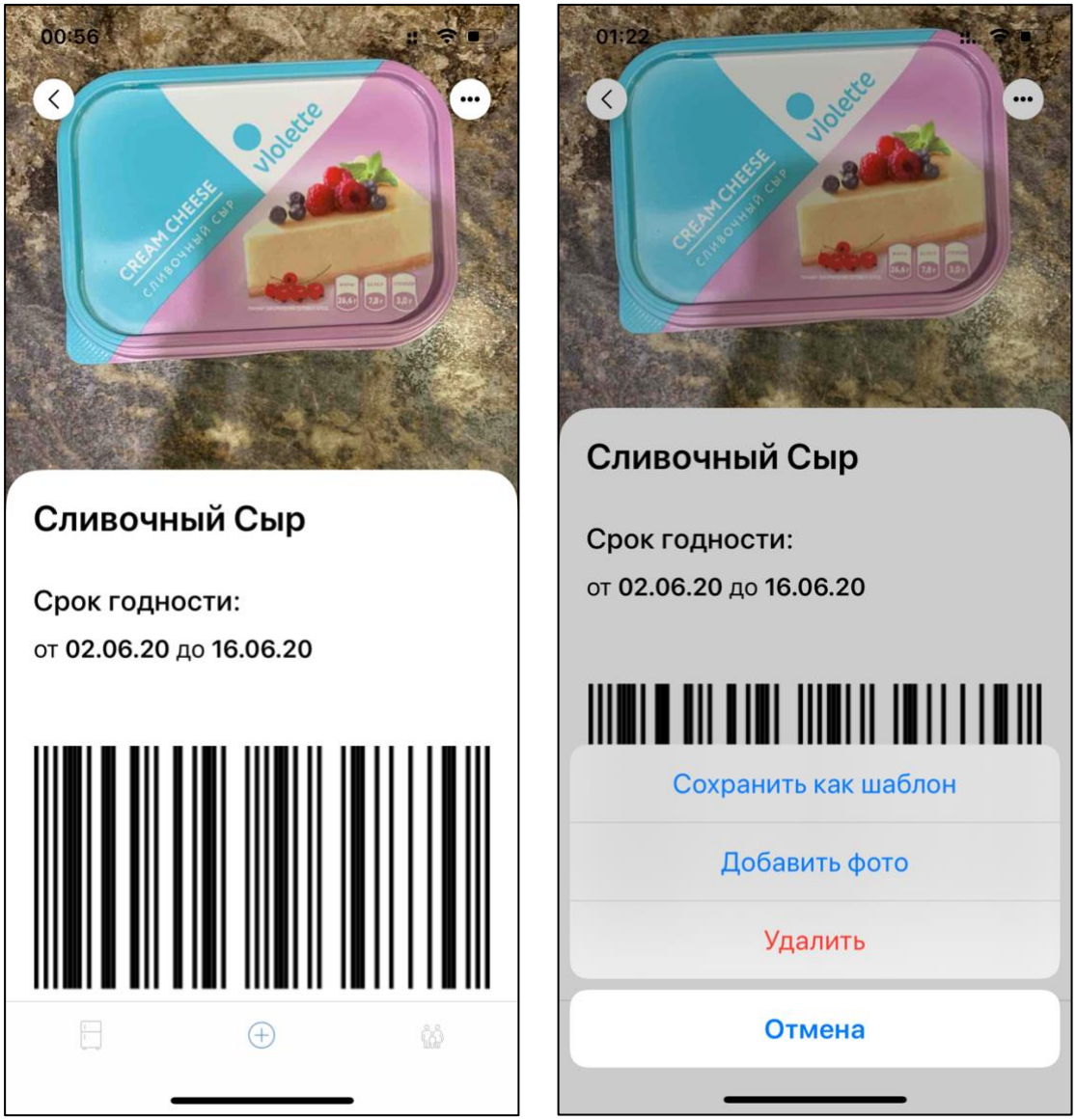

Рис. 20. Просмотр детальной информации о продукте

Для синхронизации нескольких устройств, необходимо на одном из них ввести код-приглашение, отправленный с другого устройства. Данный код можно как скопировать, нажав на сам код, так и поделиться с помощью системного меню, нажав на соответствующую кнопку рядом с кодом, что отображено на рисунке 21.

На рисунке 22 мы видим, что после ввода кода на экране управления отображается введённый нами код-приглашение, а список продуктов синхронизировался.

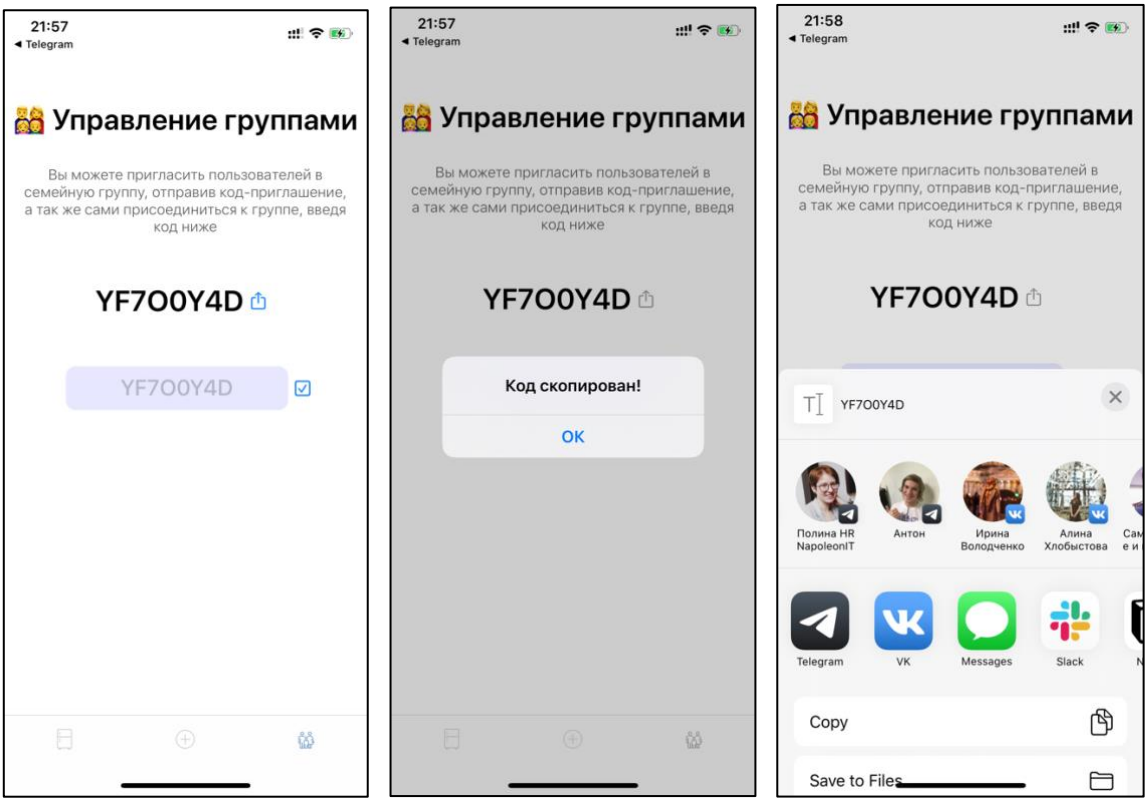

Рис. 21. Управление семейными группами

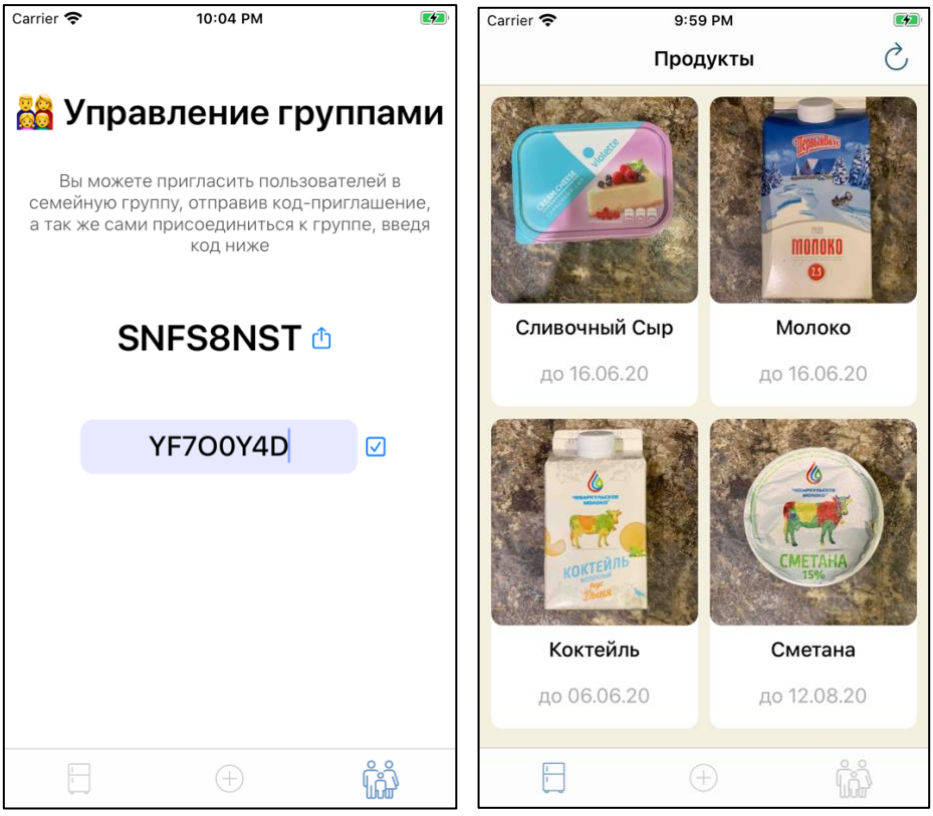

Рис. 22. Ввод кода-приглашения и синхронизация списка продуктов

4. ТЕСТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

4.1. Функциональное тестирование мобильного приложения

Для функционального тестирования мобильного приложения были выбраны все функции, определенные в функциональных требованиях при проектировании приложения. Результаты тестирования представлены в табл. 1.

Табл. 1. Протокол тестирования мобильного приложения

№	Функция	Шаги	Ожидаемый результат	Итог
1	Добавить продукт, отсканировав штрих-код	1. Открыть экран добавления 2. Нажать кнопку добавления 3. Отсканировать штрих-код	Продукт добавится в список	Функция работает
2	Отредактировать срок годности продукта	1. Открыть карточку продукта 2. Нажать на редактирование срока годности 3. Выбрать новую дату	У продукта изменится срок годности на выбранный пользователем и установится уведомление	Функция работает
3	Добавить изображение для продукта	1. Открыть карточку продукта 2. Нажать на кнопку меню 3. Выбрать добавить изображение 4. Сфотографировать продукт	У продукта обновится изображение и загрузится на сервер	Функция работает
4	Уведомление об истечении срока годности	Пользователю не требуется дополнительных шагов, а только добавить какой-либо товар в приложение	За день до истечения срока годности пользователь получит уведомление	Функция работает
5	Создание шаблонов продуктов	1. Открыть карточку продукта 2. Нажать на кнопку меню 3. Выбрать сохранение шаблона	Шаблон для продукта создастся или обновится и будет доступен для использования	Функция работает
6	Создание семейных групп	1. Открыть экран управления группой 2. Поделиться кодом группы или ввести код группы и применить его	Список продуктов двух или более пользователь будет синхронизирован	Функция работает

Окончание таблицы 1.

№	Функция	Шаги	Ожидаемый результат	Итог
7	Просмотр актуального списка продуктов	1. Открыть экран просмотра списка продуктов	Пользователю будет отображён список продуктов, которые он добавил раннее	Функция работает
8	Просмотр подробной информации о продукте	1. Открыть экран списка продуктов или добавления продуктов 2. Нажать на продукт в списке. Если продукты отсутствуют, то прежде их необходимо добавить	Откроется карточка продукта с подробной информацией о продукте	Функция работает
9	Удаление продукта	1. Открыть экран списка продуктов 2. Открыть карточку продукта 3. Нажать на меню 4. Из списка выбрать вариант удаления	Продукт удалится, у пользователя закроется карточка продукта и в списке продуктов будет отсутствовать данный продукт	Функция работает

Помимо функционального тестирования были реализованы модульные тесты приложения с помощью XCTest [15], при сборке автоматически проверяющие логику работы приложения.

Вывод по главе 4

В результате проведенного тестирования было установлено соответствие разработанного приложения исходным функциональным требованиям заказчика. В ходе тестирования не было выявлено ошибок в функционале приложения.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано мобильное приложение для учёта продуктов на платформе iOS. В ходе разработки были изучены и применены актуальные способы разработки, а также архитектурные подходы к проектированию приложения. Объём конечного приложения составил 2700 строк.

Для достижения этой цели был проведён анализ предметной области. После чего было спроектировано и разработано мобильное приложение, с последующим тестированием его на соответствие всем функциональным требованиям.

Разработанное приложение было сдано заказчику в опытную эксплуатацию.

ЛИТЕРАТУРА

1. LuckyRocky – разработчик приложений в AppStore. [Электронный ресурс] URL: https://itunes.apple.com/ru/developer/luckyrocky/id565691353 (дата обращения: 20.05.2020).

2. OptiLifeApps – разработчик приложений в AppStore. [Электронный ресурс] URL: https://itunes.apple.com/ru/developer/optilife-apps/id1364927452?mt=8 (дата обращения: 20.05.2020).

3. XCode 11. [Электронный ресурс] URL: https://developer.apple.com/xcode/ (дата обращения: 20.05.2020).

4. Swift. [Электронный ресурс] URL: https://www.apple.com/ru/swift/ (дата обращения: 20.05.2020).

5. Interface Builder. [Электронный ресурс] URL: https://developer.apple.com/xcode/interface-builder/ (дата обращения: 20.05.2020).

6. Cocoa Touch. [Электронный ресурс] URL: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html (дата обращения: 20.05.2020).

7. Firebase – документация. [Электронный ресурс] URL: https://firebase.google.com/docs (дата обращения: 20.05.2020).

8. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования. — 3-е изд. — М.: Вильямс, 2006. — 736 с.

9. Сканнеры штрих–кодов [Электронный ресурс] URL: https://scancode.ru/catalog/skanery–shtrikh–koda (дата обращения: 25.04.2020)

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

11. UIKit. [Электронный ресурс] URL: https://developer.apple.com/documentation/uikit (дата обращения: 16.05.2020).

12. Foundation Kit Framework. [Электронный ресурс] URL: https://developer.apple.com/documentation/foundation (дата обращения: 18.05.2020).

13. Усов В.А. Swift. Основы разработки приложений под iOS и macOS. – 4-е изд., перераб. и доп. – СПб.: Питер, 2018. – 448 с.

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

15. XCTest. [Электронный ресурс] URL: https://developer.apple.com/reference/xctest (дата обращения: 20.05.2020).