МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ П.М. МАШЕРОВА»

Факультет математики и информационных технологий

Кафедра прикладного и системного программирования

Допущена к защите

«\_\_»\_\_\_\_\_\_\_20\_\_г.

Заведующий кафедрой

\_\_\_\_\_\_\_\_С.А.Ермоченко

ДИПЛОМНЫЙ ПРОЕКТ

РАЗРАБОТКА PWA ПРИЛОЖЕНИЯ ПО УЧЕТУ ЗАКАЗОВ СЕТИ ЦВЕТОЧНЫХ МАГАЗИНОВ

Специальность: 1-40 01 01 Программное обеспечение информационных технологий

Хлопяников Дмитрий Эдуардович,

4 курс, 44 группа

Научный руководитель:

Никитин Александр Игоревич,

Старший преподаватель

Витебск, 2020

**РЕФЕРАТ**

Дипломный проект представлен следующим образом. Электронные носители: 1 компакт-диск. Пояснительная записка составляет 86 страниц, 32 рисунка, 3 таблицы, 17 литературных источников, 3 приложения.

Ключевые слова: Spring, Spring Boot, Hibernate, ReactJS, Redux, Saga, JavaScript, Java, терминал, админ-панель.

Предметная область: автоматизация и учет цветочного магазина. Объект разработки: клиент-серверное приложение для автоматизации и учета цветочного магазина.

Целью разработки является клиент-серверное приложение, которое предоставит возможность автоматизировать процесс администрирования цветочного магазина.

При разработке клиент-серверного приложения были использованы следующие среды разработки и сторонние программы: IntelliJ IDEA, Visual Studio Code, Chrome, Postman. Языки программирования, которые были использованы при разработке: Java, JavaScript. База данных – PostgreSQL.

В результате технико-экономического обоснования было выяснено, что инвестирование в разработку является выгодным вложением, а само приложение является прибыльным, что говорит об экономической обоснованности данной разработки.

Разработанный продукт может использоваться работниками цветочного магазина для упрощения формирования заказов и их контроля, а также владельцами магазинов для осуществления администрирования, отслеживания статистики продаж, установки ассортимента, ведения складского учета и управления сотрудниками.

Задачи, поставленные перед реализацией дипломного проекта, были выполнены. В перспективе возможно расширение функционала новыми возможностями.

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 6

1 ОБЗОР ЛИТЕРАТУРЫ 8

1.1 Обзор аналогов 8

1.1.1 ABM Retail 8

1.1.2 CloudShop 10

1.1.3 FloraPOS 12

1.2 Обзор средств разработки 13

1.2.1 Языки программирования 14

1.2.2 Философия REST 14

1.2.3 Обзор инструментов разработки серверной части приложения 15

1.2.4 Обзор инструментов разработки клиентской части приложения 17

1.2.5 Обзор систем управления базами данных 21

1.3 Постановка задачи 22

1.4 Обоснование выборки средств разработки 22

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 24

2.1 Модуль работы backend 24

2.2 Модуль работы frontend 24

2.3 Модуль работы с данными 25

2.4 Модуль терминала 25

2.5 Модуль админ-панели 26

2.6 Модуль авторизации для терминала 26

2.7 Модуль авторизации для админ-панели 27

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 28

3.1 Модуль работы backend 28

3.2 Модуль работы с данными 40

3.3 Модуль реляционной базы данных 44

3.4 Модуль работы frontend 49

3.5 Модуль терминала 50

3.6 Модуль админ-панели 53

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 56

4.1 Аутентификация и авторизация для админ–панели с применением JWT-токенов 56

4.2 Аутентификация и авторизация для терминала с применением JWT-токенов 59

4.3 Алгоритм работы флориста в терминале 62

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 66

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 71

6.1 Руководство пользователя для терминала 71

6.2 Руководство пользователя для админ-панели 76

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА ПО РАЗРАБОТКЕ СИСТЕМЫ АВТОМАТИЗАЦИИ И УЧЕТА ДЛЯ ЦВЕТОЧНЫХ МАГАЗИНОВ 80

7.1 Характеристика программного средства 80

7.2 Расчет затрат на разработку ПО 80

7.2.1 Затраты на основную заработную плату команды разработчиков 80

7.2.2 Затраты на дополнительную заработную плату команды разработчиков 81

7.2.3 Отчисления в фонд социальной защиты населения и фонд обязательного страхования 82

7.2.4 Прочие затраты 82

7.3 Оценка экономического эффекта от продажи ПО 83

7.3.1 Экономический эффект от продажи ПО для организации разработчика 83

7.4 Расчет показателей эффективности инвестиций в разработку ПО 84

ЗАКЛЮЧЕНИЕ 86

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 87

ПРИЛОЖЕНИЕ А 89

ПРИЛОЖЕНИЕ Б 108

ПРИЛОЖЕНИЕ В 109

# ВВЕДЕНИЕ

В сфере современного бизнеса важнейшую роль играет учет. Подсчет производственных затрат, зарплат сотрудникам, складских накоплений и многого другого, имеет большую значимость. Статистика – чрезвычайно важный источник информации, который невозможно недооценить. Не обладая достоверными статистическими данными, успех и развитие бизнеса практически невозможны.

Важность и актуальность учета стали причиной создания большого количества предложений по решению проблемы сбора статистики. Принципы таких решений основываются на объеме рынка. Таким образом, решить проблему автоматизации учета для сети ресторанов, магазинов и отелей не составит труда. Но, в то же время, сложно найти специализированные решения для учета узконаправленных задач, что и стало причиной для дальнейшего исследования.

Для владельца малого бизнеса, занимающегося реализацией какого-либо продукта, рано или поздно поднимается вопрос учета. С развитием производства этот вопрос приобретает другой характер – необходимость поиска средства для автоматизации данного учета. Однако, если сфера, которой занимается владелец, является специфической, поиск специализированного решения может стать большой проблемой. В этом случае, для владельца есть несколько способов решения проблемы. Первый - приобрести программное обеспечение, предназначенное для учета в других сферах, и подстроить его под себя. Минусом данного способа является переплата за избыточный функционал, который бесполезен для специализированных нужд, а также возможные неудобства, которые могут возникнуть из-за использования некоторых функций не по назначению, предусмотренному при их разработке. Вторым способом является заказ программного обеспечения под собственные нужды и специфику производства, что и определило актуальность темы.

Цветочный бизнес достаточно широко распространен в сфере малого и среднего бизнеса, и меньше – в крупном бизнесе. Из-за этого количество предложений по автоматизации в данной сфере достаточно мало, многие из которых обладают существенными недостатками.

Таким образом, целью дипломного проекта является разработка системы автоматизации и учета цветочных магазинов, специализированных под нужды реального владельца.

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Обзор аналогов

Разрабатываемый программный продукт не представляет собой уникальную разработку. Существует множество решений для автоматизации учета, которые в некоторой степени можно использовать как систему учета для цветочных магазинов, но это будет иметь под собой очевидные недостатки. Найти универсальный продукт, который смог бы решить все необходимые задачи, очень сложно. Максимально нецелесообразно переплачивать за продукт, в котором есть избыточный функционал, часть из которого останется непригодным. Поэтому стоит задача разработать продукт, соответствующий всем требованиям и реализующий все необходимые задачи.

В качестве аналогов были рассмотрены следующие программные продукты:

- ABM Retail;

- CloudShop;

- FloraPOS.

### 1.1.1 ABM Retail

Владельцем ABM Retail (см. рисунок 1.1) является компания ABMcloud [1]. ABM Retail – это готовое решение для автоматизации магазина, сети магазинов и аптек.

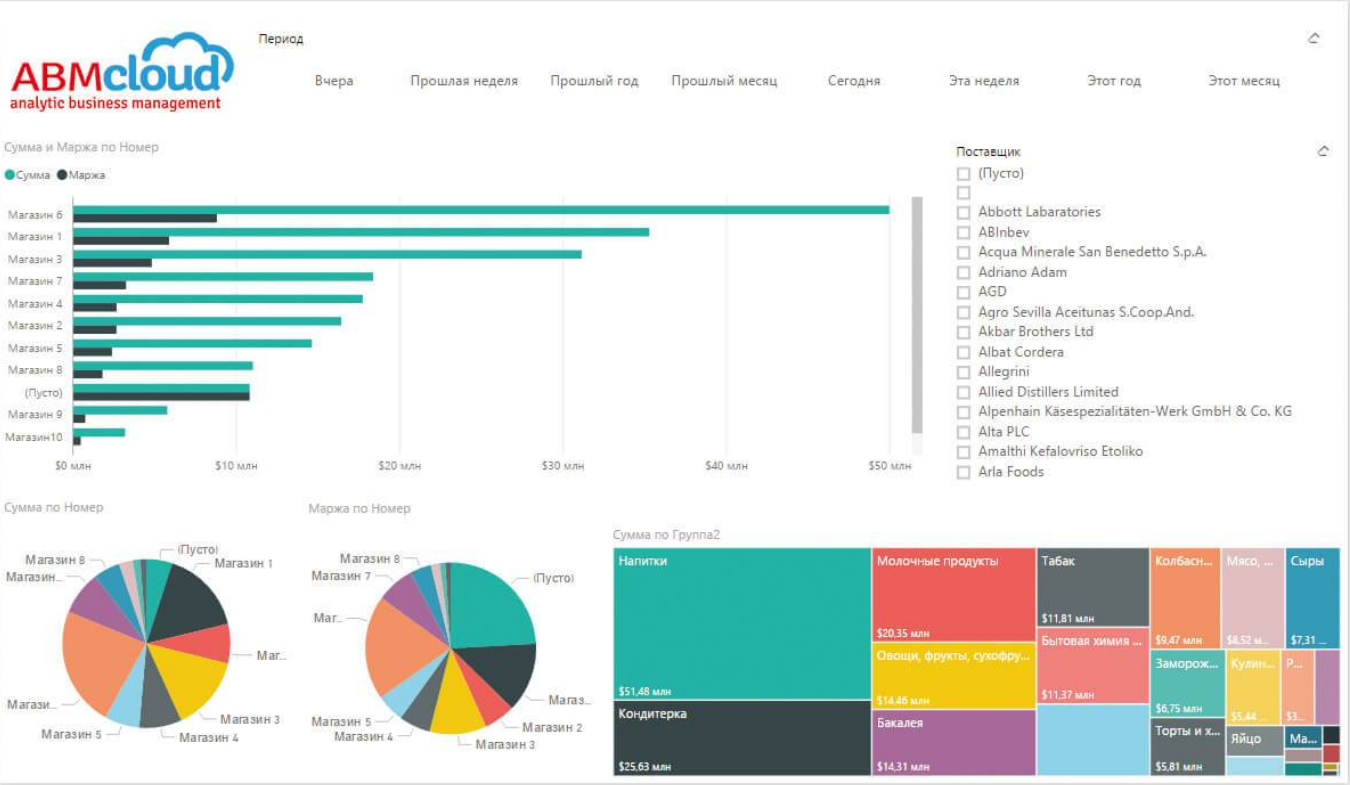


Рисунок 1.1 – ABM Retail

Возможности, реализованные в ABM Retail:

* назначение категорийных менеджеров, управление ассортиментом;
* создание и анализ ассортиментных матриц;
* контроль плановой и фактической глубины ассортимента;
* задание неограниченного количества ценовых диапазонов и связывание их с категориями товаров;
* получение данных о закупках, продажах, остатках товаров на складах, оборачиваемости товаров;
* настраивание правил расчета розничных цен и установка цены реализации товаров;
* ведение спецификации поставщиков и контроль цен поставки;
* настраивание под себя и печать ценников и этикеток;
* планирование расчета заказа на производство, подбор материалов;
* прозрачный учет использования сырья в производстве, ведение учета разделки, приготовления, дефростации и разборки готовой продукции;
* получение информации о продажах товаров, себестоимости реализованной номенклатуры, показателях прибыли и реализованной наценки в реальном времени;
* проведение оприходования, списания, инвентаризации, выполнение пересортицы товаров;
* использования терминала сбора данных в офлайн и онлайн режиме для отражения операций приемки;
* ведение кадрового учета сотрудников с использованием Кадрового приказа, выполнение внутренней ротации и увольнение сотрудников;
* возможность использования отчетов продаж по группам товара, отчетов о покупателях, ABC-анализа, статистики поставщиков;
* выгрузка первичных документов 1С: Бухгалтерия, обмен документами с поставщиками через EDI.

Помимо ABM Retail, ABMcloud также имеет специализированные решения ABM Inventory, ABM Shelf, ABM Loyalty. Все эти продукты могут успешно работать в связке с представленным аналогом. Они подходят для крупных розничных сетей, раскрывая дополнительные нюансы автоматизации учета.

Широкий спектр возможностей и гибкость данного программного продукта позволяет использовать его для автоматизации магазинов различных специализаций. Компания ABMcloud имеет опыт работы в 15 лет и сотни внедренных проектов, преимущественная часть которых представляет собой сеть продуктовых магазинов. Использование ABM Retail для одного магазина или небольшой сети делает данный продукт менее эффективным в соотношении цена-качество. В нашем случае, для цветочного магазина, имеющего свою специфику в списании товара и формировании цены, некоторый функционал, представленный в ABM Retail теряет актуальность и остается ненужным, а некоторый функционал не представлен вообще, что повлечет за собой неудобства в процессе дальнейшего масштабирования бизнеса.

ABM Retail представляет собой десктопное приложение, что накладывает ограничения на возможность использования, в сравнении с аналогами, реализованными в виде веб-клиента.

Компания ABMcloud не предоставляет продукт для общего доступа. Разговор с потенциальными клиентами происходит индивидуально, определяются цели и задачи, преследуемые клиентом.

Приложение является кроссплатформенным, то есть не зависит от платформы, на которой это приложение будет разворачиваться. Также, возможен доступ к мобильной версии приложения, если клиент преследует данную цель.

На старте сотрудничества, компания ABMcloud рассчитывает точный бюджет внедрения, который не изменяется. Далее система предоставляется в аренду, как сервис, то есть не переходит во владение покупателя.

Таким образом, можно сказать, что программный продукт ABM Retail от компании ABMcloud является универсальным продуктом на рынке, который можно использовать в любой сфере розничной торговли. Однако, для автоматизации цветочного магазина, обладающего своей спецификой, а также беря в расчет другие аспекты, раскрытые выше, ABM Retail не является наилучшим решением.

### 1.1.2 CloudShop

CloudShop (рисунок 1.2) – программа для контроля торговли и автоматизации розничного магазина, принадлежащее обществу с ограниченной ответственностью «КлаудШоп». Сервис содержит основные операции – закупка, продажа и возврат, загрузка номенклатуры из таблиц, импорт и экспорт поставщиков и покупателей. В программе учета товаров можно оформлять приходы и расходы денег, следить за балансом в кассе, настраивать скидки, вести статистику и аналитику продаж. К сервису можно подключить онлайн-кассу, сканер штрихкодов и интернет-магазин.

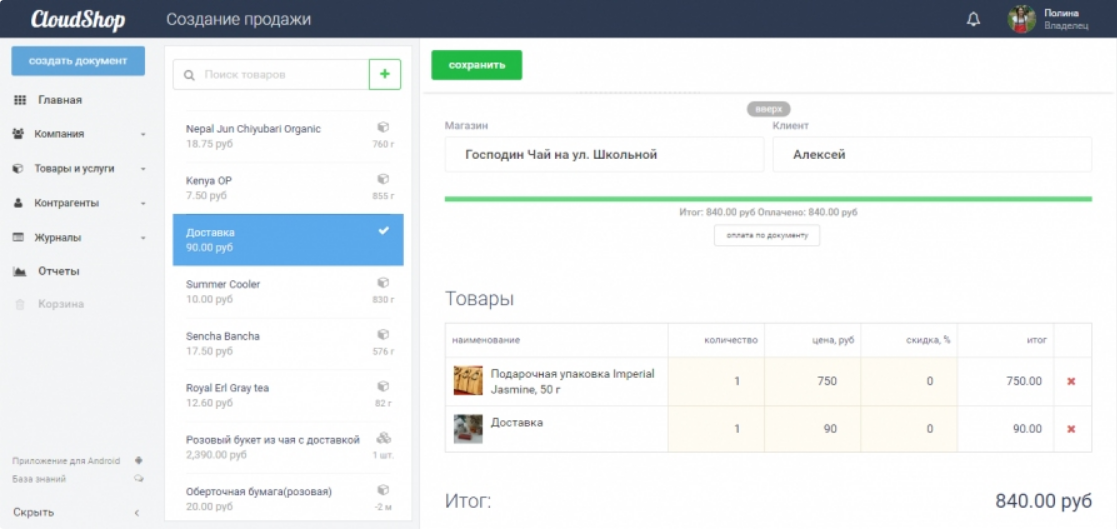


Рисунок 1.2 – CloudShop [2]

В отличие от первого аналога, данная программа имеет меньший спектр возможностей и не подойдет для большого бизнеса. В это же время CloudShop обладает всем необходимым функционалом для малого бизнеса.

CloudShop имеет установленный список тарифных планов со своими ценами. Всего четыре тарифных плана: базовый, простой, стандарт и безлимит. От тарифного плана в первую очередь зависит количество поддерживаемых магазинов и сотрудников. Также от тарифного плана зависит возможность получения отчетов, скорость ответа от технической поддержки, доступ к онлайн-кассе 54-ФЗ. При оплате на несколько месяцев сразу, сервисом предусмотрены скидки.

Программу можно использовать бесплатно, так как у нее есть бесплатный тариф, на котором можно вести учет одного магазина, в котором сможет работать один сотрудник. Поэтому CloudShop может работать, как любая программа для небольшого магазина или одной торговой точки, при этом ее не нужно скачивать, устанавливать и обновлять – программа представляет собой веб-клиент. CloudShop также имеет мобильное приложение, которое предоставляет полный доступ к данным. Мобильное приложение представлено как в App Store, так и в Google Play. CloudShop совместим со следующими операционными системами: Windows 10, Windows 7, Windows 8, Linux, MacOS, Android 4, Android 5, Android 6, Android 7, iOS.

Программа имеет пробный период, который длится 14 календарных дней. В это время доступны все функции CloudShop без ограничений. Информация хранится на серверах CloudShop и попадает туда в зашифрованном виде, пользователь не зависит от компьютера. Также, программа обеспечивает резервное копирование в режиме реального времени.

Программа для учета и автоматизации торговли CloudShop универсальна. Сервис может быть программой для магазина одежды, автозапчастей, косметики или продуктового магазина.

Однако, некоторые специфические особенности цветочного магазина будет проблематично решить функционалом CloudShop. Несмотря на это, для малого бизнеса CloudShop выглядит очень симпатичным вариантом при выборе сервиса для автоматизации магазина, в большей степени из-за доступной цены.

### 1.1.3 FloraPOS

FloraPOS (рисунок 1.3) – программа для автоматизации флористического салона или сети салонов, владельцем которой является группа компаний «Асгард», основанная в 2004 году.

FloraPOS представляет собой десктопное приложение, что, также как и в случае первого аналога, накладывает ограничения на возможность использования.

В отличие от предыдущих аналогов, данный представляет собой специализированную программу для цветочных магазинов.

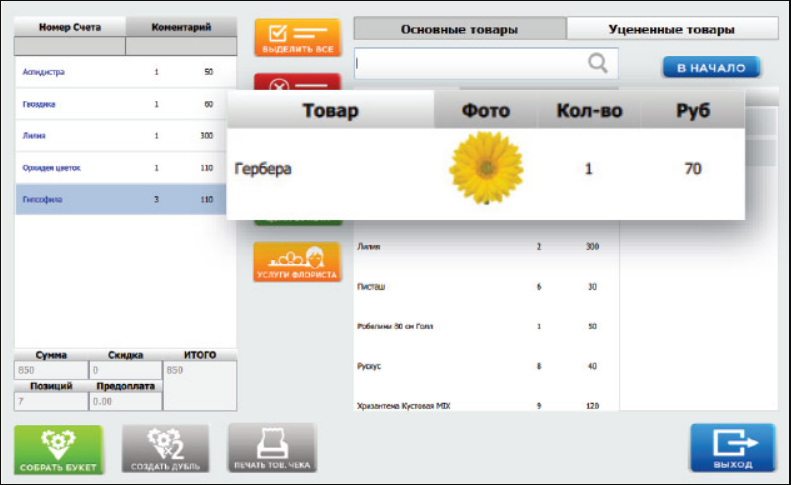


Рисунок 1.3 – FloraPOS [3]

Преимуществом FloraPOS является ее адаптация к ежедневным процессам флористического бизнеса, а именно:

* продажа и возврат товаров;
* составление букетов;
* уценка товара (в том числе и неоднократная);
* списание товара и частичное списание букетов при его переборке;
* инвентаризация (полная и частичная);
* предзаказ и ведение клиентской базы;
* гибкая система лояльности;
* интеграция с интернет-магазинами и CRM;
* интеграция с курьерской службой и службой логистики.

Также, FloraPOS является кроссплатформенным приложение, для его использования достаточно любого компьютера, планшета или смартфона. При обмене данными между сервером, магазинами и офисом происходит шифрование данных, что не дает возможность третьим лицам использовать конфиденциальную информацию, составляющую коммерческую и служебную тайну.

Программа разделена на 4 модуля: «Администратор», «Кассир», «Склад», «Курьер».

Модуль «Кассир» позволяет производить продажи, применять скидки, вносить изъятия из кассы и собирать букеты.

Модуль «Склад» предоставляет информацию о поступлениях, списаниях, уценках и инвентаризации товара.

Модуль «Администратор» позволяет вести клиентскую базу, иметь бонусную систему, настраивать маркетинг и получать аналитические отчеты.

Модуль «Курьер» представляет собой специальное мобильное приложение. Отличием от аналогов является наличие модуля курьера, что делает информирование курьера о заказе очень простым. Для этого необходимо в предзаказе выбрать курьера из списка и ему на смартфон поступит уведомление о том, откуда забрать заказ и куда его привезти.

Есть два способа получения доступа к пользованию приложением: аренда и покупка. В случае аренды покупателем производится посуточная оплата в размере 50 российских рублей. В случае покупки цена формируется в зависимости от необходимого набора модулей. При приобретении модулей «Кассир» и «Склад» необходимо оплатить 17 тысяч российских рублей. При добавлении к ним модуля «Администратор» цена вырастает до 25 тысяч. Цена всех модулей – 29 тысяч 500 рублей.

Помимо приложения, от ГК «Асгард» предоставляется возможность заказа типовых комплектов оборудования для быстрого запуска магазина, что является очень привлекательным вариантом для создания бизнеса с нуля.

Можно сделать вывод, что из всех представленных аналогов, FloraPOS наиболее приближен по функционалу для поставленных проектом целей. Однако, так как отсутствует задача запуска бизнеса с нуля, а также невыполнение приоритетной задачи о наличии веб-клиента делает FloraPOS не самым лучшим решением.

## 1.2 Обзор средств разработки

Для разработки клиент-серверного приложения используются языки программирования и фреймворки, которые облегчают разработку. В процессе разработки для отдельных модулей проекта может использоваться несколько языков программирования.

### 1.2.1 Языки программирования

*Java* – язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования. Является языком с сильной типизацией. Написанное на Java приложение можно запустить на любой платформе, если на ней установлена среда для исполнения Java (JRE, Java Runtime Environment) [4].

Эта задача решается благодаря компиляции написанного на Java кода в байт-код. Этот формат исполняет JVM или виртуальная машина Java. JVM — часть среды исполнения Java (JRE). Виртуальная машина не зависит от платформы.

Java входит в тройку самых популярных языков программирования и по данным компании Oracle, программы на Java запускаются на 3 миллиардах девайсов.

*JavaScript* – это специальный язык программирования, разработанный для использования в браузерах, являющийся на сегодняшний момент безальтернативным [5].

Стоить отметить, что активно начал набирать популярность WebAssembly, однако на текущий момент он является лишь дополнением JavaScript, хотя и претендует на становление независимым языком работающем в браузере.

JavaScript – мультипарадигменный язык программирования, а значит поддерживает 3 основные парадигмы программирования: функциональная, императивная и объектно-ориентированная. Поддержка разных парадигм программирования позволяет придерживаться основных паттернов разработки как клиентских, так и серверных приложений.

### 1.2.2 Философия REST

*REST* (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful системами [6].  
В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протоколом является HTTP. Для HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия CRUD (create, read, update, delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

### 1.2.3 Обзор инструментов разработки серверной части приложения

*Spring Framework*, или просто *Spring* — один из самых популярных фреймворков для создания веб-приложений на Java [7].

Но важно понять, что спринг — это просто набор каких-то классов и интерфейсов, которые уже написаны за вас.

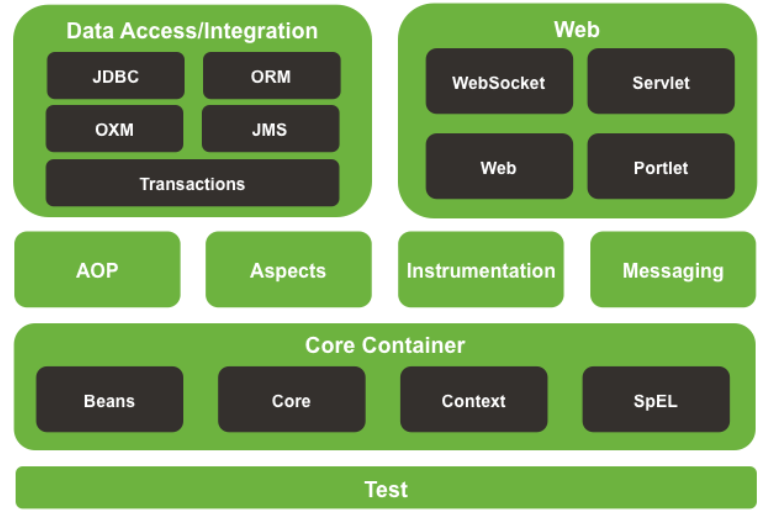


Рисунок 1.4 – Разделения Spring

Как видно, у спринга модульная структура. Это позволяет подключать только те модули, что нам нужны для нашего приложения и не подключать те, которыми мы заведомо не будем пользоваться. Именно этот подход и помог спрингу обойти своего конкурента (EJB) и захватить лидерство. Потому что приложения, использующие EJB, тянули очень много зависимостей за собой, да и вообще получались медленные и неповоротливые.

К примеру, добавляя зависимость spring-boot-starter-web, она подтянет в проект все библиотеки, необходимые для разработки Spring MVC-приложений, такие как validation-api, jackson-json, spring-webmvc, Tomcat.

*Spring Boot* позволяет легко создавать полноценные, производственного класса Spring-приложения, про которые можно сказать - "просто запусти" [8]. Большинству Spring Boot приложениям требуется совсем маленькая Spring-конфигурация. Основными возможностями Spring Boot являются:

* создание полноценных Spring приложений;
* встроенный Tomcat или Jetty (не требуется установки WAR файлов);
* обеспечивание начальными POMs для упрощения Maven конфигурации;
* автоматическая конфигурация Spring, когда это возможно;
* обеспечивание такими возможностями, как метрики, мониторинг состояниями и расширенная конфигурация.

В огромном и разнообразном фреймворке Spring существует подключаемый модуль *Spring Security* [9], который позволяет разработать очень гибкую систему безопасности для серверной части приложения. На его основе и будет организована аутентификация и авторизация пользователей. Модуль привносит в приложение контекст безопасности, который может хранить полученные из токена данные о клиенте в течении выполнения запроса одним потоком. Также можно удобно использовать этот модуль для добавления фильтров безопасности и гибкого разделения конечных точек сервера по ролям. Это позволяет использовать клиентам с урезанной ролью только часть функционала приложения, а клиентам-администраторам – полный набор конечных точек.

*Сервер приложений*([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) application server) — это программная платформа ([фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)), предназначенная для эффективного исполнения процедур (программ, скриптов), на которых построены приложения. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API ([интерфейс прикладного программирования](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), определённый самой платформой.

В случае [Java](https://ru.wikipedia.org/wiki/Java)-сервера приложений, сервер приложений ведёт себя как расширенная [виртуальная машина](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) для запуска приложений, прозрачно управляя соединениями с базой данных, с одной стороны, и соединениями с веб-клиентом, с другой.

Преимущества API серверов:

* целостность данных и кода;
* централизованная настройка и управление;
* безопасность;
* поддержка транзакций.

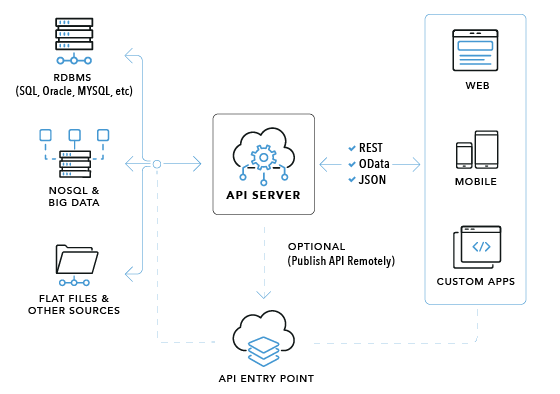


Рисунок 1.5 – Пример работы API сервера

### 1.2.4 Обзор инструментов разработки клиентской части приложения

*React* – это JavaScript библиотека для создания пользовательских интерфейсов. Она была основана в компании Facebook в 2013 году [10]. Можно выделить три основных принципа данной библиотеки:

1. Декларативная. Разработчику достаточно описать, как части приложения выглядят в разных состояниях, а React будет обновлять их, когда они изменятся. Декларативное представление делает код более предсказуемым.
2. Основана на компонентах. Есть возможность создавать инкапсулированные компоненты с состояниями, а затем объединять в общие интерфейсы.
3. Повторное использование кода. Используя React, можно разрабатывать части приложения, абстрагируясь от других компонентов, что делает разработку в команде удобнее.

React использует виртуальное Document Object Model (DOM). Во время работы, React создает кэш структуру в памяти, которая позволяет вычислять разницу между текущим и предыдущим состоянием интерфейса для оптимального обновления DOM. Синтаксис, который используется в React – расширенный JavaScript.

В React используется однонаправленная передача данных. Данные передаются от родительских компонентов к дочерним. Все свойства являются неизменяемыми, однако изменить их можно с помощью callback функций. Такой подход называется «свойства вниз, события вниз».

*NodeJS* – серверная программная платформа, которая позволяет работать с JavaScript через его движок. Движок, который используется в NodeJS – V8 [11].

*V8* – движок, который компилирует JavaScript в машинный код, пропуская стадию компиляции в промежуточный байт-код [12].

*ExpressJS* – это популярный, быстрый и гибкий MVC NodeJS фреймворк, который предлагает широкий набор функций для веб-приложений, а также для разработки мобильных приложений [13]. Он имеет API, который позволяет пользователям настраивать маршруты для отправки и получения запросов между frontend и базой данных. Кроме того, Express поставляется с системой просмотра, которая поддерживает более 14 шаблонизаторов и согласование контента.

Redux – это библиотека для управления глобальным состоянием приложения. Библиотека возникла на основе шаблона Flux. Flux – это архитектурный подход, который состоит из 4 компонент:

«Действие» – вспомогательная функция, которая упрощает передачу данных Диспетчеру.

«Диспетчер» – принимает Действия и рассылает данные обработчикам.

«Хранилище» – контейнер, в котором находится состояние приложения.

«Представление» – React-компоненты, которые собирают состояния из хранилищ и передают дочерним компонентам.

Принцип работы Flux представлен на рисунке 1.6.

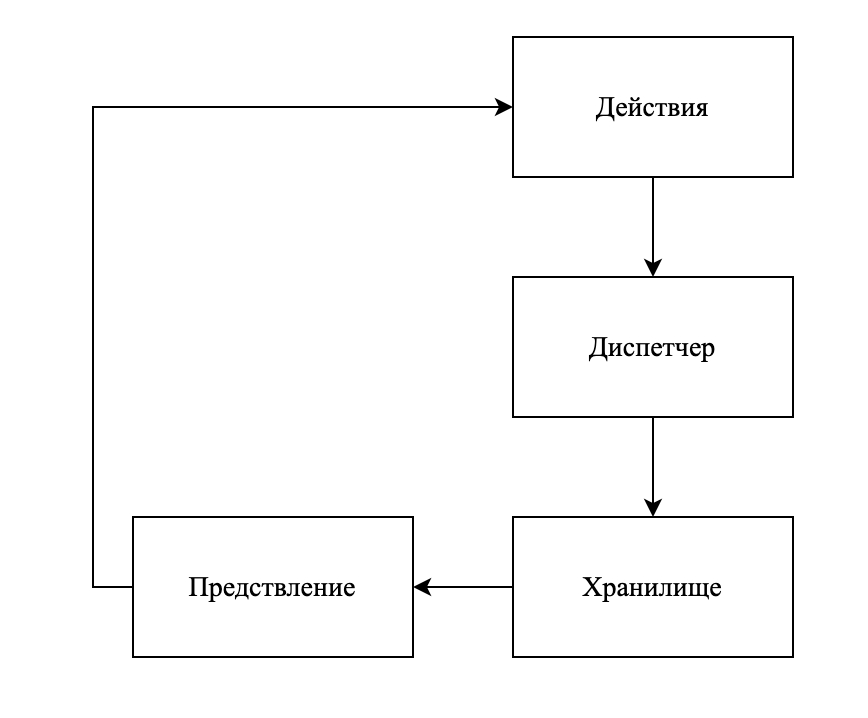


Рисунок 1.6 – Принцип работы Flux

«Действие» – это такая функция, которая вызывается из «Представления», чтобы отправить действие «Диспетчеру». Для различения действий используется константный подход, эта константа посылается вместе с данными действия.

«Диспетчер» – это главный узел процесса. Он получает данные и отправляет их всем зарегистрированным действиям, в свою очередь обработчики вызываются в определенном порядке, а далее действия могут быть обработаны в хранилище и состояние приложения будет изменено. Пример работы «Диспетчера» приведен на рисунке 1.7.

Функции, которые вызываются в «Диспетчерах» должны быть «чистыми».

Чистая функция – это такая функция, которая является детерминированной и не обладает сторонним эффектами.

Детерминированная функция – это такая функция, которая для одного и того же набора входных значений будет возвращен одинаковый результат.

Стоит отметить, что к сторонним эффектам можно отнести такие вещи, как: операции ввода-вывода, реагирование на исключительные ситуации, вызов обработчиков исключительных ситуаций, модификация глобальных переменных.

Существует так же другой тип побочных эффектов – это изменение входных значений в процессе работы функции.

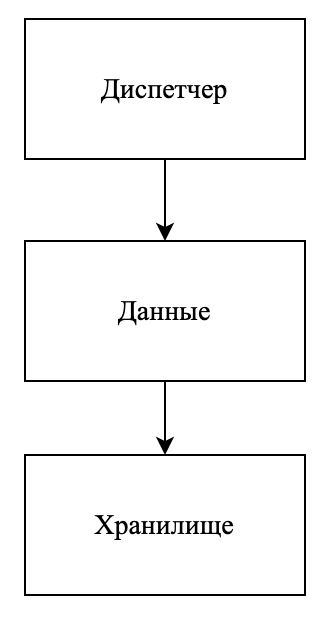


Рисунок 1.7 – Принцип работы «Диспетчера»

Хранилище во Flux занимается управлением состояниями приложения. Хранилище хранит данные и методы для получения данных. Подробная роль Хранилища во Flux представлена на рисунке 1.8.

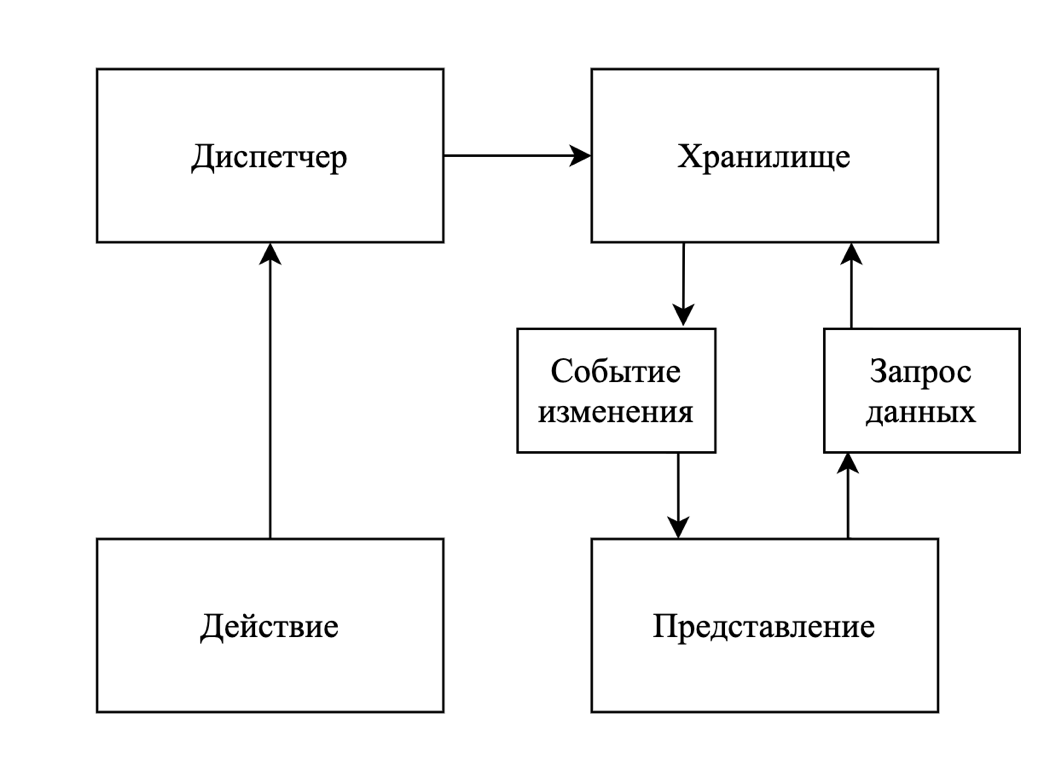


Рисунок 1.8 – Роль Хранилища во Flux

Как видно из рисунка, «Представление» подписано на хранилище. «Действие» вызывает «Диспетчера», в котором происходит обработка данных. Данные изменяются и кладутся в хранилище, как только произошло изменение «Представление» оповещается об этом с помощью события.

«Представление» – это компонента, написанная на React, которые подписываются на события и получают из хранилища состояние приложения, связанные с этим компонентом, каждый раз, когда оно обновляется.

Стоит отметить, что подпоясывание на «Хранилище» происходит в главном представлении, а далее данные передаются по свойствам дочерних «представлений».

Рисунок, с «Представлением» в шаблоне Flux представлен на рисунке 1.9.

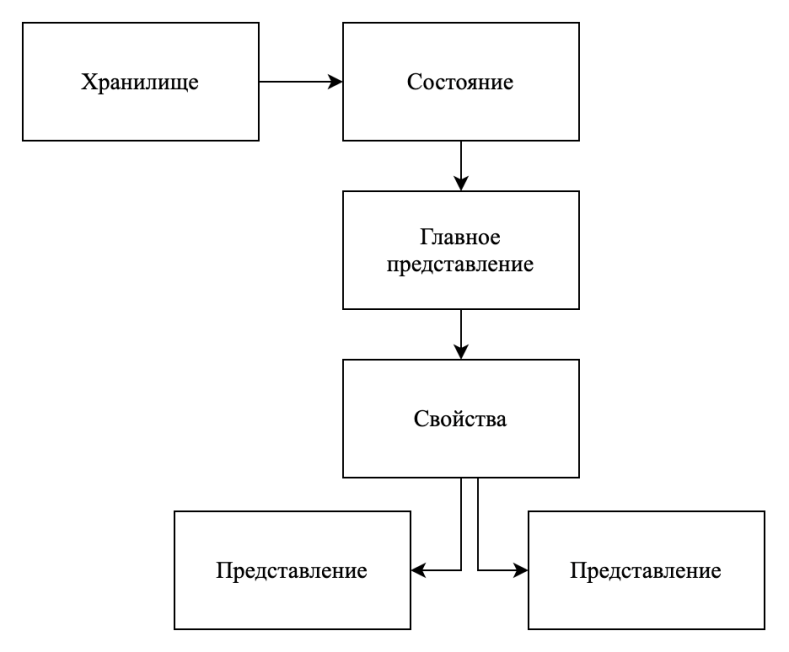


Рисунок 1.9 – Роль «Представления» во Flux

### 1.2.5 Обзор систем управления базами данных

*PostgreSQL* – реляционная система для управления базами данных. В реляционных базах данных информация сведена в таблицы, столбы и строки [14].

Плюсы реляционной базы данных:

* удобный инструмент для работы приложений в реальном времени;
* отлично подходит для обработки больших объемов данных.

Минусы реляционной базы данных:

* медленный доступ к данным;
* трудоемкость разработки.

PostgreSQL активно обновляется и дорабатывается, а также имеет множество расширений, которые добавляют поддержку новых типов данных, возможность версионирования, поддержку работы с географическими координатами, возможности полнотекстового поиска и многое другое. Возможно, многие расширения не будут использоваться при разработке программного обеспечения, но не стоит забывать о вероятном добавлении нового функционала, где они могут пригодиться. Модульность, активная доработка новых версий, открытость и простота использования продукта – сильные стороны PostgreSQL.

Для языка Java есть технология, которая обеспечивает объектно-реляционное отображение простых Java объектов и предоставляющая API для сохранения, получения и управления такими объектами – *JPA*[15].

JPA – это спецификация. Сам JPA не умеет ни сохранять, ни управлять объектами, а только определяет правила игры: как что будет действовать. Также, JPA определяет интерфейсы, которые должны будут быть реализованы провайдерами.

Самая популярная реализация спецификации JPA – *Hibernate*[16].

Данная библиотека не только решает задачу связи классов Java с таблицами базы данных, но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC- кода.

## 1.3 Постановка задачи

Исходя из выделенных недостатков существующих аналогов и основываясь на цели дипломного проекта, были поставлены следующие задачи:

* построение архитектуры приложения;
* разработка RESTful API сервера;
* разработка клиентского приложения;
* разработка и конфигурация СУБД.

Приложение в конечном виде будет обладать следующими возможностями:

* составление заказа;
* продажа товара;
* складской учет (поставки, убытки, списания);
* управление каталогом товаров;
* управление персоналом;
* создание акций на товар;
* статистика продаж, рабочих смен, сотрудников.

Для реализации перечисленных возможностей приложение будет представлять собой два блока: терминал и админ-панель.

Терминал предназначен для работы флористов с заказами. Админ-панель предназначена для администрирования и отслеживания всевозможной статистики.

## 1.4 Обоснование выборки средств разработки

Было принято решение реализовать приложение в виде двух отдельных модулей: RESTful API backend сервера и frontend клиента.

Языком программирования для backend выбран Java, фреймворком – Spring. Причиной выбора именно этих средств разработки послужили простота и удобство конфигурации, а также опыт в использовании их в других проектах. Также, для упрощения и ускорения процесса разработки принято использовать Spring Boot, как набор утилит, предоставленный разработчиками Spring специально для веб-приложений. Для реализации системы аутентификации и авторизации выбран подключаемый модуль Spring Security.

Языком программирования для frontend выбран JavaScript, библиотека – React. React выбран в целях ознакомления как одна из самых востребованных технологий, и как самая трендовая технология на StackOverflow. Фреймворком для клиентской части был выбран ExpressJS, так как он является одним из самых производительных, обладает гибкостью при разработке и предоставляет обширный набор функций как для мобильных, так и для веб-приложений.

Реляционной базой выбрана PostgreSQL, так как она не является проприетарной, поддерживает контейнеризацию и пользуется популярностью среди разработчиков.

Для связи ООП и реляционной базы данных выбран Hibernate. Hibernate является наиболее популярной реализацией ORM-модели и прост в конфигурации и внедрении в проект.

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Поставив задачи, а также обосновав выбор средств разработки, разбиваем приложение на модули:

* модуль работы backend;
* модуль работы frontend;
* модуль работы с данными;
* модуль терминала;
* модуль админ-панели;
* модуль базы данных;
* модуль авторизации для терминала;
* модуль авторизации для админ-панели.

## 2.1 Модуль работы backend

Модуль работы backend сервера выполняет следующие задачи:

* получение и обработка HTTP-запросов;
* взаимодействие с базой данных;
* обработка статистических данных;
* авторизация пользователей.

Модули серверов в разрабатываемом приложении является ключевыми, так как включают в себя многие другие модули, которые позволяют функционировать приложению.

С точки зрения кода программы, он содержит в себе все необходимые для функционирования сервера конфигурационные классы, файлы настроек и каркас API.

При разработке backend сервера был использован фреймворк Spring, а точнее – набор утилит Spring Boot. Он позволяет автоматизировать настройку и ускорить процесс создания и развертывания Spring-приложения.

Автоматическая конфигурация позволяет разработчикам не изобретать колесо каждый раз, когда они создают новое приложение.

## 2.2 Модуль работы frontend

Модуль работы frontend сервера предназначен для отправки HTTP-запросов API серверу и обработки получаемых ответов. Данный модуль содержит в себе конфигурации и файлы настроек, необходимые для функционирования сервера.

При разработке frontend сервера в основу лег такой фреймворк, как ExpressJS, а основная платформа разработки NodeJS.

## 2.3 Модуль работы с данными

База данных содержит в себе всю необходимую для функционирования приложения информацию: пользователи, магазины, терминалы, чеки, товары и т.п.

Модуль взаимодействует с базой данных посредством SQL запросов, отправляемых сервером. В зависимости от запроса, модуль позволяет получать, сохранять, изменять или удалять данные.

Этот программный модуль имеет свой репозиторий (программная абстракция, которая взаимодействует с базой данных) для осуществления всех необходимых операций. Связь с ядром серверной части можно представить на примере получения списка пользователей по заданным параметрам фильтрации:

* запрос попадает в контроллер сервера;
* происходит авторизация пользователя;
* параметры запроса и передаются в соответствующий метод модуля работы с данными персонала;
* модуль подготавливает и выполняет запрос в БД, после чего возвращает структуру данных обратно в контроллер.

Для упрощения взаимодействия с базой данных был использован Hibernate.

При разработке сервера был сделан выбор в сторону PostgreSQL. PostgreSQL – реляционная система для управления базами данных. В реляционных базах данных информация сведена в таблицы, столбы и строки.

## 2.4 Модуль терминала

Модуль терминала является рабочей средой флористов и предназначен для осуществления продаж товара и его учета.

Терминал предусматривает следующие возможности:

* обработка заказа;
* управление циклом кассовой смены и смены сотрудника;
* создание транзакций (расход, приход, инкассация);
* формирование букета;
* управление списком заказов.

Внешний графический интерфейс будет реализован на языке JavaScript с использованием фреймворка React.

В основу разработки легла библиотека ReactJS, а для управления состоянием было принято решение использовать Redux в совокупности с Redux-Saga и Reselect. Также было принято решение использовать библиотеку с компонентами «Core UI». Эта библиотека предоставляет набор гибких компонент, для создания панели администратора.

## 2.5 Модуль админ-панели

Модуль админ-панели является основным инструментом для администрирования магазина, а также отслеживания информации о продажах, активности сотрудников, складских накоплениях.

Админ-панель включает в себя следующие разделы:

* статистика;
* финансы;
* товары;
* склад;
* доступ.

В разделе статистики предоставляется информация по продажам магазина, товаров, эффективности продавцов, просмотр отдельных чеков.

В разделе финансы отображаются все выполненные в терминале транзакции, смены сотрудников, устанавливаются и рассчитываются зарплаты сотрудников.

В разделе товары осуществляется формирование доступного ассортимента, возможность добавления товаров в категории товаров.

В разделе склад происходит управление складскими запасами магазина, осуществление поставок, списания, отслеживание остатка.

В разделе доступ осуществляется управление персоналом, установка паролей доступа к терминалу.

Технологии, которые легли в основу разработки модуля точно такие, как и для терминала.

## Модуль авторизации для терминала

Модуль авторизации для терминала предназначен для идентификации магазина и флориста, осуществляющего продажи, с целью сохранения его активности.

Для авторизации в терминале использован метод Basic Authorization. При успешном логине, после обращения к API, клиенту приходит закодированная base64 строка, которую клиент сохраняет в заголовке Authorization. В свою очередь сервер сохраняет в базе связку данных логина и сгенерированной строки.

При последующих обращениях к API endpoints, связанных с модулем терминала, доступ к ресурсу будет проверятся считыванием строки, содержащимся в заголовке Authorization.

Жизнь сгенерированного токена определяется временем записи в таблицу на сервере, и установлено на 24 часа – каждый час производится проверка строк таблицы на валидность. Это обусловлено ежедневной сменой кадров.

Принято решение разделить авторизацию для терминала и для админ-панели, из-за необходимости привязывать каждое подключение к рабочей смене сотрудника, что не имеет смысла для админ-панели.

## 2.7 Модуль авторизации для админ-панели

Данный модуль авторизации является ключевым, с точки зрения безопасности, так как предназначен для осуществления доступа владельца магазина к конфиденциальной информации, а также администрированию магазином.

Для авторизации был использован JSON Web Token (JWT). JWT – стандарт для создания токенов доступа. Все данные, которые в нем находятся хранятся в формате JSON. Такой токен используется для передачи данных от сервера к клиенту.

В Spring, за генерацию токенов отвечает библиотека – Java JWT, которая подключается с помощью maven-зависимости java-jwt. Она предоставляет большое API для работы с авторизации в целом. В приложении используется только 2 основные функции – генерации и проверка на валидность.

Стоит также выделить основные и базовые параметры, которые хранит в себе любой JWT токен:

* время создания;
* время окончания действия;
* домен, с которого он был создан;
* название шифровочного алгоритма.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Целью функционального проектирования является разработка функционала программного продукта. В этот этап входит разработка классов, которые позволят реализовать требуемые функции. Определяются конкретные поля и методы каждого из классов, задачи, которые будут на них возложены.

Раздел определяет функционирование разрабатываемой программы и исчерпывающую информацию о ее структуре с точки зрения описания данных и обрабатывающих их функций. Описывается структура таблиц и связи между ними.

## 3.1 Модуль работы backend

Модуль работы backend содержит классы обработки HTTP-запросов, защиты приложения от несанкционированного доступа и настройку фреймворка Spring. Классы данного модуля вместе со всеми другими классами серверной части проекта представлены на чертеже ГУИР.400201.023 РР.1.

Класс MainConfiguration, предназначен для общих настроек приложения. Содержит следующие методы:

* corsConfigurer – предоставляет доступ к получению ответы на запросы веб-сайту, который сгенерировал AJAX запрос к серверу. Этот метод отключает ограничительную меру, придуманную разработчиками браузеров. Также, метод переопределяет свойство origins, в котором устанавливается адрес, на котором развернут пользовательский интерфейс, что делает невозможным получить информацию от запросов при обращении со сторонних серверов;
* passwordEncoderBean – возвращает объект класса, выполняющий шифрование паролей перед их сохранением в базу.

Класс SwaggerConfig необходим для настройки Swagger – библиотеки, которая предоставляет функции документирования API серверной части приложения. Этот инструмент поддерживает формализованную спецификацию OpenAPI, что позволяет описать структуру API сервиса в виде YAML/YML или JSON файла, а затем отобразить ее в окне интернет браузера. При подключении к приложению, библиотека предоставляет возможность отправки и формирования HTTP-запросов, в том числе заголовки и тело запроса. После отправки отображается ответ, полученный со стороны сервера. Класс содержит методы:

* metaInfo – метод, который устанавливает информацию о названии, описании, контактной информации для автоматически генерируемой документации;
* productApi – метод, устанавливающий базовый пакет, который будет считываться для генерации документации, и URL, по которому развернуто приложение.

Класс WebSecurityConfig предназначен для настройки библиотеки Spring Security, который отвечает за защиту от несанкционированного доступа. Класс содержит метод:

* configure – метод, который принимает на вход объект класса HttpSecurity. В методе можно определить те запросы, которые могут быть выполнены без авторизации. Также можно указать множество других требований, выполнение которых необходимо для получения положительного результата запроса. В конфигурации отключается механизм встроенной защиты от CSRF, так как в рамках данного приложения защита реализована с применением технологии JWT-токенов.

Классы-контроллеры, предназначенные для обработки HTTP-запросов, поступающих на сервер:

* TerminalAuthController;
* TerminalController;
* AdminPanelAuthController;
* AdminPanelController;
* OrderController.

Класс TerminalAuthController обрабатывает запросы, связанные с реализацией защиты приложения. Для этого используется технология JWT-токенов. Класс генерирует токены доступа после успешной аутентификации. Особенностью данного контролера является невозможность регистрации новых терминалов или флористов напрямую. Права на регистрацию принадлежат контроллеру админ-панели.

Содержит следующие поля:

* authHeaderName – строка, содержащая имя заголовка, с которого необходимо производить чтение токена. В рамках данного проекта недостаточно стандартного заголовка Authentication, так как необходимо производить аутентификацию как для флористов в терминале, так и для владельцев в админ-панели, алгоритмы которых отличны друг от друга;
* floristRepository – объект, работающий с сущностью Florist и предоставляющий доступ к таблице базы данных, созданной этой сущностью;
* terminalRepository – объект, работающий с сущностью Terminal и предоставляющий доступ к таблице базы данных, созданной этой сущностью;
* authTerminalService – объект, содержащий логику обработки каждого запроса.

Класс контроллера содержит в себе следующие методы:

* preLogin – метод обработки POST запроса, которому соответствует URL /terminal/auth/preLogin. Принимает на вход объект, содержащий логин и пароль терминала. Вызывает метод класса-сервиса для проверки наличия необходимого терминала. В случае нахождения терминала, происходит возвращение ответа со статусом 200 (Ok), содержащего объект, аналогичный пришедшему на вход;
* login – метод обработки POST запроса, которому соответствует URL /terminal/auth/login. Принимает на вход объект, содержащий логин и пароль терминала, а также пароль флориста, которым осуществляется вход в терминал. После удачного нахождения терминала и флориста вызывается метод генерации токена, который будет отправлен в виде объекта. Если же идентифицировать терминал или флориста не вышло, возвращается пустой ответ со статусом 401 (Unauthorized);
* checkAccess – метод обработки GET запроса, осуществляющий проверку валидности токена, записанного в необходимом заголовке. Методу соответствует URL /terminal/auth/checkAccess.

Класс TerminalController предназначен для обработки запросов, происходящих в терминале вне работы с заказами. Класс содержит следующие поля:

* authHeaderName – строка, содержащая имя заголовка, с которого необходимо производить чтение токена;
* authTerminalService – объект, содержащий логику проверки токена на доступ к запросам;
* terminalService – объект, содержащий логику обработки каждого запроса.

Класс контроллера содержит следующие методы:

* getFloristName – метод обработки GET запроса, которому соответствует URL /terminal/getFloristName. Возвращает имя флориста, произведшего авторизацию в терминале;
* getCheckById – метод обработки GET запроса, которому соответствует URL /terminal/getCheck/{id}, где id – идентификатор чека, информация о котором должна быть получена;
* getTerminalData – метод обработки GET запроса, которому соответствует URL /terminal/getTerminalData. Возвращает информацию о терминале, в который произведен вход;
* startWorkShift – метод обработки POST запроса, которому соответствует URL /terminal/startWorkShift. Инициализирует рабочую смену флориста, которым был произведен вход в текущий треминал;
* endWorkShift – метод обработки POST запроса, которому соответствует URL /terminal/endWorkShift. Завершает рабочую смену флориста, которым был произведен вход в текущий терминал. Данный метод автоматически вызывается при выходе флористом из терминала;
* makeTransaction – метод обработки POST запроса, которому соответствует URL /terminal/makeTransaction. Принимает на вход объект TransactionForm, содержащий тип транзакции (расход, приход или инкассация), а также необходимую информацию о транзакции. Вызывается метод генерации транзакции внутри текущего терминала. В случае успешного выполнения транзакции возвращает пустой ответ со статусом 200 (Ok).

Класс AdminPanelAuthController обрабатывает запросы, связанные с реализацией защиты приложения. Он генерирует токены доступа после успешной аутентификации. По назначению класс схож с контроллером TerminalAuthController и отличается лишь тем, что предоставляет доступ в админ-панель, у которого отличается алгоритмавторизации. Класс содержит следующие поля:

* authHeaderName – строка, содержащая имя заголовка, с которого необходимо производить чтение токена. Значения этой строки и аналогичной сроки в контроллере TerminalAuthController; должны разниться;
* ownerRepository – объект, работающий с сущностью Owner и предоставляющий доступ к таблице базы данных, созданной этой сущностью;
* authAdminPanelService – объект, содержащий логику обработки каждого запроса.

Класс контроллера содержит методы:

* login – метод обработки POST запроса, которому соответствует URL /adminPanel/auth/login. Принимает на вход объект, содержащий логин и пароль владельца, которым осуществляется вход в админ-панель. После удачного нахождения терминала и флориста вызывается метод генерации токена, который будет отправлен в виде объекта. Если же идентифицировать владельца не вышло, возвращается пустой ответ со статусом 401 (Unauthorized);
* checkAccess – метод обработки GET запроса, осуществляющий проверку валидности токена, записанного в необходимом заголовке. Методу соответствует URL /adminPanel/auth/checkAccess.

Класс AdminPanelController предназначен для обработки запросов, происходящих в админ-панели. Контроллер содержит поля:

* authHeaderName – строка, содержащая имя заголовка, с которого необходимо производить чтение токена;
* authAdminPanelService – объект, содержащий логику проверки токена на доступ к запросам;
* adminPanelService – объект, содержащий логику обработки каждого запроса.

Класс контроллера содержит в себе следующие методы:

* getOwnerData – метод обработки GET запроса, которому соответствует URL /adminPanel/getOwnerData. Возвращает информацию о владельце, выполнившему авторизацию в админ-панели;
* getAdminPanelData – метод обработки GET запроса, которому соответствует URL /adminPanel/getAdminPanelData. Возвращает информацию о админ-панели, в которою произведен вход;
* getShops – метод обработки GET запроса, которому соответствует URL /adminPanel/getShops. Возвращает список магазинов, которые принадлежат владельцу, выполнившему авторизацию;
* getStorages – метод обработки GET запроса, которому соответствует URL /adminPanel/getStorages. Возвращает список складов, которые принадлежат владельцу, выполнившему авторизацию;
* addCategory – метод обработки POST запроса, которому соответствует URL /adminPanel/addCategory. Принимает на вход объект CategoryDto, содержащий информацию о категории товара, которую необходимо добавить. В случае успешного добавления категории возвращает пустой ответ со статусом 201 (Created);
* addProduct – метод обработки POST запроса, которому соответствует URL /adminPanel/addProduct. Принимает на вход объект ProductDto, содержащий информацию о товаре, который необходимо добавить. В случае успешного добавления товара возвращает пустой ответ со статусом 201 (Created);
* deleteCategory – метод обработки DELETE запроса, которому соответствует URL /adminPanel/deleteCategory. Принимает на вход объект CategoryDto, содержащий информацию о категории товара, которую необходимо удалить. В случае успешного удаления категории товара возвращает пустой ответ со статусом 200 (Ok);
* deleteProduct – метод обработки DELETE запроса, которому соответствует URL /adminPanel/deleteProduct. Принимает на вход объект ProductDto, содержащий информацию о товаре, который необходимо удалить. В случае успешного удаления товара возвращает пустой ответ со статусом 200 (Ok);
* getFlorists – метод обработки GET запроса, которому соответствует URL /adminPanel/getFlorists. Возвращает список флористов, которые принадлежат владельцу, выполнившему авторизацию;
* addFlorist – метод обработки POST запроса, которому соответствует URL /adminPanel/addForist. Принимает на вход объект FloristDto, содержащий информацию о флористе, которого необходимо добавить. В случае успешного добавления флориста возвращает пустой ответ со статусом 201 (Created). Впоследствии флорист может авторизоваться в терминале под паролем, установленным владельцем при добавлении флориста;
* changeFlorist – метод обработки POST запроса, которому соответствует URL /adminPanel/changeForist. Принимает на вход объект FloristDto, содержащий необходимую для изменения информацию о флористе. В случае успешного изменения данных о флористе возвращает пустой ответ со статусом 200 (Ok);
* deleteFlorist – метод обработки DELETE запроса, которому соответствует URL /adminPanel/deleteFlorist. Принимает на вход объект FloristDto, содержащий информацию о флористе, которого необходимо удалить. В случае успешного удаления флориста возвращает пустой ответ со статусом 200 (Ok);
* makeTransaction – метод обработки POST запроса, которому соответствует URL /adminPanel/makeTransaction. Принимает на вход объект AnyTransactionForm, содержащий тип транзакции (расход, приход или инкассация), а также необходимую информацию о транзакции. Отличается от создания транзакции в терминале тем, что не привязан к конкретному терминалу и может быть выполнен для любого магазина, владельцем которого является авторизованный в админ-панели пользователь. В случае успешного выполнения транзакции возвращает пустой ответ со статусом 200 (Ok).
* getCategories – метод обработки GET запроса, которому соответствует URL /adminPanel/getCategories. Возвращает список категорий товаров, когда-либо созданных владельцем, выполнившим авторизацию;
* getProducts – метод обработки GET запроса, которому соответствует URL /adminPanel/getProducts. Возвращает список товаров, когда-либо созданных владельцем, выполнившим авторизацию;
* getStatisticsOfShopByDay – метод обработки GET запроса, которому соответствует URL /adminPanel/ getStatisticsOfShopByDay. Принимает на вход объект, содержащий день и идентификатор магазина, статистику которого необходимо получить. Возвращает список объектов StatisticsDto для магазина, принадлежащего владельцу, выполнившему авторизацию;
* getStatisticsOfShopByWeek – метод обработки GET запроса, которому соответствует URL /adminPanel/ getStatisticsOfShopByWeek. Принимает на вход объект, содержащий неделю и идентификатор магазина, статистику которого необходимо получить. Возвращает список объектов StatisticsDto для магазина, принадлежащего владельцу, выполнившему авторизацию;
* getStatisticsOfShopByMonth – метод обработки GET запроса, которому соответствует URL /adminPanel/ getStatisticsOfShopByMonth. Принимает на вход объект, содержащий месяц и идентификатор магазина, статистику которого необходимо получить. Возвращает список объектов StatisticsDto для магазина, принадлежащего владельцу, выполнившему авторизацию;
* getStatisticsOfShopByCustomTime – метод обработки GET запроса, которому соответствует URL /adminPanel/ getStatisticsOfShopByCustomTime. Принимает на вход объект, содержащий временной промежуток и идентификатор магазина, статистику которого необходимо получить. Возвращает список объектов StatisticsDto для магазина, принадлежащего владельцу, выполнившему авторизацию;
* getStatisticsOfShopByDay – метод обработки GET запроса, которому соответствует URL /adminPanel/ getStatisticsOfAllShopByDay. Принимает на вход объект, содержащий только день для сбора статистики. Возвращает список объектов StatisticsDto для всех магазинов, принадлежащих владельцу, выполнившему авторизацию;
* getStatisticsOfShopByWeek – метод обработки GET запроса, которому соответствует URL /adminPanel/ getStatisticsOfAllShopByWeek. Принимает на вход объект, содержащий только неделю для сбора статистики. Возвращает список объектов StatisticsDto для всех магазинов, принадлежащих владельцу, выполнившему авторизацию;
* getStatisticsOfShopByMonth – метод обработки GET запроса, которому соответствует URL /adminPanel/ getStatisticsOfAllShopByMonth. Принимает на вход объект, содержащий только месяц для сбора статистики. Возвращает список объектов StatisticsDto для всех магазинов, принадлежащих владельцу, выполнившему авторизацию;
* getTransactionsOfShop – метод обработки GET запроса, которому соответствует URL /adminPanel/getTransactionsOfShop. Принимает на вход объект, содержащий идентификатор магазина. Возвращает список объектов TransactionDto, содержащий все выполненные в конкретном магазине транзакции. В случае попытки получения списка транзакций чужого магазина, возвращает пустой ответ со статусом 403 (Forbidden);
* getAllTransactions – метод обработки GET запроса, которому соответствует URL /adminPanel/getAllTransactions. Возвращает список объектов TransactionDto, содержащий все транзакции всех магазинов, владельцем которых является авторизованный в админ-панели пользователь;
* getWorkShiftsOfFlorist – метод обработки GET запроса, которому соответствует URL /adminPanel/getWorkShiftsOfFlorist. Принимает на вход идентификатор флориста, рабочие смены которого необходимо получить. Возвращает список объектов WorkShiftDto, содержащий все рабочие смены флориста. В случае попытки получения рабочих смен флориста, не принадлежащего владельцу, возвращает пустой ответ со статусом 403 (Forbidden);
* getAllWorkShifts – метод обработки GET запроса, которому соответствует URL /adminPanel/getAllWorkShifts. Возвращает список объектов WorkShiftDto, содержащий все рабочие смены всех флористов, когда-либо созданных авторизованным в админ-панели владельцем;
* getStocksOfStorage – метод обработки GET запроса, которому соответствует URL /adminPanel/getStocksOfStorage. Принимает идентификатор склада, накопления которого необходимо получить. Возвращает список объектов StockDto, содержащий все складские накопления конкретного склада, принадлежащего авторизованному владельцу;
* getStocksOfAllStorages – метод обработки GET запроса, которому соответствует URL /adminPanel/getStocksOfAllStorages. Возвращает список объектов StockDto, содержащий все складские накопления всех складов, принадлежащих авторизованному владельцу;
* addStock – метод обработки POST запроса, которому соответствует URL /adminPanel/addStock. Принимает на вход объект StockDto, содержащий информацию о складском накоплении, которое необходимо добавить в конкретный склад. В случае успешного добавления накопления возвращает пустой ответ со статусом 201 (Created);
* changeStock – метод обработки POST запроса, которому соответствует URL /adminPanel/changeStock. Принимает на вход объект StockDto, содержащий информацию о складском накоплении, которое необходимо изменить в конкретном складе. В случае успешного изменения данных о накоплении возвращает пустой ответ со статусом 200 (Ok);
* deleteStock – метод обработки DELETE запроса, которому соответствует URL /adminPanel/deleteStock. Принимает на вход объект StockDto, содержащий информацию о складском накоплении, которое необходимо удалить. В случае успешного удаления накопления возвращает пустой ответ со статусом 200 (Ok).

Класс OrderController предназначен для обработки запросов, происходящих в терминале и связанных с работой с заказами. Класс содержит следующие поля:

* authHeaderName – строка, содержащая имя заголовка, с которого необходимо производить чтение токена;
* authTerminalService – объект, содержащий логику проверки токена на доступ к запросам;
* orderService – объект, содержащий логику обработки каждого запроса.

Класс контроллера содержит в себе следующие методы:

* getOrders – метод обработки GET запроса, которому соответствует URL /terminal/getOrders. Возвращает список объектов заказа OrderDto, когда-либо созданных в текущем терминале авторизованным флористом;
* addOrder – метод обработки POST запроса, которому соответствует URL /terminal/addOrder. Принимает объект типа OrderDto, содержащий информацию о заказе, который необходимо добавить. В случае успешного добавления заказа возвращает пустой ответ со статусом 201 (Created);
* changeOrder – метод обработки POST запроса, которому соответствует URL /adminPanel/changeOrder. Принимает на вход объект OrderDto, содержащий необходимую для изменения информацию о заказе. В случае успешного изменения данных о заказе возвращает пустой ответ со статусом 200 (Ok);
* deleteOrder – метод обработки DELETE запроса, которому соответствует URL /adminPanel/deleteOrder. Принимает на вход объект OrderDto, содержащий информацию о заказе, который необходимо удалить. В случае успешного удаления заказа возвращает пустой ответ со статусом 200 (Ok);
* findById – метод обработки GET запроса, которому соответствует URL /terminal/findById/{id}, где id – идентификатор заказа, информация о котором должна быть получена;
* findByDate – метод обработки GET запроса, которому соответствует URL /terminal/findByDate. Принимает на вход значение параметра value, содержащее искомую дату. Возвращает список найденных по дате заказов, когда-либо созданных авторизованным флористом;
* findByTime – метод обработки GET запроса, которому соответствует URL /terminal/findByTime. Принимает на вход значение параметра value, содержащее искомое время доставки. Возвращает список найденных по времени доставки заказов, когда-либо созданных авторизованным флористом;
* findByCustomer – метод обработки GET запроса, которому соответствует URL /terminal/findByCustomer. Принимает на вход значение параметра value, содержащее искомое имя заказчика. Возвращает список найденных по имени заказчика заказов, когда-либо созданных авторизованным флористом;
* findByPrice – метод обработки GET запроса, которому соответствует URL /terminal/findByPrice. Принимает на вход значение параметра value, содержащее искомую цену. Возвращает список найденных по цене заказов, когда-либо созданных авторизованным флористом;
* sortByPriceDesc – метод обработки GET запроса, которому соответствует URL /terminal/sortByPriceDesc. Возвращает список заказов, когда-либо созданных авторизованным флористом, отсортированных по цене по убыванию;
* sortByPriceAsc – метод обработки GET запроса, которому соответствует URL /terminal/sortByPriceAsc. Возвращает список заказов, когда-либо созданных авторизованным флористом, отсортированных по цене по возрастанию;
* sortByDateAndTimeDesc – метод обработки GET запроса, которому соответствует URL /terminal/sortByDateAndTimeDesc. Возвращает список заказов, когда-либо созданных авторизованным флористом, отсортированных по дате и времени доставки по убыванию;
* sortByDateAndTimeAsc – метод обработки GET запроса, которому соответствует URL /terminal/sortByDateAndTimeAsc. Возвращает список заказов, когда-либо созданных авторизованным флористом, отсортированных по дате и времени доставки по возрастанию;
* getSorts – метод обработки GET запроса, которому соответствует URL /terminal/getSorts. Возвращает список сортировок заказа SortsDto. Предназначен для упрощения формирования запросов с клиентской стороны;
* getSearches – метод обработки GET запроса, которому соответствует URL /terminal/getSearches. Возвращает список объектов из типов поиска заказа SearchesDto. Предназначен для упрощения формирования запросов с клиентской стороны;
* getCategories – метод обработки GET запроса, которому соответствует URL /terminal/getCategories. Возвращает список объектов из категорий товаров CategoryDto;
* getProducts – метод обработки GET запроса, которому соответствует URL /terminal/getProducts. Принимает на вход значение параметра value, содержащее идентификатор категории товара, в которую должны входить искомые товары. Возвращает список объектов из товаров ProductDto.

Класс JwtTerminalTokenProvider предназначен для реализации создания и чтения JWT токенов в терминале. Класс содержит в себе поля:

* secret – строка, содержащая в себе секретное слово для шифрования токенов;
* validityInMilliseconds – переменная типа long, определяющая время жизни токенов.

Основными методами класса являются:

* resolveToken – метод, принимающий на вход строку, содержащуюся в заголовке запросов и возвращающий исключительно значение токена;
* validateToken – метод, принимающий на вход токен и, используя секретную строку для шифрования, проверяет токен на подлинность. Также проверяется время жизни токена. В случае успешной проверки токена возвращается значение true типа boolean, в противном случае возвращается значение false;
* getClaims – метод, принимающий на вход токен и возвращающий объект типа Jws<Claims>, содержащий в себе данные о флористе и терминале, в который производился вход при генерации токена;
* createToken – метод, принимающий на вход объекты типа Florist и Terminal. Предназначен для генерации токена, в котором записываются идентификаторы флориста и терминала, устанавливается время жизни, время создания. Токен кодируется по алгоритму HS256 секретным словом, указанным в поле secret.

Многие методы контроллеров возвращали DTO-объекты. DTO (Data Transfer Object) – структурный паттерн передачи информации между разными частями приложения. Данный паттерн позволяет решить проблему при отправлении объектов, и используется для преобразования классов типа Entity, которые предназначены для сохранения в базу данных. Проблемой передачи данных классов по сети напрямую является то, что между различными entity могут быть сложные отношения. DTO класс позволяет преобразовать информацию из entity в более удобный для отправки вид. Некоторые свойства entity в dto можно опустить.

В проекте реализовано 17 DTO-классов:

* TerminalLoginDto;
* TerminalTokenDto;
* SortDto;
* SearchDto;
* ProductDto;
* OrderDto;
* FloristDto;
* CategoryDto;
* OwnerDto;
* ShopDto;
* StorageDto;
* AdminPanelDto;
* StatisticsDto;
* TransactionDto;
* WorkShiftDto;
* StockDto;
* TerminalDto.

Среди этих классов самыми часто используемыми являются OrderDto, StatisticsDto, TerminalLoginDto и TerminalTokenDto. О них подробнее ниже.

Класс OrderDto предназначен для передачи клиенту информации о заказе. Содержит следующие поля:

* id – переменная типа Long, идентификатор заказа;
* date – строка, содержащая информацию о дате доставки заказа;
* time – строка, содержащая информацию о времени доставки заказа;
* address – строка, содержащая информацию о адресе доставки заказа;
* price – переменная типа Long, содержащая информацию о стоимости заказа;
* customer – строка, содержащая информацию о имени заказчика;
* customerNumber – строка, содержащая информацию о номере телефона заказчика;
* receiver – строка, содержащая информацию о имени получателя;
* receiverNumber – строка, содержащая информацию о номере телефона получателя;
* paymentMethod – строка, содержащая информацию о методе оплаты заказа;
* notes – строка, содержащая примечания к заказу;
* courier – строка, содержащая информацию о имени курьера, ответственного за доставку заказа;
* payStatus – переменная типа boolean, содержащая статус оплаты.

В DTO не вошли поля объекта Order, имеющие зависимости с другими сущностями (Florist, Terminal, Check). Данный класс предназначен для отображения информации о заказе в табличном виде, и поля сущностей не имеют ценности.

Класс содержит два метода, реализованных для преобразования класса Order в OrderDto:

* from – метод, принимающий объект класса Order. Преобразование происходит при использовании аннотации @Builder от Lombok. Метод возвращает объект класса OrderDto;
* from – метод, принимающий список объектов класса Order и возвращает список объектов класса OrderDto. Метод предназначен для упрощения генерации списка объектов.

Во всех других DTO-классах также реализованы два соответствующих метода, которые используются для преобразования entity сущностей в dto.

Класс TerminalLoginDto применяется при предварительной авторизации в терминал, которая осуществляется перед вводом пароля флориста. Используется как подтверждение успешного прохождения первого этапа авторизации. Содержит два поля:

* login – строка, содержащая информацию о имени терминала;
* password – строка, содержащая информацию о пароле терминала.

Класс TerminalTokenDto применяется при окончании авторизации в терминал и передается как подтверждении успешной авторизации в терминал. Содержит единственное поле

* token – строка, содержащая информацию о сгенерированном JWT-токене.

## 3.2 Модуль работы с данными

Данный модуль содержит в себе классы entity-объектов, которые сохраняются в базу данных, классы интерфейсов, для взаимодействия с базой данных, и классы сервисов, содержащие бизнес-логику обработки входящих HTTP-запросов. Обработку HTTP-запросов можно разделить на две части: обработка HTTP-запросов для терминала и для админ-панели. Последовательности обработки обеих частей представлены в виде диаграмм последовательности на чертежах ГУИР.400201.023 РР.2 и ГУИР.400201.023 РР.3.

Класс BaseEntity представляет класс-предок для Entity классов. Такая реализация возможна благодаря аннотации @MappedSuperclass, которая реализует наследование Entity классов только в модели предметной области без отражения ее в схеме базы данных. Описанные и аннотированные свойства и методы данного класса будут применены в классах-наследниках. Класс содержит следующие поля:

* id – идентификатор объекта в базе данных;
* createdAt – время создания объекта;
* updatedAt – время последнего изменения объекта;
* status – статус строки, отображающий ее доступность.

Все последующие классы наследуют класс BaseEntity.

Класс Category является контейнером для категорий товаров. Содержит следующие поля:

* name – название категории;
* products – поле, содержащее список из объектов класса Product, которое помечено аннотацией @OneToMany, которое в связке с аннотацией @ManyToOne со стороны класса Product позволяет настроить механизм объектно-реляционного отображения;
* shop – поле, содержащее объект класса Shop, которое помечено аннотацией @ManyToOne.

Класс Product является контейнером для товаров. Содержит следующие поля:

* name – название товара;
* price – стоимость товара;
* category – поле, содержащее объект класса Category, которое помечено аннотацией @ManyToOne;
* shop – поле, содержащее объект класса Shop, которое помечено аннотацией @ManyToOne.

Класс Check является контейнером для чеков и содержит следующие поля:

* price – стоимость чека;
* purchases – поле, содержащее список из объект класса Purchase, которое помечено аннотацией @OneToMany.

Класс Order является контейнером для заказов. Содержит следующие поля:

* date – дата доставки заказа;
* time – время доставки заказа;
* address – адрес доставки заказа;
* price – стоимость заказа;
* customer – имя заказчика;
* customerNumber – номер телефона заказчика;
* receiver – имя получателя;
* receiverNumber – номер телефона получателя;
* paymentMethod – метод оплаты заказа;
* notes – примечания к заказу;
* courier – имя курьера, ответственного за доставку заказа;
* payStatus – статус оплаты;
* florist – поле, содержащее объект класса Florist, которое помечено аннотацией @ManyToOne;
* terminal – поле, содержащее объект класса Terimnal, которое помечено аннотацией @ManyToOne;
* check – поле, содержащее объект класса Check, которое помечено аннотацией @ManyToOne.

Класс Owner является контейнером для владельцев. Содержит следующие поля:

* firstName – дата доставки заказа;
* lastName – время доставки заказа;
* userName – адрес доставки заказа;
* password – стоимость заказа;
* email – имя заказчика;
* contacts – номер телефона заказчика;
* shops – поле, содержащее список из объект класса Shop, которое помечено аннотацией @OneToMany.
* storages – поле, содержащее список из объект класса Storage, которое помечено аннотацией @OneToMany.

Класс Stock является контейнером для складских запасов. Содержит следующие поля:

* count – количество товаров в накоплении;
* product – поле, содержащее объект класса Product, которое помечено аннотацией @ManyToOne.
* storage – поле, содержащее объект класса Storage, которое помечено аннотацией @ManyToOne.

Для получения информации из базы данных, приложению необходим набор классов, в которых будет реализовано получение и сохранение информации в базу данных. Данная задача была решена на базе фреймворка Spring Data.

Для доступа к базе данных отсутствуют Java-классы, используются исключительно интерфейсы, расширяющие интерфейс CrudRepository, который предоставляется Spring Data. При расширении данного интерфейса, все определенные в нем стандартные методы поиска, сохранения и удаления сущностей передаются новому интерфейсу. Для некоторых таблиц уже достаточно только этих методов, поэтому многие из реализованных в проекте интерфейсов не имеют собственных методов, а только задают класс сущности, а также класс идентификатора сущности, под которые автоматически генерируется SQL код.

Классы-интерфейсы, используемые для получения доступа к базам данных:

* CategoryRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Category;
* CheckRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Check;
* FloristRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Florist;
* OrderRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Order;
* OwnerRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Owner;
* ProductRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Product;
* PurchaseRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Purchase;
* SearchRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Search;
* ShopRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Shop;
* SortRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Sort;
* StockRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданных сущностью Stock;
* StorageRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Storage;
* TerminalRepository – интерфейс, осуществляющий доступ к таблице базы данных, созданной сущностью Terminal.

Класс-интерфейс OrderRepository является исключением, который предназначен для доступа к таблице базы данных заказа. Содержит в себе декларации новых методов:

* findAllByFloristAndTerminal – метод, извлекающий из базы сущность заказов на основании флориста и терминала;
* findAllByFloristAndTerminalAndDate – метод, извлекающий из базы сущность заказов на основании флориста, терминала и даты доставки заказа;
* findAllByFloristAndTerminalAndTime – метод, извлекающий из базы сущность заказов на основании флориста, терминала и времени доставки заказа;
* findAllByFloristAndTerminalAndCustomer – метод, извлекающий из базы сущность заказов на основании флориста, терминала и заказчика;
* findAllByFloristAndTerminalAndAddress – метод, извлекающий из базы сущность заказов на основании флориста, терминала и адреса;
* findAllByFloristAndTerminalAndPrice – метод, извлекающий из базы сущность заказов на основании флориста, терминала и цены;
* findOneByFloristAndTerminalAndId – метод, извлекающий из базы сущность заказа на основании флориста, терминала и идентификатора.

Бизнес-логика обработки запросов находится в специально выделенных классах-сервисах. Для каждого из классов контроллеров реализованы свои интерфейсы-сервисы:

* TerminalAuthService;
* TerminalService;
* AdminPanelAuthService;
* AdminPanelService;
* OrderService.

Классы, реализующие соответствующие интерфейсы:

* TerminalAuthServiceImpl;
* TerminalServiceImpl;
* AdminPanelAuthServiceImpl;
* AdminPanelServiceImpl;
* OrderServiceImpl.

Методы, содержащиеся в классах-сервисах, совпадают по названиям с аналогичными методами в контроллерах, которые были рассмотрены выше. Отличием являются поля сервисов, которые содержат в себе объекты интерфейсов, осуществляющих доступ к базам данных.

## 3.3 Модуль реляционной базы данных

В ходе анализа предметной области приложения были выделены необходимые сущности, их структура перенесена на таблицы реляционной базы данных, описание которых будет представлено ниже.

Все таблицы имеют некоторое количество одинаковых полей, имеющих специальное назначение. Отличительной особенностью является наличие поля Status. Данное поле используется как альтернатива полному удалению данных из таблицы. Вместо удаления строк из базы значение поля Status устанавливается в Deleted. Таким образом данные с соответствующим статусом не видны пользователям, но в тоже время остаются сохраненными в базе данных и могут быть восставлены, или использованы для аналитических целей. Другими, одинаковыми для всех таблиц, полями, являются поля Created и Updated. Они хранят в себе время создания и время последнего изменения конкретной строки таблицы соответственно.

Таблица Owners содержит в себе данные о владельцах сети магазинов и имеет следующие поля:

* Id: bigint – уникальный идентификатор владельца;
* Created: timestamp – время создания записи о владельце;
* Updated: timestamp – время последнего изменения записи о владельце;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* FirstName: varchar(255) – имя владельца;
* LastName: varchar(255) – фамилия владельца;
* UserName: varchar(255) – никнейм владельца;
* Email: varchar(255) – почта владельца;
* PasswordHash: varchar(255) – хэш пароля, используемый при аутентификации владельца.

Таблица Shops содержит в себе данные о цветочных магазинах. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор владельца;
* Created: timestamp – время создания записи о магазине;
* Updated: timestamp – время последнего изменения записи о магазине;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Name: varchar(255) – название магазина;
* Address: varchar(255) – адрес магазина;
* Contacts: varchar(255) – контактная информация магазина;
* OwnerId: bigint – идентификатор владельца, которому принадлежит магазин.

Таблица Storages содержит в себе данные о складах магазинов. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор склада;
* Created: timestamp – время создания записи о складе;
* Updated: timestamp – время последнего изменения записи о складе;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Name: varchar(255) – название склада;
* Address: varchar(255) – адрес склада;
* OwnerId: bigint – идентификатор владельца, которому принадлежит склад.

Таблица Terminals содержит в себе данные о терминалах магазинов. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор терминала;
* Created: timestamp – время создания записи о терминале;
* Updated: timestamp – время последнего изменения записи о терминале;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Login: varchar(255) – логин терминала;
* PasswordHash: varchar(255) – хэш пароля, используемый при входе в терминал;
* ShopId: bigint – идентификатор магазина, которому принадлежит терминал.

Таблица Categories содержит в себе данные о категориях товаров. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор категории товара;
* Created: timestamp – время создания записи о категории товара;
* Updated: timestamp – время последнего изменения записи о категории товара;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Name: varchar(255) – наименование категории товара;
* ShopId: bigint – идентификатор магазина, в котором данная категория товара была создана.

Таблица Products содержит в себе данные о товарах. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор товара;
* Created: timestamp – время создания записи о товаре;
* Updated: timestamp – время последнего изменения записи о товаре;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Name: varchar(255) – наименование товара;
* Price: bigint – стоимость товара;
* CategoryId: bigint – идентификатор категории, которой принадлежит товар;
* ShopId: bigint – идентификатор магазина, в котором данный товар был создан.

Таблица Florists содержит в себе данные о флористах. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор флориста;
* Created: timestamp – время создания записи о флористе;
* Updated: timestamp – время последнего изменения записи о флористе;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* FirstName: varchar(255) – имя флориста;
* LastName: varchar(255) – фамилия флориста;
* UserName: varchar(255) – никнейм флориста;
* Password: bigint – пароль флориста для входа в терминале.

Таблица Checks содержит в себе данные о чеках. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор чека;
* Created: timestamp – время создания записи о чеке;
* Updated: timestamp – время последнего изменения записи о чеке;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Price: bigint – стоимость чека.

Таблица Stocks содержит в себе данные о складских накоплениях. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор складских накоплений;
* Created: timestamp – время создания записи о складских накоплениях;
* Updated: timestamp – время последнего изменения записи о складских накоплениях;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Count: bigint – количество содержащегося товара;
* ProductId: bigint – идентификатор товара, который представлен на складе в виде накоплений;
* StorageId: bigint – идентификатор склада, в котором содержатся данные накопления.

Таблица Purchases содержит в себе данные о покупках заказчиков. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор покупки;
* Created: timestamp – время создания записи о покупке;
* Updated: timestamp – время последнего изменения записи о покупке;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Count: bigint – количество содержащегося товара;
* ProductId: bigint – идентификатор товара, который был куплен заказчиком;
* CheckId: bigint – идентификатор чека, в которую входит данная покупка.

Таблица Searches содержит в себе данные о методах поиска заказов. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор метода поиска;
* Created: timestamp – время создания записи о методе поиска;
* Updated: timestamp – время последнего изменения записи о методе поиска;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Title: varchar(255) – наименование метода поиска заказа.
* Value: varchar(255)- URL-запрос, необходимый для поиска по конкретному параметру.

Таблица Sorts содержит в себе данные о способах сортировок заказов. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор метода сортировки;
* Created: timestamp – время создания записи о методе сортировки;
* Updated: timestamp – время последнего изменения записи о сортировке;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Title: varchar(255) – наименование метода сортировки заказа.
* Value: varchar(255)- URL-запрос, необходимый для сортировки по конкретному условию.

Таблица Orders содержит в себе данные о заказах. Данная таблица имеет следующую структуру:

* Id: bigint – уникальный идентификатор заказа;
* Created: timestamp – время создания записи о заказе;
* Updated: timestamp – время последнего изменения записи о заказе;
* Status: varchar(25) – статус, указывающий на актуальность записи;
* Address: varchar(255) – адрес доставки заказа;
* Date: varchar(255) – дата доставки заказа;
* Price: varchar(255) – цена заказа;
* Time: varchar(255) – время доставки заказа;
* Courier: varchar(255) – имя курьера, ответственного за доставку заказа;
* Customer: varchar(255) – имя заказчика;
* CustomerNumber: varchar(255) – номер телефона заказчика;
* Notes: varchar(1024) – примечания о заказе;
* PayStatus: boolean – флаг, указывающий совершена ли оплата заказа заказчиком;
* PaymentMethod: varchar(255) – метод оплаты заказа заказчиком;
* Receiver: varchar(255) – имя получателя;
* ReceiverNumber: varchar(255) – номер телефона получателя;
* FloristId: bigint – идентификатор флориста, обслуживающего заказ;
* CheckId: bigint – идентификатор чека, содержащего заказанный товар;
* TerminalId: bigint – идентификатор терминала, в котором данный заказ был создан и обработан.

## 3.4 Модуль работы frontend

Клиентская часть приложения реализована на JavaScript библиотеке React. Так как основным принципом библиотеки React является построение приложения с помощью компонент, то и функциональное проектирование будет строиться на создании и описании компонент.

Данное приложение было создано с помощью механизма create-react-app, а также дополнялось с помощью пакетного менеджера npm. Данный механизм создает полностью настроенное React приложение. Для учета зависимостей используется файл package.json, в котором сохраняются названия библиотек, способов запуска, настройки линтеров, используемые приложением. Все зависимости устанавливаются в папку node\_moduls. В папке public лежит html-файл, в который будет рендерится приложение. Каталог src отвечает непосредственно за само приложение со всеми компонентами, логикой.

Компонент EnterPage предназначен для отображения первой страницы приложения, которая предназначена для выбора той части приложения, с которой пользователю предстоит работать. В нем реализована форма выбора, состоящая из двух альтернатив: терминала и админ-панели.

Кнопки перехода к определенной части приложения также являются и ссылками, благодаря которым в React реализован роутинг. Основными библиотеками для этого служат react-router и react-router-dom, которые и используются в данном приложении. Роутинг в React удобен тем, что можно с легкостью менять как полностью всю страницу, так и некоторые ее части.

Так же в этом и дальнейших компонентах UI будет реализован с использованием фреймворка Material UI. Данный фреймворк предоставляет разнообразный набор кастомных компонентов различной сложности, от кнопок до датапикеров. Так же Material предоставляет возможность стилизации компонентов приложения с использованием технологии CSS in JS. Однако в этом и последующих компонентах будет использоваться такая CSS in JS библиотека как Aphrodite. Основное ее преимущество над обычным CSS является локальное пространство имен, благодаря которому конфликтующие стили не будут замещать друг друга из-за одинаково названных различных классов, такая проблема часто встречалась на больших проектах.

## 3.5 Модуль терминала

Компонент TerminalPage отвечает за авторизацию в модуле терминала. Редирект на данный компонент происходит после выбора пункта терминала из формы выбора в компоненте EnterPage, благодаря описанному react-router-dom.

Компонент включает в себя две формы, которые имеют условный рендеринг. Это значит, что при определенных условиях будут отображаться определенные ее части. Первая форма используется при авторизации в магазине. Она предназначена для обозначения того, в каком конкретно магазине будет использоваться терминал. Форма включает в себя логин и пароль магазина. Вторая форма используется при авторизации флориста в магазине. Здесь используется пароль флориста. Таким образом будет учтена информации о конкретном магазине с определенным флористом. Это удобно для учета смен флористов.

Данная форма отличается от прочих тем, что реализована она с использованием библиотеки formik, а в данном случае formik-material-ui для взаимодействия с UI фреймворком Material UI. Преимущество данной библиотеки состоит в том, что не обязательно заводить переменную, писать обработчик под каждое поле формы, достаточно в каждом элементе формы указать атрибут name, который будет содержать название переменной, formik сам понимает, как нужно обрабатывать то или иное поле ввода. Также, очень просто использовать данные значения, так как они хранятся в объекте с названием values.

Чтобы перейти к авторизации флориста необходимо успешно пройти вход в магазин, для этого на сервер нужно отправить логин и пароль. После успешной авторизации от сервера приходит ответ, позволяющий перейти к форме авторизации флориста. После отправки пароля флориста для дальнейшей работы в приложении необходимо получить от сервера access token.

Компонент OrderListPage предназначен для основной страницы в приложении терминала, а именно страницы с отображением заказов. Редирект на компонент происходит после отправки пароля флориста. Для доступа к данным компонента необходимо получить от сервера access token и все дальнейшие HTTP-запросы производить с заголовком авторизации, содержащим данный токен. При рендере компонента появляются несколько вопросов, таких как получение списка заказов и их хранение.

Отправку запроса на сервер с целью получить список заказов необходимо сделать после маутинга компонента, чтобы компонент мог сразу отобразить данные, для этого могут использоваться методы жизненного цикла или хуки. Существует два типа компонентов: функциональные и классовые, первые использую хуки, вторые – методы жизненного цикла, однако последнее время первые очень быстро набирают популярность, они и используются в приложении. В качестве метода, который будет отрабатывать после маутинга компонента может служить метод жизненного цикла ComponentDidMount, либо хук useEffect. Другими словами, это функция, которая выполнится после сборки компонента.

Непосредственно сам запрос описан с помощью библиотеки axios, которая является альтернативой fetch и основана она на промисах, асинхронных механизмах языка JavaScript.

Следующий вопрос – вопрос хранения данных на клиенте. Так как в React осуществляется однонаправленный поток данных от родителя к потомку, возникает проблема, когда нужно передать какие-то данные предку, и не обязательно ближайшему, для решения этой проблемы используется библиотека Redux, основанная на методологии Flux. Redux - это глобальное хранилище в приложении, доступ к которому имеют все компоненты, находящиеся в провайдере, который подписан на хранилище store. В Redux также существует цикл, осуществляющий работу данной библиотеки. Он основан на том, что существует диспатчер, который является тригером actions, которые имеют какой-то тип, reducers слушают данные экшены и в зависимости от их типа понимают, действия с какими данными в хранилище нужно производить. Таким образом обновляется store, хранилище, на которое подписаны компоненты, а значит обновляются и сами компоненты. Стоит отметить что данные в store иммутабельны.

Важной частью в работе Redux во взаимодействии с асинхронными http-запросами являются middlewares, или посредники между actions и reducers, которые позволяют сделать функции actions и reduсers чистыми, а также повышают удобство тестирования и обработки запросов. В данном приложении используется посредник Redux Saga, основанный на асинхронном JavaScript механизме генераторов.

Таким образом получаем следующую цепочку действий: хук useEffect срабатывает и вызывает функцию экшн, middleware перехватывает данный экшн и отправляет запрос на сервер, после получения ответа вызывается новая функция экшн, которую в свою очередь перехватывает reducer и заносит изменения в store, таким образом мы занесли данные о заказах в хранилище. Так как все компоненты подписаны на наше хранилище, мы можем без проблем забрать данные из него и отобразить на странице.

Компонент OrderListPage включает в себя два других:

* OrderList – компонент, предназначенный для отрисовки полученного от сервера списка заказов;
* OrderListSearch – компонент, предназначенный для выбора типов поиска и сортировок.

Компонент OrderList содержит функции, которые были описаны выше.

Компонент OrderListSearch основан на том, что приложение получает от сервера возможные типы поиска и сортировок, которые используются в тегах select формы, отмечающей за поиск и сортировку, для хранения параметров поиска, либо сортировки используется хук useState, который является альтернативой state в классовом компоненте. Он представляет собой переменную в связке с функцией изменения этой переменной. После заполнения необходимых параметров отправляется запрос на сервер по принципу, описанному выше. Таким образом мы получаем нужные нам элементы списка заказов, отсортированные или выбранные по какому-то параметру.

Компонент CreateCheckPage предназначен для создания чека. На страницу компонента можно попасть благодаря роутеру. Для данной страницы у сервера запрашивается данные о категориях. Впоследствии, используя id категории можно будет получить продукты определенной категории. При клике на определенные продукты, благодаря методам обработчика, которые используют хук useCallback, добавляются в корзину, в которой можно изменять их количество. Также в ней рассчитывается стоимость заказа, исходя из продукта и его количества. После создания заказа у пользователя есть возможность распечатать чек с информацией о заказе, а информация о нем путем post запроса отправится в базу данных на сервер. Стоит отметить, что заказы, созданные на странице создания чека учитываются не как заказы, а как покупка и сохраняются только для статистики.

Компонент включает в себя два компонента-потомка:

* ProductsPage – компонент, предназначенный для отображения продуктов;
* SelectedProductsPage – компонент, предназначенный для отображения корзины из выбранных продуктов.

Компонент CreateOrderPage предназначен для создания заказа. Часть его функционала схожа с компонентом CreateCheckPage. Здесь нет возможности печати чека, но появляется большое количество дополнительных полей, в которые заносится информация о заказчике, дате, месте и времени исполнения заказа. Заказы, созданные на данной странице, будут отображаться в списке заказов для последующей работы с ними флористов. Компонент включает в себя два компонента-потомка:

* CustomerInfo – компонент, предназначенный для отображения информации о заказчике;
* OrderInfo – компонент, предназначенный для отображения информации о заказе.

## 3.6 Модуль админ-панели

Компонент AdminPanelPage отвечает за авторизацию в модуле админ-панели. Редирект на данный компонент происходит после выбора пункта админ-панели из формы выбора в компоненте EnterPage. Компонент авторизации в админ-панели включает в себя форму, реализованную посредством formic с такими полями как, логин и пароль администратора. После отправки асинхронного запроса на сервер с авторизацией, приложение получает токен, который кешируется с помощью такой библиотеки как universal-cooki.

Компонент AdminPanel предназначен для отрисовки главной страницы админ-панели. Редирект на данный компонент происходит после получения токена и проверки его актуальности.

Компонент включает в себя следующие компоненты-потомки:

* AdminPanelMenu – компонент, предназначенный для отображения меню;
* AdminPanelContent – компонент, предназначенный для отображения контента, зависящего от выбранного пункта меню.

Компонент AdminPanelMenu содержит список пунктов меню. По клику на определенный пункт меню будет отображаться определенная часть админ панели.

Компонент AdminPanelContent отрисовывает информацию предопределенного вида. Содержит следующие компоненты-потомки:

* AdminPanelStatistic – компонент, предназначенный для отображения статистики продаж;
* AdminPanelFinance – компонент, предназначенный для отображения транзакций и рабочих смен флориста;
* AdminPanelMenu – компонент, предназначенный для отображения списка товаров и категорий товаров;
* AdminPanelAccess – компонент, предназначенный для управления имеющимися флористами, терминалами, заведениями;
* AdminPanelStorage – компонент, предназначенный для отображения складских поставок и остатков.

Компонент AdminPanelStatistic отвечает за страницу статистики продаж. Компонент отображается при выборе в компоненте AdminPanelMenu пункта меню Статистика. Данный компонент хранит в себе информацию о продажах, прибыли, убытках, количестве чеков и клиентов за определенный промежуток времени.

Компонент AdminPanelFinance включает в себя транзакции и рабочие смены. Содержит следующие компоненты:

* AdminPanelTransactions – компонент, предназначенный для учета данных о транзакциях. Отображается при выборе в компоненте AdminPanelMenu подпункта меню Транзакции;
* AdminPanelSessions – компонент, отображающий информацию о всех сменах флористов, их эффективности и продолжительности. Отображается при выборе в компоненте AdminPanelMenu подпункта меню Кассовые смены.

Компонент AdminPanelMenu включает в себя транзакции и рабочие смены. Компонент отображается при выборе в компоненте AdminPanelMenu пункта меню Меню. Содержит следующие компоненты:

* AdminPanelProducts – компонент, предназначенный для управления товарами;
* AdminPanelCategories – компонент, предназначенный для управления категориями товаров.

Компонент AdminPanelProducts отображается при выборе в компоненте AdminPanelMenu подпункта меню Товары. Содержит следующие компоненты:

* ProductList – компонент, представляющий список товаром со всеми данными о них;
* CreateProduct – компонент модального окна, предоставляющий возможность добавления нового товара.

Компонент AdminPanelCategories отображается при выборе в компоненте AdminPanelMenu подпункта меню Категории товаров. Содержит следующие компоненты:

* CategoriesList – компонент, представляющий список категорий товаров со всеми данными о них;
* CreateCategory – компонент модального окна, предоставляющий возможность добавления новой категории товара.

Компонент AdminPanelAccess включает в себя транзакции и рабочие смены. Компонент отображается при выборе в компоненте AdminPanelMenu пункта меню Доступ. Содержит следующие компоненты:

* WorkersPage – компонент, предназначенный для учета данных о сотрудниках;
* TerminalsPage – компонент, предназначенный для учета данных о терминалах;
* ShopsPage – компонент, предназначенный для учета данных о магазинах.

Компонент WorkersPage используется для учета, добавления, изменения и удаления сотрудников. Отображается при выборе в компоненте AdminPanelMenu подпункта меню Сотрудники. Содержит следующие компоненты:

* WorkersList – компонент, представляющий список созданных когда-либо сотрудников;
* CreateWorker – компонент модального окна, предоставляющий возможность добавления новой сотрудника.

Компонент TerminalsPage используется для учета, добавления, изменения и удаления терминалов. Отображается при выборе в компоненте AdminPanelMenu подпункта меню Терминалы. Содержит следующие компоненты:

* TerminalList – компонент, представляющий список созданных когда-либо терминалов;
* CreateTerminal – компонент модального окна, предоставляющий возможность добавления нового терминала.

Компонент ShopsPage используется для учета, добавления, изменения и удаления магазинов. Отображается при выборе в компоненте AdminPanelMenu подпункта меню Заведения. Содержит следующие компоненты:

* ShopsList – компонент, представляющий список созданных когда-либо магазинов;
* CreateShop – компонент модального окна, предоставляющий возможность добавления нового магазина.

Компонент AdminPanelStorage включает в себя складские остатки и поставки, когда-либо произведшиеся авторизованным владельцем. Компонент отображается при выборе в компоненте AdminPanelMenu пункта меню Склад. Содержит следующие компоненты:

* AdminPanelSupplies – компонент, предназначенный для отображения и добавления складских поставок;
* AdminPanelStocks – компонент, предназначенный для отображения складских остатков.

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## 4.1 Аутентификация и авторизация для админ–панели с применением JWT-токенов

Аутентификация и авторизация являются ключевыми функциями сервисной инфраструктуры для защиты от несанкционированного доступа к данным приложения. Несмотря на то, что эти два термина используются в одном контексте, они имеют принципиально разные значения.

Аутентификация используется для подтверждения личности пользователя, которое заключается в процессе проверки учетных данных. Если введенные данные совпадают с данными, сохраненными в базе данных, пользователю предоставляется доступ. В противном случае вход и дальнейшее использование сервиса блокируется.

Авторизация происходит уже после того, когда пользователем успешно произведена аутентификация. Данный процесс определяет, имеет ли пользователь, прошедший проверку, доступ к определенным ресурсам сервиса.

В Spring существует фреймворк Spring Security, который предоставляет возможность организовать аутентификацию и авторизацию, реализовав интерфейсы UserDetails, UserDetailsService, AuthenticationProvider, Authentication и установив в конфигурационном файле фильтр аутентификации. Однако, использование этого фреймворка по назначению в рамках данного проекта организовать затруднительно, так как необходимо реализовать аутентификацию для терминала и админ-панели, которые отличаются сущностями пользователей, производящими попытку входа, а использовать одновременно несколько реализаций интерфейсов Spring Security невозможно.

Поэтому функции Spring Security принято использовать для аутентификации в админ-панели, а для процесса аутентификации терминала реализовать собственный фильтр.

Алгоритм аутентификации для админ-панели реализует класс AuthAdminPanelServiceImpl. На вход алгоритма подается форма авторизации владельца, содержащая поля login и password, где login – строка, содержащая имя аккаунта владельца, password – строка, содержащая пароль аккаунта владельца. На выходе алгоритма – сгенерированный для авторизовавшегося владельца JWT-токен. Процесс аутентификаци происходит по следующему алгоритму:

Шаг 1. На сервер поступает HTTP-запрос по следующему URL: adminPanel/auth/login. В теле запроса содержится JSON объект, соответствующий форме авторизации, содержащий поля login и password.

Шаг 2. Вызывается метод authentication класса authAdminPanelService. На вход метода отправляется принятая в шаге 1 форма. Метод проверяет наличие владельца по логину и паролю, содержащимся в форме авторизации. Аутентификация админ-панели:

Optional<Owner> ownerCandidate = ownerRepository.findOneByLogin(loginForm.getOwnerLogin());

if (ownerCandidate.isPresent()) {

Owner owner = ownerCandidate.get();

if ((owner.getPassword().equals(loginForm.getOwnerPassword()))) {

String token = jwtAdminPanelTokenProvider.createToken(owner);

return AdminPanelTokenDto.from(token);

}

} throw new IllegalArgumentException("Owner not found");

Шаг 3. В случае успешного нахождения владельца в базе данных, вызывается метод createToken класса JwtAdminPanelTokenProvider. На вход метода отправляется объект найденного в базе данных владельца. На выходе метода – сгенерированный для владельца JWT-токен. Для админ-панели, в каждом токене содержится следующая информация:

* идентификатор владельца, которому токен предназначен;
* уникальный идентификатор токена;
* время создания токена;
* время истечения срока действия токена.

Создание токена для админ-панели:

Claims claims = Jwts.claims();

claims.put("owner\_id", owner.getId());

Date now = new Date();

Date validity = new Date(now.getTime() + validityInMilliseconds);

return Jwts.builder()//

.setClaims(claims)//

.setIssuedAt(now)//

.setExpiration(validity)//

.signWith(SignatureAlgorithm.HS256, secret)//

.compact();

Шаг 4. Если владелец не найден – генерируется исключение IllegalArgumentException, содержащее комментарий Owner not found. Переход на шаг 6.

Шаг 5. Если владелец найден - сгенерированный JWT-токен отправляется клиенту в виде dto-объекта AdminPanelTokenDto.

Шаг 6. Конец алгоритма.

Алгоритм авторизации для админ-панели реализует класс AuthAdminPanelServiceImpl. На вход алгоритма подается сгенерированный ранее JWT-токен. На выходе алгоритма – объект класса TokenAuthentication, содержащий данные об авторизации владельца. Процесс авторизации происходит по следующему алгоритму:

Шаг 1. На сервер поступает HTTP-запрос в URL которого содержится adminPanel. В заголовке AuthAdmin запроса содержится информация о JWT-токене.

Шаг 2. Вызывается метод doFilter класса AuthAdminFilter. Извлекается значение заголовка AuthAdmin.

Шаг 3. Вызывается метод resolveToken класса JwtAdminPanelTokenProvider. На вход метода отправляется извлеченное в шаге 2 значение заголовка. На выходе – исходный JWT-токен. Получение исходного токена:

if (bearerToken != null && bearerToken.startsWith("Bearer ")) {

log.info("IN resolveToken - token: {}", bearerToken.substring(7));

return bearerToken.substring(7);

} throw new IllegalArgumentException("Token not found");

Шаг 4. Вызывается метод validateToken класса JwtAdminPanelTokenProvider. На вход метода отправляется полученный в шаге 3 токен. Метод производит валидацию токена. Значение должно быть не пустым. Валидация токена:

try {

Jws<Claims> claims = Jwts.parser().setSigningKey(secret).parseClaimsJws(token);

if (claims.getBody().getExpiration().before(new Date())) {

throw new JwtValidationException("Token expired");

}

return true;

} catch (JwtException | IllegalArgumentException e) {

throw new JwtAuthenticationException("JWT token is expired or invalid");

}

Шаг 5. Если токен в результате расшифровки не валиден, генерируется исключение JwtAuthenticationException, содержащее комментарий JWT token is invalid. Переход на шаг 9.

Шаг 6. Если в результате расшифровки у токена истекло время жизни, генерируется исключение JwtValidationException, содержащее комментарий JWT token is expired. Переход на шаг 9.

Шаг 7. Если в результате расшифровки токен валиден. Создается новый экземпляр класса TokenAuthentication, в конструктор которого передается расшифрованный токен.

Шаг 8. Вызывается метод authenticate класса TokenAuthenticationProvider. В методе для объекта класса TokenAuthentication в поле userDetails устанавливается объект авторизовавшегося владельца и поле isAuthenticated устанавливается в значение true.

Шаг 9. Конец алгоритма.

## 4.2 Аутентификация и авторизация для терминала с применением JWT-токенов

Алгоритм аутентификации для терминала реализует класс AuthTerminalServiceImpl. На вход алгоритма подается форма авторизации терминала, содержащая поля terminalLogin, terminalPassword и floristPassword, где terminalLogin – строка, содержащая имя терминала, terminalPassword – строка, содержащая пароль терминала, floristPassword – строка, содержащая пароль флориста. На выходе алгоритма – сгенерированный для авторизовавшегося флориста JWT-токен. Процесс аутентификаци происходит по следующему алгоритму:

Шаг 1. На сервер поступает HTTP-запрос по следующему URL: terminal/auth/preLogin. В теле запроса содержится JSON объект, соответствующий форме авторизации терминала, содержащей поля login и password.

Шаг 2. Вызывается метод preAthentication класса authTerminalService. На вход метода отправляется принятая в шаге 1 форма. Метод проверяет наличие терминала по логину и паролю, содержащимся в форме авторизации. Аутентификация терминала:

Optional<Terminal> terminalCandidate = terminalRepository.findOneByLogin(terminalForm.getLogin());

if (terminalCandidate.isPresent()) {

Terminal terminal = terminalCandidate.get();

return new TerminalLoginDto(terminal.getLogin(), terminal.getPassword());

} throw new IllegalArgumentException("Terminal not found");

Шаг 3. В случае успешного нахождения владельца в базе данных возвращается dto-объект класса TerminalLoginDto.

Шаг 4. Если владелец не найден – генерируется исключение IllegalArgumentException, содержащее комментарий Terminal not found. Переход на шаг 10.

Шаг 5. На сервер поступает HTTP-запрос по следующему URL: terminal/auth/login. В теле запроса содержится JSON объект, соответствующий форме авторизации флориста, содержащаей поля terminalLogin, terminalPassword и floristPassword.

Шаг 6. Вызывается метод authentication класса authATerminalService. На вход метода отправляется принятая в шаге 5 форма. Метод проверяет наличие флориста по паролю и терминала по логину и паролю, содержащимся в форме авторизации. Аутентификация флориста:

Optional<Terminal> terminalCandidate = terminalRepository.findOneByLogin(loginForm.getTerminalLogin());

Optional<Florist> floristCandidate = floristRepository.findOneByPassword(loginForm.getFloristPassword());

if (terminalCandidate.isPresent() && floristCandidate.isPresent()) {

Terminal terminal = terminalCandidate.get();

Florist florist = floristCandidate.get();

if ((terminal.getPassword().equals(loginForm.getTerminalPassword()))) {

String token = jwtTerminalTokenProvider.createToken(florist, terminal);

return TerminalTokenDto.from(token);

}

} throw new IllegalArgumentException("Terminal or Florist not found");

Шаг 7. В случае успешного нахождения владельца в базе данных, вызывается метод createToken класса JwtTerminalProvider. На вход метода отправляется объекты найденных в базе данных терминала и флориста. На выходе метода – сгенерированный для флориста JWT-токен. Для терминала, в каждом токене содержится следующая информация:

* идентификатор флориста, которому токен предназначен;
* идентификатор терминала, в который была произведена аутентификация;
* уникальный идентификатор токена;
* время создания токена;
* время истечения срока действия токена.

Создание токена для терминала:

Claims claims = Jwts.claims();

claims.put("florist\_id", florist.getId());

claims.put("terminal\_id", terminal.getId());

Date now = new Date();

Date validity = new Date(now.getTime() + validityInMilliseconds);

return Jwts.builder()//

.setClaims(claims)//

.setIssuedAt(now)//

.setExpiration(validity)//

.signWith(SignatureAlgorithm.HS256, secret)//

.compact();

Шаг 8. Если терминал или флорист не найдены – генерируется исключение IllegalArgumentException, содержащее комментарий Terminal or Florist not found. Переход на шаг 10.

Шаг 9. Если терминал и флорист найдены - сгенерированный JWT-токен отправляется клиенту в виде dto-объекта TerminalTokenDto.

Шаг 10. Конец алгоритма.

Алгоритм авторизации для админ-панели реализует класс TerminalPanelServiceImpl. На вход алгоритма подается сгенерированный ранее JWT-токен. На выходе алгоритма - объект класса JWS<Claims>, содержащий расшифрованные данные, содержащиеся в JWS-токене. Процесс авторизации происходит по следующему алгоритму:

Шаг 1. На сервер поступает HTTP-запрос в URL которого содержится terminal. В заголовке AuthTerm запроса содержится информация о JWT-токене.

Шаг 2. Вызывается метод authorization класса AuthTerminalServiceImpl. Авторизация термиала:

String token = jwtTerminalTokenProvider.resolveToken(bearerToken);

jwtTerminalTokenProvider.validateToken(token);

Jws<Claims> claims = jwtTerminalTokenProvider.getClaims(token);

return claims;

Шаг 3. Вызывается метод resolveToken класса JwtTerminalTokenProvider. На вход метода отправляется извлеченное в шаге 2 значение заголовка. На выходе – исходный JWT-токен.

Шаг 4. Вызывается метод validateToken класса JwtTerminalTokenProvider. На вход метода отправляется полученный в шаге 3 токен. Метод производит валидацию токена. Значение должно быть не пустым.

Шаг 5. Если токен в результате расшифровки не валиден, генерируется исключение JwtAuthenticationException, содержащее комментарий JWT token is invalid. Переход на шаг 8.

Шаг 6. Если в результате расшифровки у токена истекло время жизни, генерируется исключение JwtValidationException, содержащее комментарий JWT token is expired. Переход на шаг 8.

Шаг 7. Если в результате расшифровки токен валиден, вызывается метод getClaims класса JwtTerminalTokenProvider. Получение Jws<Claims>:

Jws<Claims> claims = Jwts.parser().setSigningKey(secret).parseClaimsJws(token);

return claims;

Шаг 8. Конец алгоритма.

## 4.3 Алгоритм работы флориста в терминале

В данном алгоритме описывается поведение приложения при работе флориста в терминале. Данный алгоритм представлен на чертеже схемы программы ГУИР.400201.023 ПД.1.

Алгоритм работы флориста реализует компонент TerminalPage. На входе алгоритма запрос доступа к терминалу. На выходе алгоритма – объект WorkShiftForm, содержащий информацию о рабочей смене флориста. Процесс работы происходит по следующему алгоритму:

Шаг 1. Проверка наличия заголовка AuthTerm. При наличии заголовка переход на шаг 10.

Шаг 2. Редирект на страницу авторизации терминала. Заполнение данных о логине и пароле терминала.

Шаг 3. Отправка данных formic запросом, описанным в виде саги, на сервер для авторизации. Запрос в виде саги:

function onLoginShop (data) {

data = {

login: data.terminalLogin,

password: data.terminalPassword,

}

return axios

.post('https://flowers-manager-api.herokuapp.com/api/terminal/auth/preLogin'

.then(

response => {

return response.data

}

)

}

Шаг 4. Если ответ от сервера 500 (Internal Server Error), переход на шаг 2.

Шаг 5. Если ответ от сервера 500 (Ok), занесение тела ответа в store.

Шаг 6. Переход к авторизации флориста. Заполнение данных о пароле флориста. Нажатие на кнопку входа.

Шаг 7. Выполнение запроса api/termina/auth/login.

Шаг 8. Если ответ от сервера 500 (Internal Server Error), переход на шаг 2.

Шаг 9. Принятие от сервера ответа в виде JWT-токена. Занесение токена в хранилище в виде куки с использованием библиотеки universa-cookie и в хидер каждого запроса посредством библиотеки axios:

cookies.set('AuthTerm', `Bearer ${response.data.token}`)

axios.defaults.headers['AuthTerm'] = cookies.get('AuthTerm')

Шаг 10. Если доступ не получен, переход на шаг 2.

Шаг 11. Если доступ получен, редирект на главную страницу терминала.

Шаг 12. Регистрация на сервере рабочей смены авторизовавшегося флориста.

Шаг 13. Получение данных о заказе, видах поиска и сортировок, используя технологию хуков, срабатывающих после маутинга компонента OrderListPage. Хук useEffect:

useEffect(() => {

getOrders()

getSorts()

getSearch()

}, [])

Шаг 14. Занесение списка заказов в хранилище с помощью reducers, которые перезаписывают нужные поля хранилища в зависимости от тип приходящего экшена. Пример reducers:

export default function orders (state = false, action) {

switch (action.type) {

case 'ON\_GET\_ORDERS': return action.payload

default: return state

}

}

Шаг 15. Получение компонентом данных из хранилища, используя компонент высшего порядка connect, который соединяет контейнер, получающий данные и компонент.

Шаг 16. Отрисовка полученных данных на странице.

Шаг 17. Нажатие кнопки меню создания заказа.

Шаг 18. Редирект на страницу создания заказа.

Шаг 19. Если заказ долгосрочный, заполнение информации о заказчике для возможности доставки заказа.

Шаг 20. Редирект на страницу создания чека.

Шаг 21. Получение информации о категориях товаров.

Шаг 22. Выбор определенной категории товара.

Шаг 23. Вызов функции обработчика, запрашивающего товары по id категории:

const getProductsData = useCallback(() => {

getProducts(category.id)

})

Шаг 24. Получение товаров, соответствующих выбранной категории. Сага получения товаров:

function onGetProductsData (id) {

return axios

.get(`https://flowers-manager-api.herokuapp.com/api/terminal/getProducts?value=${id}`, {

headers: { AuthTerm: cookies.get('AuthTerm') },

})

.then(

response => {

return response.data

}

)

}

Шаг 25. Выбор определенного товара.

Шаг 26. Занесение выбранного товара в корзину.

Шаг 27. Редактирование необходимого количества выбранного товара.

Шаг 28. Обновление суммы чека.

Шаг 29. Повторение шагов 21-28 до полного формирования чека.

Шаг 30. Нажатие на кнопку Готово.

Шаг 31. Отправление сформированного чека на сервер.

Шаг 32. Сохранение чека для ведения статистики.

Шаг 33. Если заказ долгосрочный, добавление чека в список заказов.

Шаг 34. Если заказ не долгосрочный – печать чека для потребителя.

Шаг 35. Переадресация на страницу со списком заказов, уже имеющим данные о новом заказе.

Шаг 36. Нажатие кнопки транзакции.

Шаг 37. Заполнение формы транзакции.

Шаг 38. Отправка на сервер запроса на формирование транзакции.

Шаг 39. Если транзакция вида расход – вычитание введенной суммы из счета терминала.

Шаг 40. Если транзакция вида приход – добавление введенной суммы в счет терминала.

Шаг 41. Если транзакция вида инкассация – вычитание введенной суммы из счета терминала в счет магазина.

Шаг 42. Сохранение транзакции в базе данных, сохранение измененного состояния счета.

Шаг 43. Повторение шагов 13-42 до окончания рабочей смены флориста.

Шаг 44. Нажатие кнопки выхода из терминала.

Шаг 45. Отправка на сервер запроса на окончание рабочей смены флориста.

Шаг 46. Закрытие зарегистрированной при входе рабочей смены флориста. Сохранение измененных за время рабочей смены данных в базе данных.

Шаг 47. Переадресация на главную страницу авторизации.

Шаг 48. Конец алгоритма.

# 5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование в процессе разработки программного обеспечения является крайне важной частью цикла, которая зачастую влияет на качество и надежность финальной версии приложения. Чем больше становится приложение, тем большее значение имеют правильно построенные тесты. Также, при смене в команде разработчика, ввиду его недостаточной осведомленности в проекте, возможны такие ситуации, когда разработчик, выполняя устранение проблемы в одном месте программы, может создать большое количество ошибок во многих других.

Функциональное тестирование – вид тестирования, при котором производится проверка программного обеспечения на требования, которые были заявлены в спецификации. Данный вид тестирования является основным в разрабатываемом приложении. У функционального тестирования существует два подхода: тестирование «белого ящика» и тестирование «черного ящика». В тестировании данного приложения использовался подход – тестирование «белого ящика». Данный подход предполагает, что внутренняя структура системы известны тестировщику.

В рамках разрабатываемого проекта тестирование производится по тест-кейсам. Тест-кейс – проверка какой-либо единичной функциональности. Они описывают последовательность действий, направленных на проверку какого-либо функционала, ожидая некий фактический результат.

Функциональное тестирование проекта можно разделить на проверку трех блоков приложения:

* пользовательского интерфейса;
* выполнения CRUD-операций;
* обработка HTTP-запросов.

Вся проверка заключается в формировании тест-кейсов для всех трех блоков вручную. Тестовый набор представлен на таблице 5.1.

Таблица 5.1 – Тестирование системы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя теста | Описание | Шаги | Ожидаемый результат | Статус |
| 1 | 2 | 3 | 4 | 5 |
| Т1 | Успешная авторизация в магазине. | Переход на страницу авторизации в терминале, ввод существующих данных в форму входа в магазин, удовлетворяющих валидации, нажатие кнопки Войти в магазин. | Ответ от сервера с кодом 200 (Ok) и подтверждением авторизации в магазине. | ОК |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Т2 | Ошибка авторизации в магазине. | Переход на страницу авторизации в терминале, ввод не существующих данных в форму входа в магазин, удовлетворяющих валидации, нажатие кнопки Войти в магазин. | Ответ от сервера с кодом 500 (Internal Server Error) и сообщение о неверных данных в полях авторизации в магазин. | ОК |
| Т3 | Успешная авторизация флориста в терминале. | Переход на форму авторизации флориста в терминале, ввод существующих данных в форму входа в магазин, нажатие кнопки Войти в терминал. | Ответ от сервера с кодом 200 (Ok) и подтверждением авторизации в терминале, редирект на главную страницу заказов терминала. | ОК |
| Т4 | Ошибка авторизации флориста в терминале. | Переход на форму авторизации флориста в терминале, ввод не существующих данных в форму входа в магазин, нажатие кнопки Войти в терминал. | Ответ от сервера с кодом 500 (Internal Server Error) и сообщение о неверных полях авторизации в магазин. | ОК |
| Т5 | Успешная сортировка заказов. | Переход на страницу со списком заказов, выбор нужной сортировки в форме выбора, нажатие на кнопку Отсортировать. | Ответ от сервера с кодом 200 (Ok) и отсортированным списком заказов, обновление списка на странице. | ОК |
| Т6 | Успешный поиск заказа. | Переход на страницу со списком заказов, выбор нужного поиска в форме поиска заказов, заполнение параметров заказа, нажатие на кнопку Поиск. | Ответ от сервера с кодом 200 (Ok) и заказом, удовлетворяющим условиям поиска, обновление списка на странице. | ОК |
| Т7 | Ошибка поиска заказа. | Переход на страницу со списком заказов, выбор нужного поиска в форме поиска заказов, оставление параметров заказа незаполненным, нажатие на кнопку Поиск. | Ответ от сервера с кодом 500 (Internal Server Error) и сообщением о незаполненных параметрах поиска. | ОК |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Т8 | Успешное создание чека. | Переход на страницу создания чека, выбор подходящей категории и занесение продуктов этой категории в корзину, нажатие на кнопку Печать чека. | Печать на принтере чека, отображающего продукты в заказе, их ценник, перенаправление на страницу списка заказов. | ОК |
| Т9 | Ошибка создания чека. | Переход на страницу создания чека, выбор подходящей категории и не занесение продуктов этой категории в корзину, нажатие на кнопку Печать чека. | Сообщение о пустой корзине в виде модального окна. | ОК |
| Т10 | Успешное удаление заказа. | Переход на страницу заказов, нажатие кнопки удаления на определенном заказе в списке, подтверждение удаления в появившемся модальном окне. | Ответ от сервера со статусом 200 (Ok) и обновленным списком заказов | ОК |
| Т11 | Успешная авторизация в админ-панели. | Переход на страницу авторизации в админ-панели, заполнение полей существующими данными, удовлетворяющих валидации, нажатие кнопки Войти в админ-панель. | Ответ от сервера со статусом 200 (Ok), получение токена, необходимого для работы в приложении. | ОК |
| Т12 | Ошибка авторизации админ-панели. | Переход на страницу авторизации в админ-панели, заполнение полей не существующими данными, или данными не удовлетворяющими валидации, нажатие кнопки Войти в админ-панель. | Ответ от сервера со статусом 500 (Internal Server Error) и сообщение о неверных параметрах запроса | ОК |
| Т13 | Ошибка авторизации админ-панели | Переход на страницу авторизации в админ-панели, попытка отправить пустые поля формы. | Ответ от сервера со статусом 500 (Internal Server Error) и сообщение о неверных параметрах запроса | ОК |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| T14 | Успешное получение продуктов по категории. | Переход на странице создания чека или создания заказа в раздел информации о заказе, клик по компоненту категории. | Ответ от сервера со статусом 200 (Ok) и список продуктов выбранной категории. | ОК |
| Т15 | Успешное получение списка заказов после маутинга страницы заказов. | Переход на страницу создания заказов и отправка запроса на получение списка заказов. | Ответ от сервера со статусом 200 (Ok) и список заказов. | ОК |
| Т16 | Успешное редактирование заказа. | Переход на страницу списка заказов, нажатие на кнопку редактирования определенного заказа, изменение нужных полей в открывшемся модальном окне, нажатие кнопки Изменить после завершение редактирования. | Ответ от сервера со статусом 200 (Ok) и список заказов с включающий в себя последние изменения. | ОК |
| Т17 | Ошибка редактирования заказа. | Переход на страницу списка заказов, нажатие на кнопку редактирования определенного заказа, оставление пустых обязательных полей в открывшемся модальном окне, нажатие кнопки Изменить после завершение редактирования. | Ответ от сервера со статусом 500 (Internal Server Error) и сообщением о незаполненных обязательных полях | ОК |
| Т18 | Успешный просмотр заказа. | Переход на страницу списка заказов, нажатие на кнопку показа полной информации о заказе. | Открытие модального окна, содержащего полную информацию о конкретном заказе. | ОК |
| Т19 | Успешное добавление сотрудника | Переход на вкладку сотрудников в админ-панели, нажатие на кнопку создания сотрудника, заполнение модального окна. | Ответ от сервера со статусом 201 (Created) и списком всех сотрудников, включая последнего добавленного. | ОК |

Окончание таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Т20 | Успешное добавление товара. | Переход на вкладку товаров в админ-панели, клик на кнопку создания товара, заполнение модального окна, нажатие на кнопку Создать. | Ответ от сервера со статусом 201 (Created) и списком всех товаров, включая последнего добавленного. | ОК |
| Т21 | Успешное добавление категории. | Переход на вкладку категории в админ-панели, клик на кнопку создания категории, заполнение модального окна, нажатие на кнопку Создать. | Ответ от сервера со статусом 200 (Ok) и списком всех категорий, включая последнюю добавленную. | ОК |

Как видно из проделанных тест-кейсов, все тесты достигли ожидаемого результата, что говорит о том, что реализованный программный продукт работоспособен и отвечает требованиям, поставленным в спецификации.

# 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Клиентские приложения - веб-приложения, поэтому они не имеют жесткой привязанности к операционной системе, однако рассмотрим некоторые важные системные требования:

* операционная система: Microsoft Windows XP и выше, Apple Mac OS X, Linux;
* веб-браузер: Safari, Safari mobile, Chrome, Chrome mobile, Mozilla Firefox, Mozilla Firefox mobile, Internet Explorer 11+;
* рекомендуемая оперативная память устройства: 512 мегабайт и выше.

## 6.1 Руководство пользователя для терминала

Для того, чтобы войти в терминал, необходимо пройти 2 стадии авторизации: вход в магазин и авторизация флориста. Форма входа в магазин представлена на рисунке 6.1.

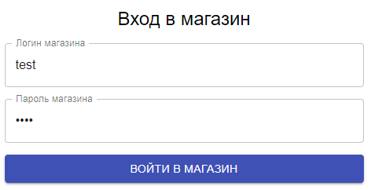


Рисунок 6.1 – Форма входа в магазин

В случае успешного входа в магазин, следующим этапом авторизации является ввод пароля флориста, для идентификации работника, который будет производить обработку заказов в терминале. Форма авторизации в терминале представлена на рисунке 6.2.

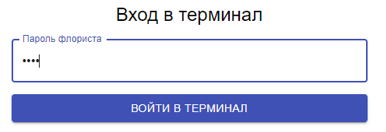


Рисунок 6.2 – Форма авторизации в терминале

После авторизации пользователь, он же флорист, попадает на страницу со списком заказов, которая является основной в терминале. Страница заказов представлена на рисунке 6.3.

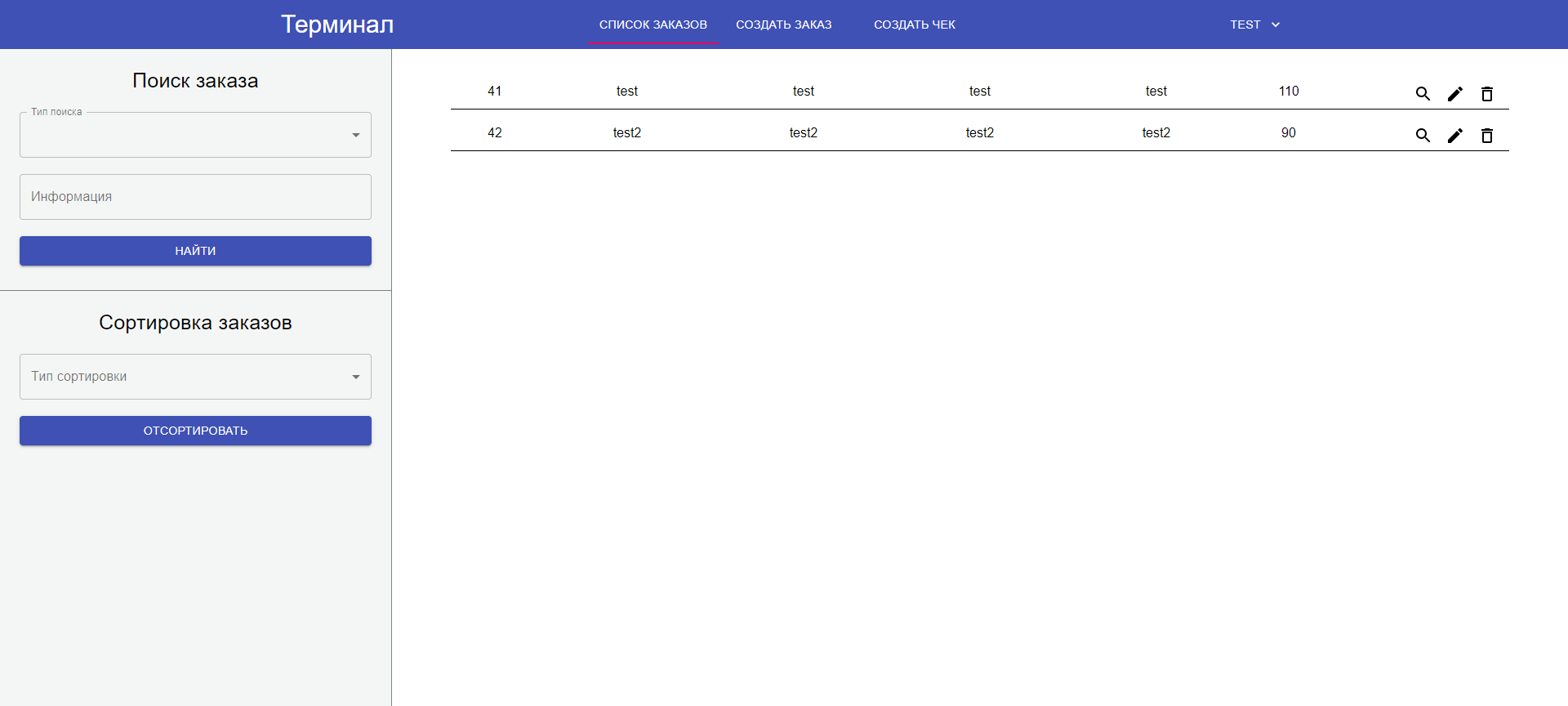


Рисунок 6.3 – Страница заказов

Данная страница включает в себя сам список заказов, а также формы, служащие для сортировок и поиска нужного заказа. Форма сортировок представлена на рисунке 6.4.

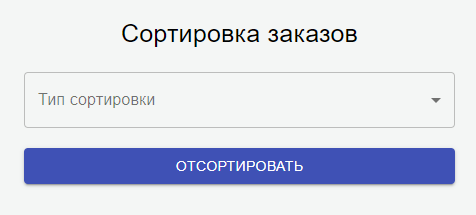


Рисунок 6.4 – Форма сортировки заказа

В данной форме выбирается тип сортировки и по нажатию на кнопку происходит сортировка заказов в списке. Форма поиска представлена на рисунке 6.5.

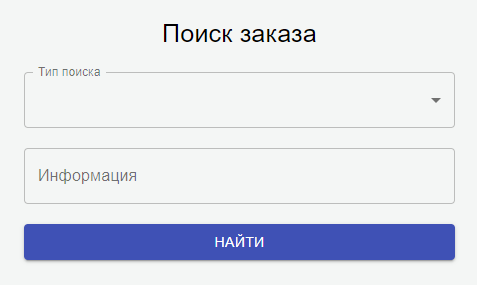


Рисунок 6.5 – Форма поиска заказов

В данной форме выбирается тип поиска и его параметр, например, поиск по цене и сама цена. По нажатию на кнопку найти происходит поиск заказа в списке.

Также, на данной странице, в списке у заказа есть набор кнопок (рисунок 6.6), имеющие возможность производить удаление, изменение и просмотр каждого заказа, по нажатию на которые открываются модальные окна с соответствующим контентом.

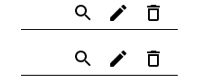


Рисунок 6.6 – Кнопки редактирования заказа

Во всем терминале, независимо от страницы, есть меню (рисунок 6.7), которое является навигацией по приложению терминала, оно находится в шапке приложения.



Рисунок 6.7 – Меню навигации терминала

По клику на пункт меню создание заказа пользователь попадает на соответствующую страницу создания заказа (рисунок 6.8).

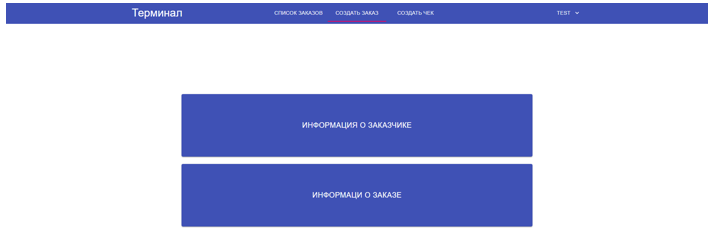


Рисунок 6.8 – Страница создания заказа

На странице создания заказа есть две кнопки, перенаправляющие на заполнение информации о заказчике (рисунок 6.9) и создание чека (рисунок 6.10).



Рисунок 6.9 – Заполнение информации о заказчике

После заполнения всех полей появится кнопка, по нажатию на которую происходит перенаправление на страницу создания чека для заказа.

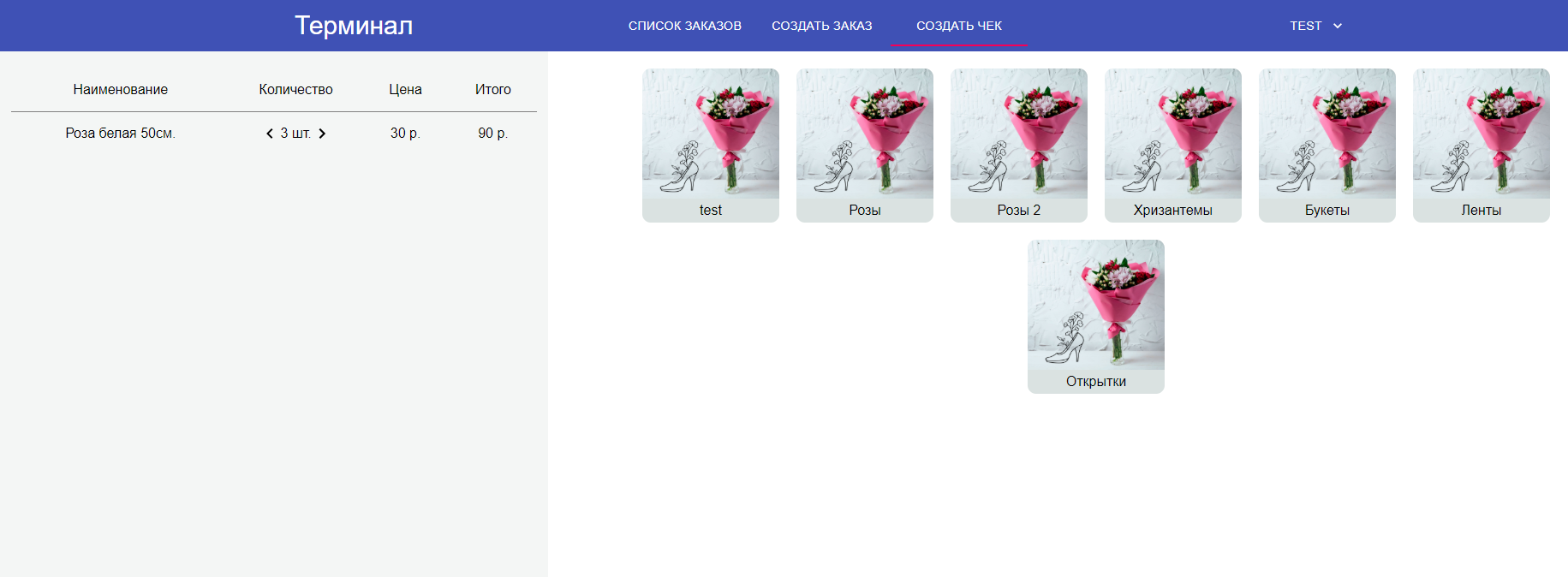


Рисунок 6.10 – Создание чека для заказа

Данная страница разделяется на 2 рабочие области: выбор товара и обработка количества товара.

Область, отвечающая за выбор товара, отображает категории, по нажатию на которые отображаются товары данной категории. По нажатии на товар, он добавляется в корзину (область, отвечающую за обработку количества продуктов в заказе). Область учета количества продукции в заказе представлена на рисунке 6.11.

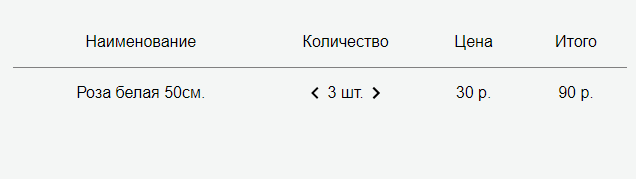


Рисунок 6.11 – Область учета количества продукции

Выбранные товары отображаются в данной области. С помощь стрелочных кнопок можно изменять количество товара. Сумма заказа рассчитывается автоматически, в зависимости от товара и его количества. По заполнению данного поля можно нажать на кнопку создания чека, что вызовет перенаправление на страницу с заказами (рисунок 6.3), в списке которых в том числе будет и только что созданный.

По клику на пункт меню создание чека, пользователь попадает сразу на страницу создания чека, без заполнения информации о заказе. Страница создания чека очень похожа на страницу при создании заказа, по внешнему виду и логике их возможности идентичны, однако добавляется дополнительная область, отвечающая за печать чека на основании выбранных товаров (рисунок 6.13). Функциональным отличием данного пункта меню является то, что данный заказ обрабатывается незамедлительно. Страница создания чека представлена на рисунке 6.12.

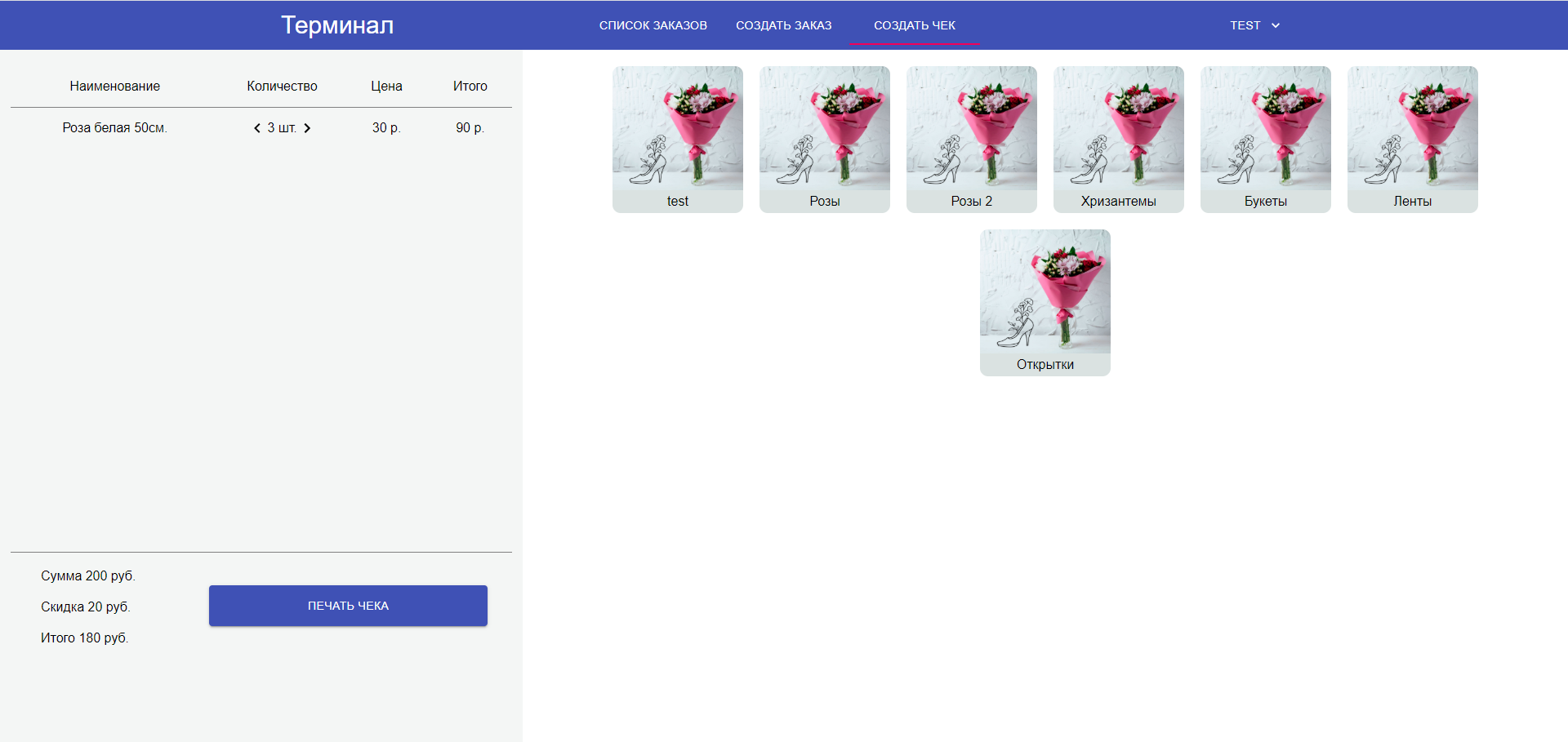


Рисунок 6.12 – Страница создания чека

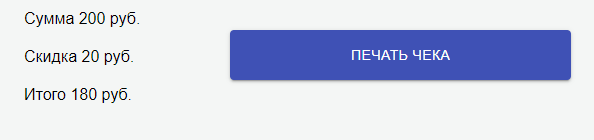


Рисунок 6.13 – Область печати чека

Весь рабочий цикл заключается в создании заказов, их редактировании, обработке. Отметив заказ как закрытый, он занесется в статистику. При входе в терминал инициализируется рабочая смена сотрудника, которая регистрирует все активности флориста. Для отслеживания статистики продаж, редактирования ассортимента товаров и учета складских накоплений предназначена админ-панель, о которой подробнее далее.

## 6.2 Руководство пользователя для админ-панели

Для входа в админ-панель, в отличие от терминала, достаточно одной стадии авторизации. Форма входа в админ-панель представлена на рисунке 6.14.

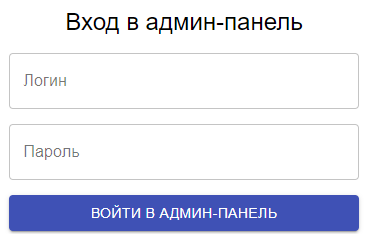


Рисунок 6.14 – Форма входа в админ-панель

После авторизации владелец попадает на страницу со статистикой продаж, которая является первой вкладкой в меню. Страница статистики продаж представлена на рисунке 6.15.

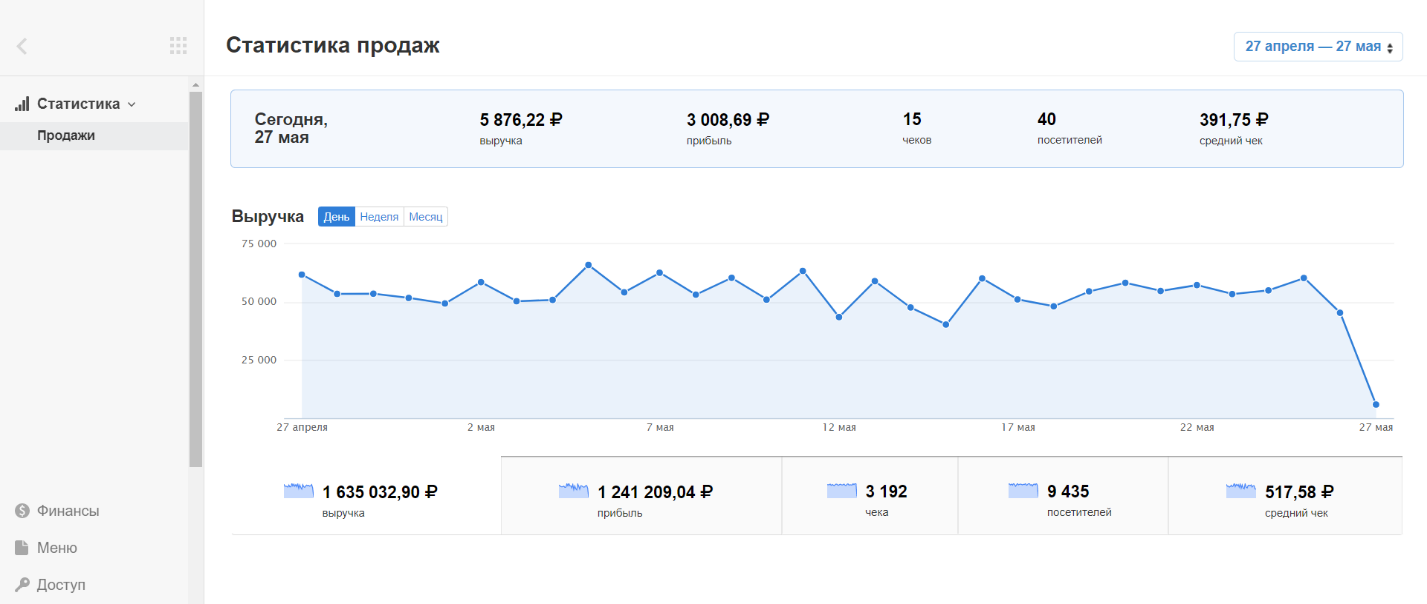


Рисунок 6.15 – Окно статистики продаж

В данном окне отображается график продаж, статистика продаж за текущий день, количество выручки, чистой прибыли, чеков, посетителей, а также средний чек. Для графика есть возможность изменения масштаба по дням, неделям, месяцам. Также, есть возможность выбора параметра для отображения на графике.

Следующим доступным владельцу пунктом меню является пункт «Финансы», имеющий подпункты «Транзакции» и «Кассовые смены», страницы которых представлены на рисунках 6.16 и 6.17 соответственно.

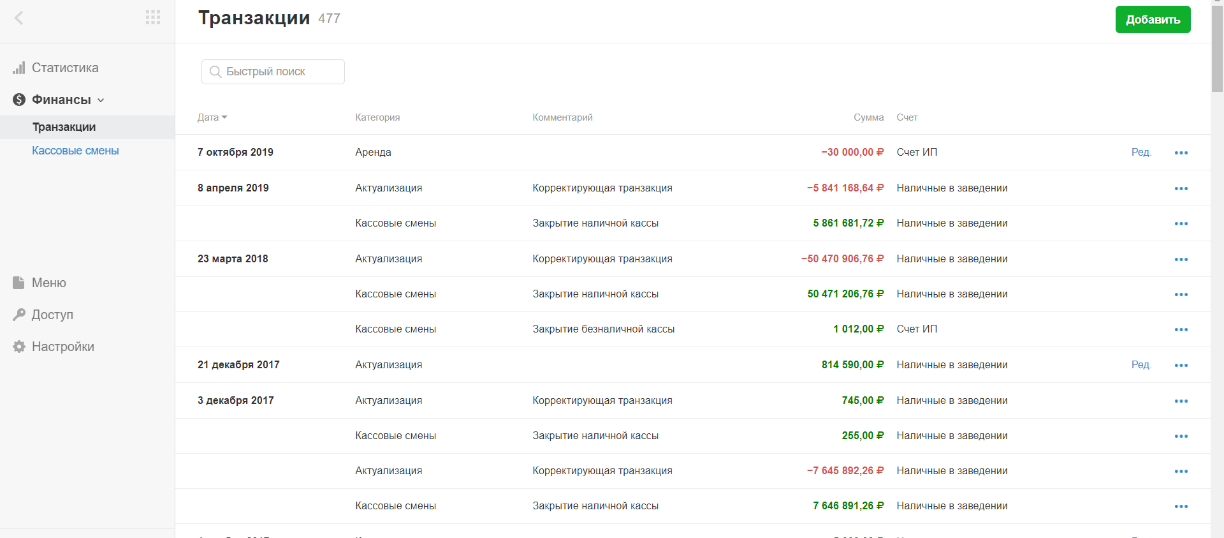


Рисунок 6.16 – Окно транзакций

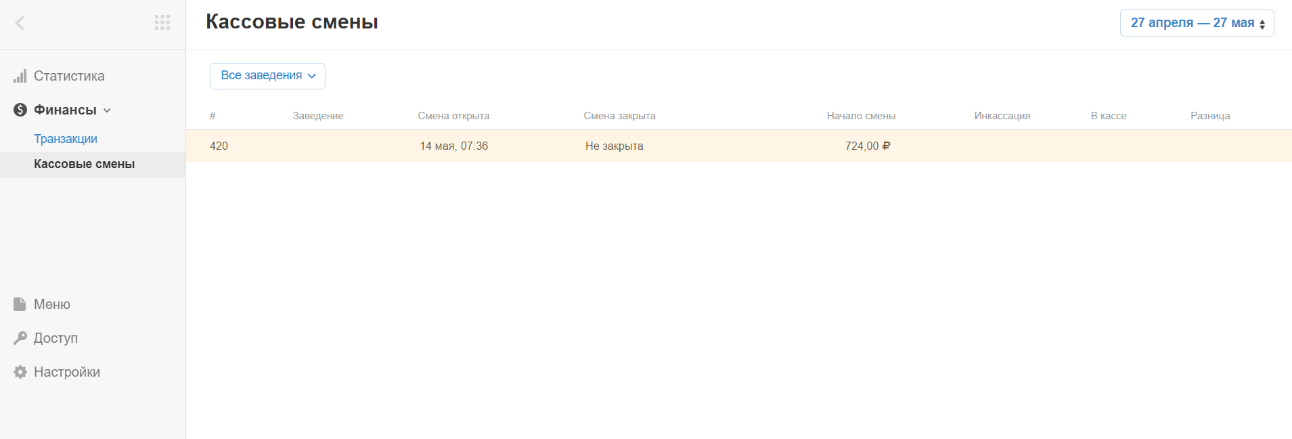


Рисунок 6.17 – Окно кассовых смен

В данном разделе меню можно наблюдать все транзакции и кассовые смены, когда-либо выполненные в магазинах, принадлежащих владельцу. Есть возможность создания новой транзакции, редактирования старой.

Раздел «Меню» состоит из двух подпунктов: «Товары» и «Категории товаров». В этом разделе можно управлять ассортиментов товаров, предоставляемых магазином. Все товары и категории товаров, созданные владельцем, будут отображаться в терминалах магазинов владельца. Также, появится возможность складских поставок конкретного товара. Окно товаров представлено на рисунке 6.18.

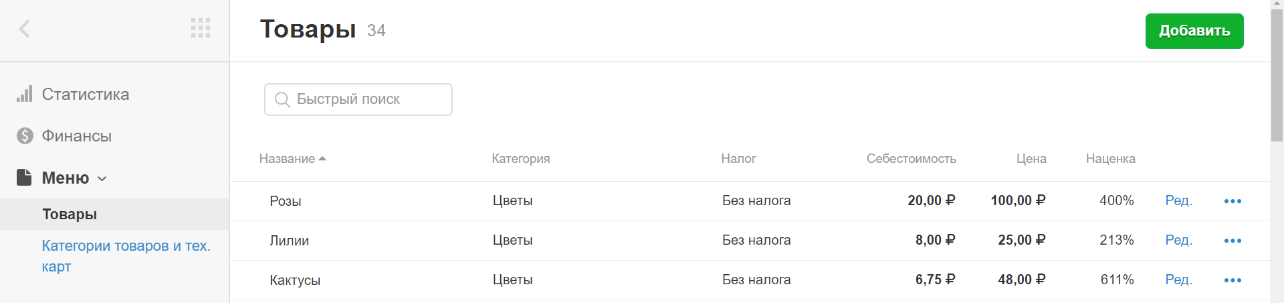


Рисунок 6.18 – Окно товаров

Окно категорий товаров представлено на рисунке 6.19:

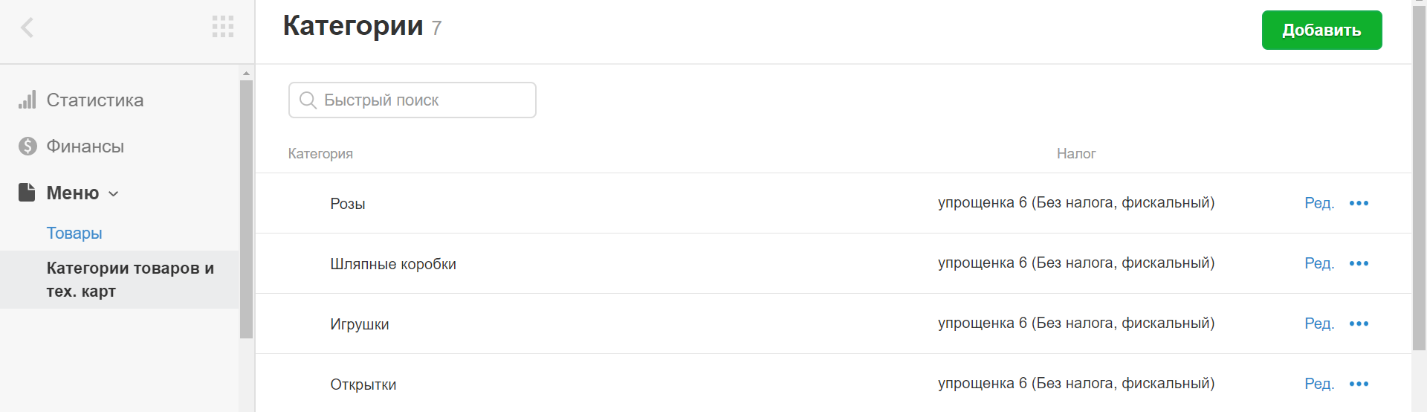


Рисунок 6.19 – Окно категорий товаров

Раздел «Доступ» состоит из трех подпунктов: «Сотрудники», «Терминалы», «Заведения». В данном разделе предоставляется возможность просмотра, создания, редактирования и удаления принадлежащих владельцу флористов, терминалов и магазинов. Окно сотрудников представлено на рисунке 6.20, окно терминалов – на рисунке 6.21, окно заведений – на рисунке 6.22.

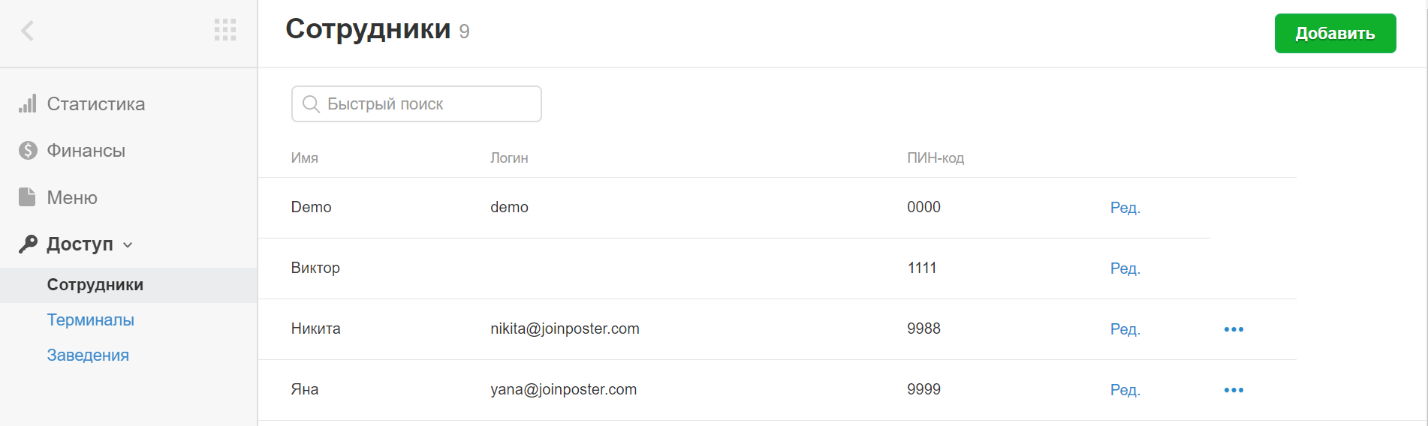


Рисунок 6.20 – Окно сотрудников

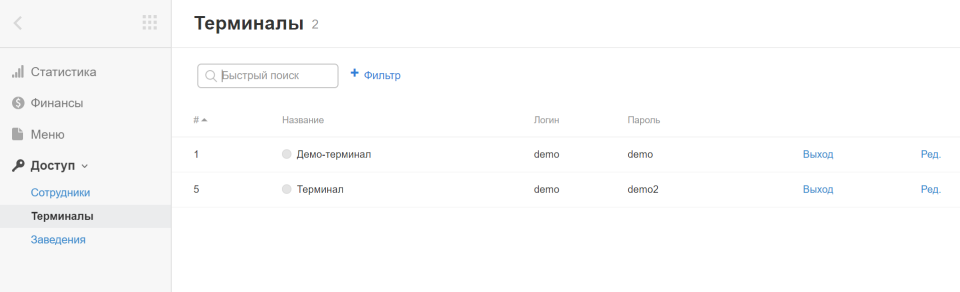


Рисунок 6.21 – Окно терминалов

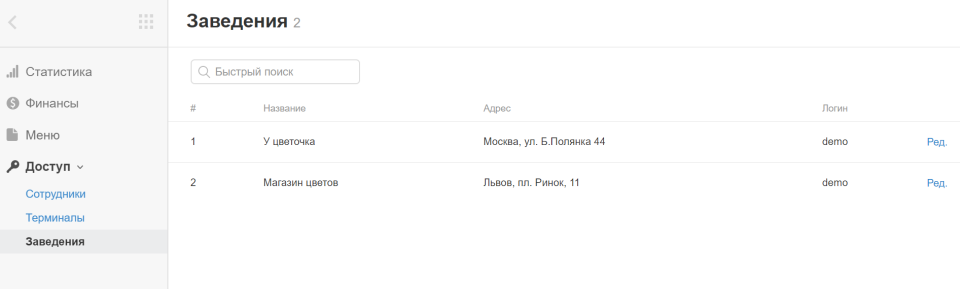


Рисунок 6.22 – Окно заведений

И последний раздел, раздел «Склад», имеющий подпункты «Остатки» и «Поставки». В данном разделе владельцу предоставляется возможность отслеживания товара на складах, добавления новых поставок товаров, их редактирования. Просмотра всех, когда-либо совершенных, поставок. Окно остатков представлено на рисунке 6.23.

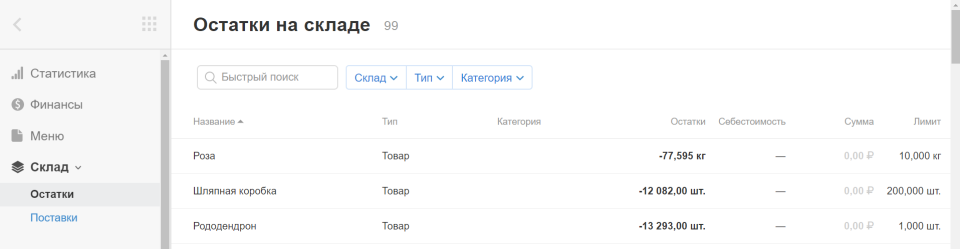


Рисунок 6.23 – Окно остатков

# 7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА ПО РАЗРАБОТКЕ СИСТЕМЫ АВТОМАТИЗАЦИИ И УЧЕТА ДЛЯ ЦВЕТОЧНЫХ МАГАЗИНОВ

## 7.1 Характеристика программного средства

В рамках данного дипломного проекта планируется разработать веб-приложение для автоматизации работы цветочного магазина, которое упростит ведение отчетности, структурирует рабочий процесс и позволит получать статистические данные о продажах.

Программное средство разрабатывается по индивидуальному заказу, предприятием УП «ИЦТ Горизонт». Пользователями приложения являются сотрудники компании-заказчика. Разработка ведется с учетом необходимости последующей расширяемости функциональной части приложения под нужды работы сотрудников компании-заказчика.

Заказчиком данного программного обеспечения является магазин цветов «Цвет Настроения», владельцем которого является индивидуальный предприниматель Пирогов Алексей Юрьевич.

После внедрения данного программного обеспечения у заказчика сократится время на обработку заказов, появится автоматически генерируемая отчетность продаж, будет легче вести складской учет.

## 7.2 Расчет затрат на разработку ПО

### 7.2.1 Затраты на основную заработную плату команды разработчиков

Затраты на основную заработную плату команд разработчиков () определяются исходя из состава и численности команды, размеров месячной заработной платы каждого из участников команды, а также общей трудоемкости разработки программного обеспечения.

Расчет величины основной заработной платы участников команды осуществляется по формуле (7.1):

|  |  |  |
| --- | --- | --- |
|  | (7.1) | (3.1) |

где *n* – количество исполнителей, занятых разработкой ПО; – коэффициент премирования (в нашем случае данное значение равно 1,5); – часовая заработная плата i-го исполнителя (руб.); – трудоемкость работ, выполняемых i-м исполнителем (ч).

Программный продукт состоит из следующих составных блоков: блок терминала, админ-панели, веб-сервиса и пользовательского интерфейса. В целях ускорения процесса разработки полный объем работ можно разделить между двумя программистами. Для руководства проектом, а также для контроля над разработкой необходим ведущий программист. Для тестирования проекта необходим один тестировщик.

Ключевыми в разработке блоками являются блоки терминала и админ-панели, требующие наибольшего количества временных затрат. При разработке одним человеком, трудоемкость модуля терминала равна 1,5 месяца, модуля админ панели – 2 месяца. Для разработки пользовательского интерфейса и модуля веб-сервиса необходимо ещё по 0,5 месяца. Среднее количество рабочих часов в месяце равно 168.

Таким образом, на основании описанных выше модулей, трудоемкость работ каждого из программистов равна 2,25 месяца или же 378 часам. Трудоемкость работ ведущего программиста равна 189 часам. Трудоемкость работ тестировщика равна двум месяцам или же 336 часам.

Данные по заработным платам команды разработчиков УП «ИЦТ Горизонт» указаны на 01.04.2020. Результаты расчета затрат на основную заработную плату команды приведены в таблице 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату команды разработчиков

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Участники команды | Вид выполняемой работы | Месячная заработная плата, руб. | Часовая заработная плата, руб. | Трудоем-кость работ, ч. | Зарплата по тарифу, руб. |
| 1 | Ведущий программист | Контроль разработки, руководство | 3000,00 | 17,86 | 189 | 3375,54 |
| 2 | Программист | Разработка | 1600,00 | 9,52 | 378 | 3598,56 |
| 3 | Программист | Разработка | 1600,00 | 9,52 | 378 | 3598,56 |
| 4 | Тестировщик | Тестирование | 900,00 | 5,36 | 336 | 1800,96 |
|  | Премия, (50%) | | | | | 6186,81 |
|  | Итого, затраты на основную заработную плату команды | | | | | 18560,43 |

Согласно результатам расчетов, затрату на основную заработную плату команды разработчиков составляют 18560,43 рублей.

### 7.2.2 Затраты на дополнительную заработную плату команды разработчиков

Дополнительная заработная плата команды по разработке ПО включает выплаты, предусмотренные законодательством о труде (оплата трудовых отпусков, льготных часов, времени выполнения государственных обязанностей и др.) и определяется по формуле (7.2):

|  |  |
| --- | --- |
|  | (7.2) |

где – затраты на основную заработную плату, (руб.); – норматив дополнительной заработной платы, равный 15 %. На основании этого имеем:

Согласно формуле (7.2), затраты на дополнительную заработную плату команды составляют 2784,06 рублей.

### 7.2.3 Отчисления в фонд социальной защиты населения и фонд обязательного страхования

Отчисления в фонд социальной защиты населения и фонд обязательного страхования определяются в соответствии с действующими законодательными актами Республики Беларусь. Расчет производится по формуле (7.3):

|  |  |
| --- | --- |
|  | (7.3) |

где – норматив отчислений в фонд социальной защиты населения и фонд обязательного страхования, % (34% – ФСЗН + 0.6% – обязательное страхование).

Таким образом, согласно формуле (7.3), размер отчислений в ФСЗН и фонд обязательного страхования составляет 7385,19 рублей.

### 7.2.4 Прочие затраты

Размер прочих затрат включается в себестоимость разработки ПО в процентах от затрат на основную заработную плату команды разработчиков (таблица 7.1) по формуле (7.4):

|  |  |
| --- | --- |
|  | (7.4) |

где – норматив прочих затрат в целом по организации (125%).

Согласно формуле (7.4), размер прочих затрат составляет 23200,54 рублей.

Полная сумма затрат на разработку программного обеспечения находится путем суммирования всех рассчитанных статей затрат и приводится в таблице 7.2.

Таблица 7.2 – Затраты на разработку программного обеспечения

|  |  |
| --- | --- |
| Статья затрат | Сумма, руб. |
| Основная заработная плата команды разработчиков | 18560,43 |
| Дополнительная заработная плата команды разработчиков | 2784,06 |
| Отчисления в фонд социальной защиты населения и фонд обязательного страхования | 7385,19 |
| Прочие затраты | 23200,54 |
| Общая сумма затрат на разработку | 51930,22 |

Полная себестоимость разрабатываемого программного средства составит 51930,22 рублей.

## 7.3 Оценка экономического эффекта от продажи ПО

### 7.3.1 Экономический эффект от продажи ПО для организации разработчика

Экономический эффект для компании-разработчика заключается в получении прибыли от реализации программного продукта. В ходе переговоров с заказчиком, а также с учетом средних рыночных цен, проект был оценен в 35000 долларов США, что составляет 90300,00 рубля, по курсу НБРБ на 01.04.2020.

Прибыль от реализации программного продукта определяется по формуле (7.5):

|  |  |
| --- | --- |
|  | (7.5) |

где Ц – цена реализации ПО заказчику (руб.); НДС – сумма налога на добавленную стоимость (руб.); – сумма расходов на разработку и реализацию (руб.).

Так как компания «ИЦТ Горизонт» не является резидентом Парка высоких технологий, то она является плательщиком НДС. Сумма налога на добавленную стоимость рассчитывается по формуле (7.6):

|  |  |
| --- | --- |
|  | (7.6) |

где – ставка налога на добавленную стоимость согласно действующему законодательству, равная 20 %.

Согласно формуле (7.6), сумма налога на добавленную стоимость составляет 15050,00 рублей.

|  |  |
| --- | --- |
|  |  |

Согласно формуле (7.5), прибыль от реализации программного продукта составляет 23319,78 рублей. Так как компания-разработчик является плательщиком налога на прибыль, её экономический эффект рассчитывается по формуле (7.7):

|  |  |
| --- | --- |
|  | (7.7) |

где – ставка налога на прибыль, равная 18 %.

Согласно формуле (7.7), размер прибыли от продажи программного продукта составляет 19122,22 рублей.

## 7.4 Расчет показателей эффективности инвестиций в разработку ПО

Исходя из того, что разработка программного продукта занимает менее года рабочего времени, можно рассчитать рентабельность инвестиций, которая характеризует доходность инвестиций, по формуле (7.8):

|  |  |
| --- | --- |
|  | (7.8) |

где – чистая прибыль, (руб.); – затраты на разработку ПО (руб.).

На основании этого вычисляем:

Согласно формуле (7.8), рентабельность инвестиций в разработку программного обеспечения составляет 36,8 %.

В результате технико-экономического обоснования разработки программного продукта данного дипломного проекта, были приведены расчеты затрат на основную заработную плату команды разработчиков, затраты на отчисления в ФСЗН и фонд обязательного страхования, а также прочие затраты. Общие затраты составили 51930,22 рублей. Были рассчитаны чистая прибыль и рентабельность инвестиций.

Чистая прибыль достигает значения в 19122,22 рублей, рентабельность инвестиций равна 36,8 %.

# ЗАКЛЮЧЕНИЕ

В ходе подготовки к разработке дипломного проекта были изучены теоретические аспекты предметной области, рассмотрены ближайшие аналоги, выявлены их преимущества и недостатки. Выбраны технологии, используемые для написания как клиентской, так и серверной части проекта. Дано краткое описание используемых технологий.

Выполнено системное и функциональное проектирование, в результате которого реализовано клиент-серверное приложение для автоматизации и учета цветочных магазинов.

Все заявленные функции в результате были выполнены в полной мере. Проведены тесты, которые выполнились успешно, что подтверждает работоспособность приложения.

Реализованы вводный и заключительный плакаты, на которых представлены цели, задачи, перечень используемых технологий, указаны достоинства и недостатки реализованного продукта. Данная разработка имеет множество возможных расширений функциональности, которые могут быть реализованы в дальнейшем.

Для пользователей приложения представлена инструкция пользования, описаны функции, доступные через пользовательский интерфейс.

Выполнено технико-экономическое обоснование разработки, оценена трудоемкость работ, произведены расчеты затрат, чистой прибыли и рентабельности разработки. Рентабельность составила 36,8%, что символизирует о актуальности и выгодности создания данного приложения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Веб-сайт компании «ABMcloud» [Электронный ресурс]. – Электронные данные. – Режим доступа: https://abmcloud.com/. – Дата доступа: 02.04.2020.

[2] Веб-сайт программы «CloudShop» [Электронный ресурс]. – Электронные данные. – Режим доступа: https://cloudshop.ru/. – Дата доступа: 02.04.2020.

[3] Веб-сайт программы «FloraPOS» [Электронный ресурс]. – Электронные данные. – Режим доступа: https://florapos.ru/. – Дата доступа: 02.04.2020.

[4] Документация Java [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.oracle.com/en/java/>. – Дата доступа: 02.04.2020.

[5] Современный учебник JavaScript [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.javascript.ru/>. – Дата доступа: 02.04.2020.

[6] Руководство REST API [Электронный ресурс]. – Электронные данные. – Режим доступа: https://restfulapi.net/. – Дата доступа: 02.04.2020.

[7] Документация Spring Framework [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html>. – Дата доступа: 02.04.2020.

[8] Документация Spring Boot [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>. – Дата доступа: 02.04.2020.

[9] Документация Spring Security [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.spring.io/spring-security/site/docs/current/reference/html5/>. – Дата доступа: 02.04.2020.

[10] Документация React [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.reactjs.org/docs/getting-started.html>. – Дата доступа: 02.04.2020.

[11] Документация Node.js [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://nodejs.org/en/docs/>. – Дата доступа: 02.04.2020.

[12] Документация движка V8 [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://v8.dev/docs>. – Дата доступа: 02.04.2020.

[13] Официальный сайт фреймворка Express [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://expressjs.com/ru/>. – Дата доступа: 02.04.2020.

[14] Официальный сайт PosgreSQL [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.postgresql.org/>. – Дата доступа: 02.04.2020.

[15] Документация спецификации JPA [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference>. – Дата доступа: 02.04.2020.

[16] Официальный сайт Hibernate [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://hibernate.org/>. – Дата доступа: 02.04.2020.

[17] Экономическое обоснование проекта по разработке программного обеспечения / В.Г. Горовой, А.В. Грицай, В.А. Пархименко. – Минск: кафедра экономики БГУИР. – 2018. – 12 с.