МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Информационные системы и технологии корпоративного управления
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Разработка облачной платформы для управления бизнес-процессами розничной торговли»

Студент Д.Д. Фролов
(И.О. Фамилия) (личная подпись)

Научный руководитель к. т. н., Т.Г. Султанов
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Оглавление

Введение

На сегодняшний день предприятия различного уровня и размера пытаются увеличить свою прибыль путем внедрения информационных технологий в свои бизнес процессы. В тяжелых условиях конкурентной борьбы каждый предприниматель стремиться наименьшими усилиями и затратами увеличить свою прибыль, но, как правило, различные ERP, CRM системы, развернутые локально или на личном сервере – не всегда лучший вариант и помимо больших затрат, они требуют периодического вмешательства специалистов поддержки и разработчиков для поддержания системы в актуальном для бизнеса состоянии. В решении данного вопроса на помощь бизнесу приходят облачные вычисления.

Облачные вычисления – способ обеспечения удобного сетевого доступа по запросу к определенному общему набору вычислительных ресурсов и программному обеспечению. Потребители облачных вычислений могут снизить свои расходы, связанные с реализацией инфраструктуры информационных технологий.

Актуальность представленной работы обусловлена тем, что увеличение пропускной способности интернет соединения позволило управлять бизнес-процессами без необходимости покупать сервера, разрабатывать и настраивать локальное программное обеспечение. Предприятию остается заботиться непосредственно о ведении бизнеса, переложив ответственность обновления и поддержки программного обеспечения (ПО) на разработчика облачной платформы или приложения. Однако, большинство предприятий не может отказаться от уже имеющихся решений ввиду отсутствия облачных систем, удовлетворяющим их потребностям и позволяющим управлять всеми основными бизнес-процессами, либо сомневается в целесообразности данного решения.

В данной работе нами подробно рассматриваются облачные вычисления и определяются преимущества, которые дает использование облачных технологий предприятиями розничной торговли.

Объект исследования – управление бизнес-процессами розничной торговли.

Предмет исследования – облачные вычисления для управления бизнес-процессами розничной торговли.

Целью магистерской работы является разработка облачной платформы для управления бизнес-процессами розничной торговли.

Задачи магистерской работы:

- исследовать существующие методы и подходы к построению облачных систем и приложений, с целью выявить лучший вариант для реализации, удовлетворяющий нуждам розничной торговли;
- исследовать современные подходы в разработке облачных приложений, для выявления наиболее современного стека технологий;
- спроектировать структуру облачной платформы, для корректной разработки и определения существующих ограничений;
- разработать программное обеспечение для управления бизнес процессами розничной торговли;
- протестировать работу разработанной программы, для определения корректности работы и выявления недостатков;
- определить этапы интеграции разработанной платформы и провести интеграцию предприятий розничной торговли в разработанное решение.

Гипотеза исследования – применение разработанной в рамках диссертационного исследования облачной платформы для управления бизнес-процессами предприятия розничной торговли позволит упростить и удешевить организацию ИТ-инфраструктуры предприятия.

Методы исследования, используемые в магистерской работе: методологии моделирования бизнес-процессов, SOA, облачно-ориентированной подход к моделированию и разработке программного

обеспечения.

Новизна исследования заключается в разработке новой облачной платформы, удовлетворяющей современным запросам предприятий розничной торговли.

Практическая значимость исследования заключается в возможности практического использования разработанной облачной платформы.

Теоретической основой данного исследования являются научные труды российских и зарубежных публицистов и ученых, рассматривающих проблемы проектирования и разработки облачных систем.

На защиту выносятся:

- облачные технологии упрощают организацию ИТ-инфраструктуры предприятия;
- использование облачных технологий позволяет предприятиям снизить расходы на содержание и поддержку работоспособности ИТ-инфраструктуры.

По теме исследования опубликовано 2 статьи:

- Фролов Д. Д. Лучшие практики облачно-ориентированной разработки программного обеспечения // Прикладная математика и информатика: современные исследования в области естественных и технических наук: сб. статей. – Тольятти, 2020. – С. 999-1003.
- Фролов Д. Д. Организация межсервисного взаимодействия в микросервисной архитектуре облачных приложений // Прикладная математика и информатика: современные исследования в области естественных и технических наук: сб. статей. – Тольятти, 2021. (принята к публикации).

Работа изложена на 92 страницах и включает 36 рисунков, 4 таблицы, 30 источников.

Глава 1 Методологические основы разработки и внедрения облачных вычислений для предприятий

Компании или люди, занимающиеся разработкой ПО должны понимать, что приложения, разработанные специально для платформы, на которой они будут работать, будут работать лучше, они будут более устойчивы и просты в управлении. Разработка публичных или частных облачных платформ не является исключением.

Однако мало кто понимает, как именно следует разрабатывать и создавать архитектуру частных или общедоступных облачных приложений. Отсутствие данных навыков и опыта, а также бурный рост популярности облачных вычислений, привели к плохо разработанным приложениям для облачных платформ. Эти приложения не обеспечивают удовлетворения потребностей предприятий и компаний, нуждающихся в них.

Для того, чтобы при разработке своего облачного приложения не допустить тех же ошибок, следует определить концепции и лучшие практики разработки программного обеспечения с учетом новых возможностей облачной среды.

1.1 Методы и подходы к построению облачных систем

Облачные приложения лучше всего развертывать в виде набора облачных сервисов или API. Создание данных обеспечивается с помощью нескольких сервисов, которые объединяются в составные сервисы или готовые составные приложения.

Это концепции называются сервис-ориентированной или микро-сервисной архитектурой. Хотя многие понимают эти концепции, разработчики по-прежнему склонны создавать тесно связанные приложения, ориентированные на пользовательский интерфейс, а не представлять основные функции как сервисы, которые они могут использовать независимо.

Разрабатывая архитектуру приложений для облака, мы имеем дело со сложными распределенными системами, которые могут использовать слабосвязанные приложения, построенные на многих сервисах, которые также можно отделить от данных. Мы можем физически разделить службы приложений, запустив их на различных серверах, а итоговое приложение будет взаимодействовать со всеми ими и предоставлять к ним доступ. Такое, управляющее сервисами приложение часто называют менеджером-управления или каталогом сервисов.

Дополнительным преимуществом такого подхода является возможность повторного использование сервисов из других приложений. Мы можем разбить приложения на сотни базовых сервисов, которые имеют ценность при использовании другими приложениями. Таким образом, мы не изобретаем «велосипед» каждый раз, когда создаем новое приложение.

Первое, что стоит определить, то, что тесное связывание данных – смертельно опасно для облачных приложений. Частные и публичные облака представляют собой сложные распределенные системы, которые лучше всего работают с архитектурами приложений, которые разбивают обработку и данные на отдельные компоненты.

Следует отделять данные по той же причине, по которой следует создать приложение из сервисов. После развязки у нас есть возможность хранить и обрабатывать данные в любом общедоступном или частном облачном экземпляре. Например, многие предприятия настаивают на том, чтобы их данные оставались на локальных серверах, но хотят воспользоваться преимуществами обычных виртуальных машин в публичном облаке [21].

Чтение и запись в базу данных через интернет может привести к задержке, поэтому стоит рассматривать варианты устранения данной проблемы.

Следует рассмотреть возможность использования систем кеширования. Они обеспечивают дополнительную производительность базы данных за счет локального хранения часто используемых данных, тем самым сокращая все

запросы на чтение базы данных обратно в физическую базу данных. Однако они лучше всего встроены в приложение, и их следует протестировать с данными приложения, чтобы определить, насколько эффективным будет кэш. Например, системы, которые постоянно читают новые данные, не получат никакой выгоды от кэширования.

Разделение приложений, как данных, так и сервисов, ещё не означает, что приложение правильно спроектировано для облака. Компоненты приложения, которые постоянно взаимодействуют друг с другом, снизят производительность всего приложения, учитывая, что они обычно распространяются по сети, где возможна высокая задержка.

Следует сосредоточиться на разработке приложений, которые оптимизируют связь между компонентами приложений. Например, объединить коммуникации в один поток данных или группу сообщений, а не постоянно запрашивать данные из приложения в приложение, как будто компоненты приложения находятся на одной платформе.

При разработке облачного приложения следует понимать, как приложение будет масштабироваться при возрастающей нагрузке.

Проектирование для повышения производительности означает с самого начала разработать модель, которая представляет поведение приложения при возрастающей нагрузке. Например, если одновременно регистрируется 1000 или более пользователей, как приложение будет обрабатывать увеличение трафика в сети, увеличение нагрузки на серверы приложений и нагрузку на внутренние базы данных? Необходимо понимать, как компоненты приложения справляются с нагрузкой, когда число пользователей увеличивается до 1000 или более пользователей.

Если не продумать данный момент сначала, вам нужно будет увеличить количество экземпляров серверов приложений. Емкость сети может остаться прежней, но может потребоваться увеличить количество экземпляров базы данных, чтобы справиться с любой дополнительной нагрузкой, что может привести к трудностям дальнейшей поддержки все системы.

Вооружившись этой моделью, мы можем выяснить, как лучше масштабировать приложение. В некоторых случаях поставщики облачных услуг предлагают возможности автоматического масштабирования. Однако наиболее эффективный путь заключается в понимании профиля рабочей нагрузки приложения и определении пути его масштабирования, а также в создании механизмов, обеспечивающих его масштабирование.

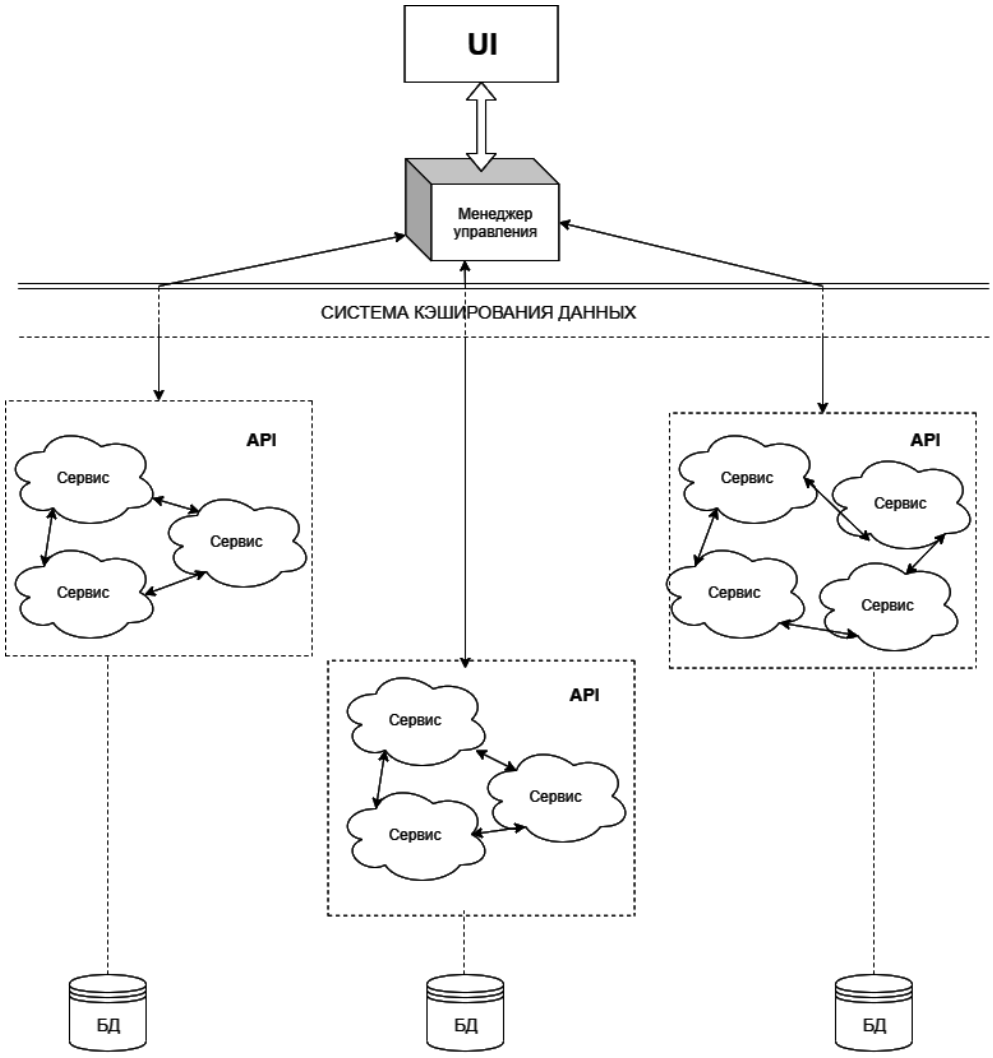

Рисунок 1 – Модель микро-сервисной архитектуры облачного приложения

Современные тенденции, а также примеры успешного прикладного применения идеи микросервисной архитектуры демонстрируют большое количество плюсов данного подхода над монолитной архитектурой облачных приложений.

Стоит отметить, что сама идея не нова и активно использовалась в различном программном обеспечении уже давно, в том числе в принципах проектирования, использованных в UNIX. Однако, возможность использования микросервисного подхода в облачных сервисах, будь то SaaS или PaaS и прочие, долгое время не была оправдана и стала возможна только с ростом пропускной способности сети интернет и ростом производительности и удешевлению серверов.

Помимо плюсов, микросервисная архитектура имеет ряд существенных недостатков [23]. И если ряд этих недостатков относится к неопределенной трактовке и применении данного подхода в конкретных случаях, то другие, весьма очевидны и связаны со сложностью организации межсервисного взаимодействия.

Разберем возможные организации межсервисного взаимодействия в рамках микросервисной архитектуры облачных сервисов (рисунок 2).

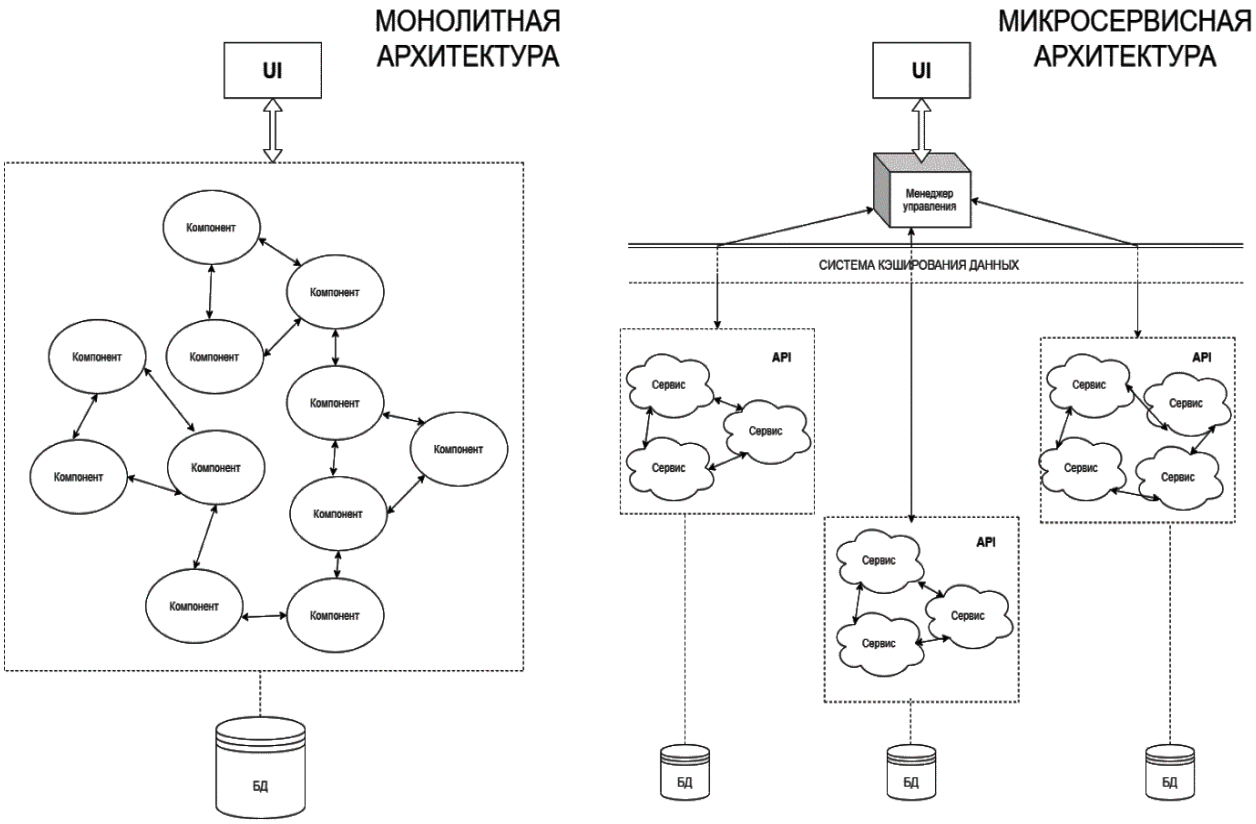

Рисунок 2 – Модели серверной архитектуры облачного приложения

Очевидны становятся сложности грамотной организации межсервисного взаимодействия в микросервисной архитектуры, по сравнению с монолитом.

Часто, при организации взаимодействия между сервисами используют REST или RPC. То есть менеджер управления отсылает запрос в один из сервисов с применением выбранных протоколов и ждет ответ, далее он отправляет запрос в следующий сервис и ждет ответ от него и. т. д. Помимо менеджера управления, соединяющего все результаты воедино, каждый сервис может взаимодействовать с другими сервисами, что ещё сильнее усугубляет ситуацию.

Скорость работы общего запроса в таком случае будет зависит от суммы скорости выполнения запросов в каждый отдельный сервис. Помимо времени на обработку и создания ответа в сервисах имеются затраты на передачу данных и прочие сопутствующие факторы. В итоге, пользователь может достаточно долго ожидать ответ на свой запрос и пользоваться таким облачным сервисом вряд ли будет.

Такой подход, как правило, называют синхронным, и если в случае с монолитной архитектурой он имеет место быть, то в случае с микросервисной возникает желание найти более оптимальный способ межсервисного взаимодействия.

Существуют различные способы оптимизации, такие как кэширование данных и прочее, однако, это скорее полумера, так как для получения динамических данных, которые закэшировать невозможно, придется проследовать по всей этой длинной цепочке.

Существует возможность реализовать асинхронное взаимодействие между сервисами, реализовав в них асинхронный API. Реализацию асинхронного подхода имеют многие современные языки программирования, либо фреймворки и библиотеки, написанные для них. В частности, такую возможность имеет Python, NodeJS и др.

Написание асинхронных сервисов полезно и может решать

обозначенную выше проблему, так как по итогу сервис запрашивающий ответ из другого сервиса не блокируется и может продолжать запрашивать данные из других сервисов, после того как ответ от какого-нибудь из данных сервисов поступит, сервис примет их и обработает.

Написание асинхронных сервисов – хорошее решение, однако, оно не всегда возможно. Для случаев, когда, написание асинхронного сервиса невозможно, асинхронности добиваются другим способом. Иллюстрация данного подхода представлена на рисунке 3.

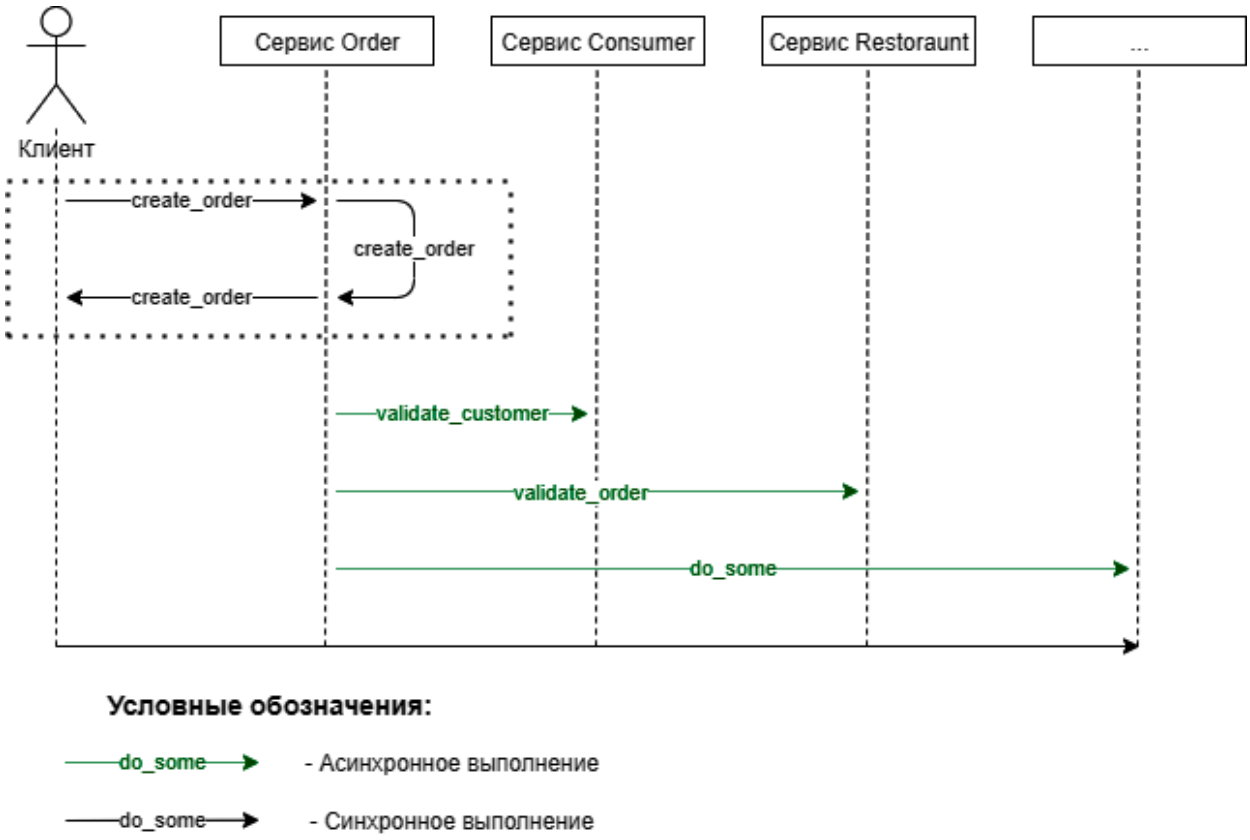

Рисунок 3 – Иллюстрация асинхронного взаимодействия в микросервисной архитектуре с использованием асинхронных сервисов

Ещё одним способом организации асинхронного протокола межсервисного взаимодействия является использование очереди сообщений. Наиболее популярными брокерами сообщений являются RabbitMQ, Apache Kafka, MSMQ, Celery. В более редких случаях для организации очереди сообщений может быть использован Redis.

Каждый из представленных брокеров имеет свои особенности, плюсы и минусы. Однако, общая суть у данных технологий примерно одинаковая.

По сути, для организации асинхронного взаимодействия путем использования брокера сообщений мы ставим задачу на получение данных в некоторый общий узел с работающим на нем брокером, указывая получателя и обработчика данной задачи, не дожидаясь её выполнения. Сервис-исполнитель, получивший задачу выполняет её.

Такой подход хорошо описан в книге Криса Ричардсона «Микросервисы. Паттерны разработки и рефакторинга» [7].

Иллюстрация описанного взаимодействия представлена на рисунке 4.

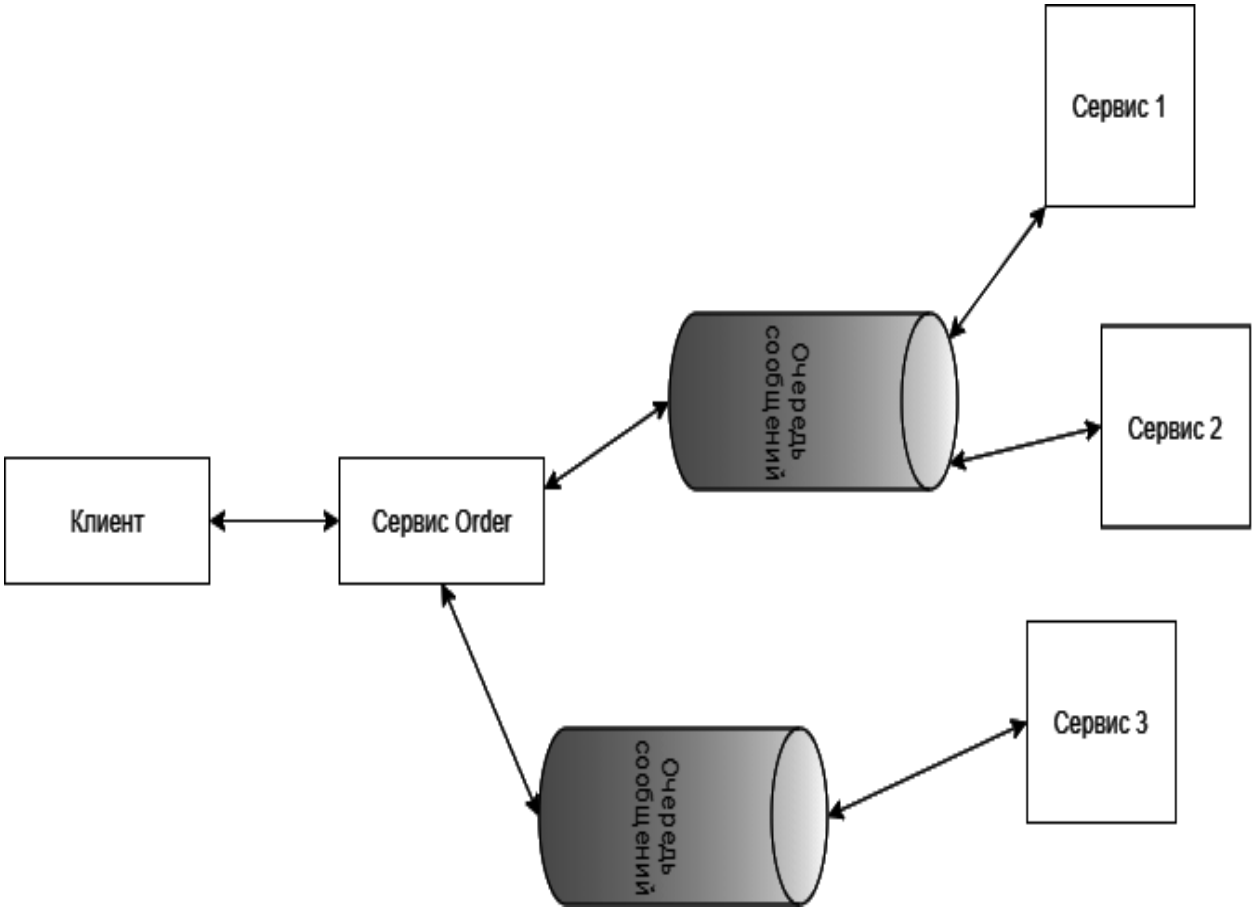

Рисунок 4 – Иллюстрация асинхронного взаимодействия в микросервисной архитектуре с использованием очередей сообщений

Следующим важным способом для минимизации синхронного

взаимодействия является репликация данных, при которых сервис хранит только необходимые ему для выполнения запросов данные (реплику), что позволяет ему не запрашивать каждый раз данные с других сервисов. Иллюстрация итоговой архитектуры с использованием очередей сообщений и репликации данных представленна на рисунке 5.

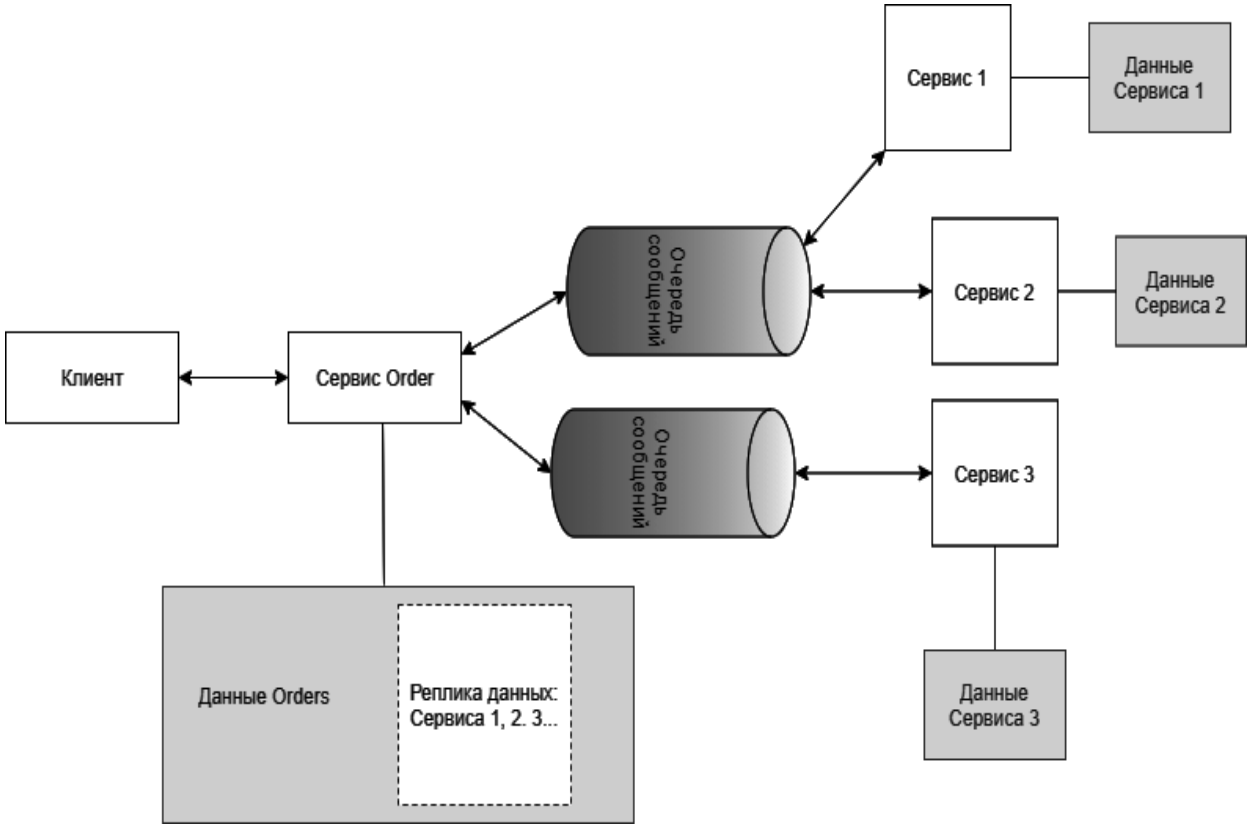

Рисунок 5 – Иллюстрация асинхронного взаимодействия в микросервисной архитектуре с использованием очередей сообщений и репликации данных

Представленные выше подходы определяют лишь общий вектор развития и проектирования микросервисной архитектуры. Конечно, итоговая архитектура может отличаться и не следовать всем представленным выше практикам межсервисного взаимодействия.

Выбор в пользу того или иного подхода в проектировании архитектуры большого облачного сервиса – это сложная и объемная задача, при решении которой необходимо учитывать множество факторов. Однако, представленные в данной магистерской работе рекомендации являются хорошим решением конкретных проблем, так или иначе возникающих при

реализации облачного сервиса, основанного на микросервисной архитектуре.

Наконец, важно отслеживать общую производительность приложения с помощью инструментов мониторинга производительности с учетом приложений и создавать собственные интерфейсы в приложении, чтобы улучшить мониторинг производительности.

Для компаний и людей, которые создают приложения, безопасность, как правило, запоздалая мысль. Однако при размещении приложения в облаке безопасность должна быть главным приоритетом. Облачная архитектура приложения обязательно должна иметь систему безопасности приложения, которая должна быть спроектирована и встроена в архитектуру приложения.

Прежде чем создавать приложение, необходимо выбрать подход и технологию безопасности, которые будут эффективны для разрабатываемого приложения, и будут решать любые проблемы с соответствием или другими проблемами безопасности на уровне данных. Кроме того, приложению необходимо обрабатывать конфиденциальные данные определенными способами с требуемыми уровнями безопасности, такими как шифрование.

Вообще говоря, облачные приложения должны использовать управление идентификацией и доступом (IAM). Предприятия, разрабатывающие зрелые IAM, могут снизить свои расходы на безопасность и, что еще важнее, стать значительно более гибкими при настройке безопасности для облачных приложений.

Во многих случаях IAM будет предоставляться как услуга для предприятия. Эта концепция облачной доставки IAM быстро приводит к концепции централизованного управления идентификацией. По мере создания большего количества облачных приложений с использованием IAM каждое приложение должно стать значительно более безопасным и более экономичным.

Основная задача – обеспечить безопасность приложения и воспользоваться преимуществами встроенных функций облака и системы IAM. Тем не менее, каждое приложение имеет свои собственные требования,

основанные на потребностях бизнеса, и безопасность всегда отличается от одного предприятия к другому.

Создание облачной архитектуры приложений требует, чтобы вы обращали внимание на нюансы данной области, но многие традиционные концепции по-прежнему важны и также должны быть использованы.

Следует понимать, что такие подходы, как ориентация на службы, должны иметь приоритет, даже если это означает более длительные начальные жизненные циклы разработки и увеличения бюджета. Даже если потребуется платить за разработку приложений в облаке больше, чем за традиционную разработку приложений, использование представленных практик значительно окупит поддержку в будущем, поэтому, это разумное вложение.

1.2 Методы и стратегии миграции предприятий в облачную среду

Облачные вычисления в конечном итоге освобождают ИТ-команду предприятия от бремени управления временем безотказной работы. Размещение приложения в облаке часто является наиболее логичным шагом для роста. Положительный ответ на некоторые или все эти вопросы может указывать на готовность предприятия перенести приложение в облако.

Миграция в облако - это процесс перемещения данных, приложений или других бизнес-элементов в среду облачных вычислений [18].

Существуют различные типы миграции в облако, которые может выполнить предприятие. Одна из распространенных моделей - это передача данных и приложений из локального центра обработки данных в общедоступное облако.

Миграция в облако может также повлечь за собой перемещение данных и приложений с одной облачной платформы или поставщика на другую - модель, известная как миграция из облака в облако.

Третий тип миграции - это обратная миграция в облако, репатриация в облако или выход из облака, когда данные или приложения перемещаются из

облака и обратно в локальный центр обработки данных.

Общей целью или преимуществом любой миграции в облако является размещение приложений и данных в максимально эффективной ИТ-среде с учетом таких факторов, как стоимость, производительность и безопасность.

Например, многие организации выполняют миграцию локальных приложений и данных из своего локального центра обработки данных в инфраструктуру общедоступного облака, чтобы воспользоваться такими преимуществами, как большая эластичность, самообслуживание, избыточность и гибкая модель с оплатой по факту использования.

Перенос рабочих нагрузок в облако требует хорошо продуманной стратегии, включающей сложную комбинацию задач управления и технологий, а также перераспределения персонала и ресурсов. Можно выбрать тип выполняемой миграции, а также тип данных, которые следует переместить. Прежде чем принимать меры, важно обдумать следующие шаги по миграции в облако.

Шаг 1. Назначение архитектора миграции.

Перед тем, как начать миграцию в облако, следует определить роль архитектора миграции, и назначить человека, который возглавит работу. Человека, отвечающего за миграцию информационной архитектуры предприятия, обычно называют архитектором миграции. Архитектор миграции - это должность на уровне системного архитектора, ответственная за планирование и завершение всех аспектов миграции; их основная ответственность должна включать определение необходимого рефакторинга, необходимого для успешной миграции, разработку стратегий миграции данных, определение требований к облачным решениям, а также определение приоритетов миграции и механизмов переключения производства.

В ходе большого проекта миграции необходимо принять множество решений и технических планов, и наличие архитектора миграции, отвечающего за все аспекты миграции, имеет решающее значение для успеха проекта.

Шаг 2. Выбор уровня облачной интеграции

Когда вы перемещаете приложение из локального центра обработки данных в облако, есть несколько способов переноса ИТ-инфраструктуры:

– неглубокая облачная интеграция;

– глубокая облачная интеграция;

– полный переход на облачную инфраструктуру.

Для неглубокой облачной интеграции вы перемещаете локальное приложение в облако и не вносите - или ограничиваете - изменения на серверах, которые вы создаете в облаке, с целью запуска заявление. Любых изменений приложения достаточно, чтобы запустить его в новой среде. Вы не используете уникальные облачные сервисы. Эта модель также называется подъемно-сдвиговой, потому что приложение поднимается «как есть» и перемещается в облако без изменений.

Для интеграции с глубоким облаком вы изменяете свое приложение в процессе миграции, чтобы воспользоваться преимуществами ключевых облачных возможностей. Это может быть не более продвинутым, чем использование автоматического масштабирования и динамической балансировки нагрузки, или может быть столь же сложным, как использование возможностей бессерверных вычислений, таких как AWS Lambda, для частей приложения. Это также может включать использование специального облачного хранилища данных, такого как Amazon S3 или DynamoDB.

Полный переход на облачную инфраструктуру заключается в переходе на использование совершенно новых программных систем, разработанных непосредственно для облачных вычислений. При полном переходе следует продумать модель миграции данных и переноса бизнес-процессов в совершенно другую среду [6]. Часто, предприятия игнорируют этот вариант, так как потенциально, он является самым рискованным из представленных, однако, при выборе надежного облачного провайдера, удовлетворяющего всем потребностям бизнеса, такой вариант может показаться самым выгодным.

Шаг 3. Выбор стратегии размещения ИТ-инфраструктуры в облаке

Прежде чем начать миграцию в облако, следует ответить на следующий вопрос: требуется ли предприятию выбрать одного поставщика облачных услуг и перенести свое приложение таким образом, чтобы оно работало оптимизированным для этой единой среды, или, ваши приложения работали на нескольких поставщиках облачных услуг?

Оптимизировать одно приложение для работы с конкретным облачным провайдером относительно просто. У команд разработчиков на предприятии есть только один набор облачных API для изучения, и мигрирующее приложение может использовать все преимущества, которые предлагает выбранный облачный провайдер.

Обратной стороной этого подхода является привязка к конкретному поставщику облачных вычислений. После того, как вы обновили свою ИТ-инфраструктуру для работы только с одним поставщиком, перемещение приложений и данных к другому поставщику может потребовать таких же усилий, как и первоначальная миграция в облако. Кроме того, наличие одного поставщика облачных услуг может негативно повлиять на способность согласовывать важные условия, такие как цены и соглашения об уровне обслуживания, с поставщиком облачных услуг.

Существует несколько различных моделей использования нескольких облачных провайдеров:

Одно приложение в одном облаке; другое приложение в другом облаке. Возможно, самый простой подход с несколькими облаками запускает один набор приложений у одного облачного провайдера, а другой - у другого. Такой подход дает больше возможностей для бизнеса с несколькими поставщиками, а также гибкость в отношении того, где размещать приложения в будущем. Он также позволяет оптимизировать каждое приложение для поставщика, на котором оно работает.

Разделение приложений между несколькими поставщиками облачных услуг. Некоторые компании предпочитают запускать части приложений у

одного поставщика облачных услуг, а другие части - у другого. Этот подход позволяет использовать ключевые преимущества, предлагаемые каждым поставщиком (например, один поставщик может иметь лучшие возможности искусственного интеллекта, чем другой, который известен своей скоростью работы с базами данных). Риск здесь заключается в том, что ваши приложения зависит от производительности обоих поставщиков, и любые проблемы с любым поставщиком могут повлиять на качество обслуживания клиентов вашего приложения.

Создать приложение без привязки к облаку. Другие компании создают свои приложения для работы на любом облачном провайдере. При таком подходе предприятие может запускать приложение одновременно на нескольких поставщиках или разделять нагрузку приложения между ними. Эта модель дает максимальную гибкость при переговорах с поставщиками, поскольку вы можете легко перекладывать нагрузки от одного облачного провайдера к другому. Обратной стороной является то, что может быть сложно использовать ключевые возможности каждого облачного провайдера, что снижает преимущества размещения приложения в облаке. Такой подход может также усложнить процессы разработки и проверки приложений.

Шаг 4. Установить ключевые показатели эффективности облака

Ключевые показатели эффективности (KPI) - это метрики, которые собираются о приложении или сервисе, чтобы измерить, насколько оно соответствует ожиданиям.

Лучшие KPI для миграции в облако показывают, как идет текущая миграция, выявляя видимые или невидимые проблемы, которые могут скрываться в приложении или в выбранном облачном провайдере. Возможно, наиболее важным является то, что ключевые показатели эффективности миграции в облако могут помочь определить, когда миграция завершена и успешна.

Есть несколько ключевых категорий ключевых показателей эффективности облачной миграции, которые мы представили в таблице 1.

Таблица 1 – Ключевые категории показателей эффективности облачной миграции с примерами

Категория	Примеры KPI
Пользовательский опыт	– Время загрузки страниц; – Зависания; – Время отклика; – Продолжительность сеанса.
Производительность	– Частота ошибок; – Пропускная способность; – Доступность.
Инфраструктура	– Использование ЦП; – Производительность диска; – Использование памяти; – Пропускная способность сети.
Деловое участие	– Стоимость обслуживания и использования; – Сторонние финансовые издержки.

Для каждой категории следует определить, какие показатели являются наиболее важными для бизнеса и на какие показатели больше всего повлияет миграция в облако.

Шаг 5. Определить базовые показатели эффективности

Базовая оценка - это процесс измерения текущей (до миграции) производительности вашего приложений или служб, чтобы определить, является ли ее будущая (после миграции) производительность приемлемой. Базовые показатели помогают определить, когда миграция завершена, и обеспечивают подтверждение ожидаемых улучшений производительности после миграции.

Следует установить базовую метрику для каждого ключевого показателя эффективности, который критичен для предприятия, и определить, как долго будут собираться данные для определения базового уровня.

Предприятию также необходимо определить, требуется ли собирать только средние или репрезентативные базовые данные, или включить данные, собранные за «пиковые» или «критические» периоды.

Шаг 6. Расстановка приоритетов для компонентов миграции

Необходимо решить, будет ли предприятие мигрировать все приложения сразу или будете переносить свою ИТ-инфраструктуру в облако компонент по компоненту или сервису по сервису, или же вовсе откажется от текущей инфраструктуры.

Следует определить связи между сервисами, и какие сервисы зависят от других сервисов. Для более крупных и сложных приложений следует использовать инструмент мониторинга производительности приложений, который может использовать карты служб для создания диаграмм зависимостей. Используя диаграмму зависимостей, необходимо решить, какие компоненты следует перенести и в каком порядке. Часто имеет смысл начать с сервисов, которые имеют наименьшее количество зависимостей. В этом случае вы сначала перенесете свои самые внутренние сервисы, а затем последуете со своими внешними сервисами, как правило, ближайшими к вашим клиентам. Альтернативный подход состоит в том, чтобы начать с услуг, ближайших к вашим клиентам, то есть с самых внешних услуг, чтобы вы могли контролировать любое влияние на своих клиентов.

Шаг 7. Создать план переноса данных

Перенос данных - одна из самых сложных частей миграции в облако. Расположение данных может существенно повлиять на производительность всего приложения. Перенос данных в облако, когда методы доступа к данным по-прежнему в основном локальные, может значительно повлиять на производительность. То же самое верно, если данные все еще находятся в локальной среде, но служба, которая обращается к ним, находится в облаке.

Варианты переноса данных включают:

– Использование механизма двунаправленной синхронизации между локальной и облачной базами данных. После перемещения всех потребителей данных в облако локальная база удаляется;

– Использование локальной базы данных с односторонней синхронизацией с облачной базой данных и разрешите потребителям

подключаться только к локальной версии. Когда вы будете готовы, отключите доступ к локальной версии, чтобы облачная версия стала основной базой данных, и разрешите облачным потребителям доступ к новой базе данных;

– Использование облачной службы миграции данных, например, доступную в Amazon Web Services.

Не стоит недооценивать сложность и важность планирования миграции данных. Если не уделить пристальное внимание плану миграции данных до начала миграции в облако, это может привести к сбою миграции или, по крайней мере, к не оправданию ожиданий. Архитектор миграции должен принимать активное участие в процессе планирования миграции данных [4].

Шаг 8. Переключение предприятия на использование облачной ИТ-инфраструктуры

Сроки перехода предприятия с устаревшего локального решения на новую облачную версию зависит от стратегии перехода предприятия в облако и от сложности и архитектуры будущего облачного решения.

Есть два распространенных подхода:

Полный переход. Переход осуществляется непосредственно после того, как миграция в облако была выполнена.

Частичный переход. Перенос некоторых процессов в облачную среду и непременное использование их по частям. Предприятие может выносить некоторые элементы своей инфраструктуры в облако постепенно.

Шаг 9. Проверьте распределение ресурсов приложения.

Даже после того, как предприятие закончило перенос всего в облако, нужно учесть еще несколько вещей. Самое главное - это оптимизация ресурсов.

Облако оптимизировано для динамического распределения ресурсов, и когда вы распределяете ресурсы (например, серверы) статически, вы не пользуетесь сильными сторонами облака. При переходе в облако убедитесь, что у ваших команд есть план распределения ресурсов для вашего приложения. Когда вам нужно выделить дополнительные ресурсы для

приложения в облаке, они обычно доступны у поставщика практически в любом количестве в любой момент.

Представленные шаги миграции в облако охватывают множество вопросов, но определенно, при миграции в облако, следует учитывать и другие вещи, рассмотренные ранее.

Выводы к главе 1

В первой главе представлена и обоснована актуальность исследуемой области, были рассмотрены существующие методы и подходы к построению облачных систем и приложений, а также, проанализированы шаги, которые необходимо выполнить предприятиям, при миграции ИТ-инфраструктуры в облако.

Следует более детально исследовать нужды компаний розничной торговли, определить, какие процессы в их деятельности можно упростить при помощи облачных технологий, а также разработать облачное программное решение, позволяющее увеличивать конверсию и структурировать данные предприятия.

Глава 2 Анализ современных практик внедрения и разработки облачных вычислений для предприятий

Проектирование облачной программной системы и выбор правильных технологий реализации являются важными частями её создания и ошибки, допущенные при проектировании программной части облачной инфраструктуры могут привести к большим проблемам в будущем.

2.1 Анализ методов и подходов облачно-ориентированной разработки программных систем

Прежде чем переходить к выбору методов и подходов к разработке облачно-ориентированных систем следует разобраться с некоторыми основными понятиями, определить, что понимать под облачно-ориентированной архитектурой и понять, какие особенности присущи такому подходу.

Облачно-ориентированная архитектура (COA) - это концептуальная модель, охватывающая все элементы облачной среды. В информационных технологиях архитектура относится к общей структуре информационной системы и взаимосвязям объектов, составляющих эту систему.

Облачно-ориентированная архитектура относится как к сервис-ориентированной архитектуре (SOA), так и к архитектуре, управляемой событиями (EDA), и представляет собой комбинацию двух других архитектурных моделей: ресурсно-ориентированной архитектуры (ROA) и гипермедиа-ориентированной архитектуры (HOA). ROA основан на идее, что любой объект, которому может быть назначен унифицированный идентификатор ресурса, является ресурсом. Таким образом, ресурсы включают в себя не только элементы инфраструктуры, такие как серверы, компьютеры и другие устройства, но также веб-страницы, сценарии и страницы, а также другие объекты.

Разработка глобальной облачно-ориентированной архитектуры является важным строительным блоком Интернета вещей, в котором можно идентифицировать все, что угодно, включая людей, кофе-машины, скамейки в парке и практически любые другие случайные объекты. элемент, о котором вы можете подумать - может быть помечен и подключен через Интернет или аналогичную глобальную сеть [28].

В предыдущей главе было представлено краткое описание методов и подходов к созданию облачных систем и приложений.

Чтобы определить, какую структуру выбрать для облачной программной системы, следует дать детальное описание наиболее широко применяемых сегодня подходов и сравнить их, с целью выявления наилучшей архитектуры для будущего программного продукта.

Сегодня все больше предприятий выбирают облачные сервисы вместо привычного локального программного обеспечения.

По сравнению с локальным аппаратным и программным обеспечением облачные решения, такие как IaaS, PaaS и SaaS, предлагают несколько основных преимуществ:

– масштабируемость. Локальные решения довольно сложно масштабировать, так как тип необходимого оборудования зависит от требований вашего приложения. Если в вашем приложении интенсивный трафик, вам может понадобиться значительно обновить локальное оборудование. Эта проблема не существует с облачным сервисом, который вы можете быстро увеличить или уменьшить с помощью нескольких кликов. Облачные сервисы являются идеальным решением для обработки пиковых нагрузок. Облачные сервисы позволяют предприятиям использовать любые необходимые им вычислительные ресурсы;
– экономичность. Облачные вычисления снижают затраты на оборудование, так как оборудование предоставляется поставщиком. Нет необходимости покупать, устанавливать, настраивать и

обслуживать серверы, базы данных и другие компоненты среды выполнения. Более того, используя облачные решения, вы платите только за то, что используете, поэтому, если вам не нужны дополнительные ресурсы, вы можете просто уменьшить их и не платить за них;

- немедленная доступность. Облачные решения доступны, как только вы заплатите за них, так что вы можете сразу начать использовать облачный сервис. Там нет необходимости устанавливать и настраивать оборудование;
- представление. Облачные компании снабжают свои центры обработки данных высокопроизводительной вычислительной инфраструктурой, которая гарантирует низкую задержку сети для ваших приложений [17];
- безопасность. Облачная инфраструктура хранится в безопасных центрах обработки данных, чтобы обеспечить высокий уровень безопасности. Данные резервируются и могут быть легко восстановлены. Кроме того, поставщики облачных услуг обеспечивают безопасность ваших данных, используя сетевые брандмауэры, шифрование и сложные инструменты для обнаружения киберпреступности и мошенничества.

Преимущества облачных решений огромны, поэтому вполне логичным фактом является то, что рынок облачных услуг сейчас находится на подъеме.

Согласно прогнозу исследовательской и консалтинговой компании Gartner, ожидается, что мировой рынок публичных облачных услуг достиг порядка 383 миллиардов долларов в 2020 году и судя по тенденции продолжает расти.

Диаграмма, построенная по прогнозам компании Gartner представлена на рисунке 6 и включает в себя общий объем рынка IaaS, PaaS и SaaS сервисов. Стоит отметить, что среди трех представленных типов сервисов, наибольший объем рынка будет представлять SaaS – 75,7 миллиардов долларов, на втором

месте IaaS – 71,6 миллиардов долларов, и на последнем месте PaaS который будет составлять 14,8 миллиардов долларов всего объема рынка облачных технологий в 2020 году.

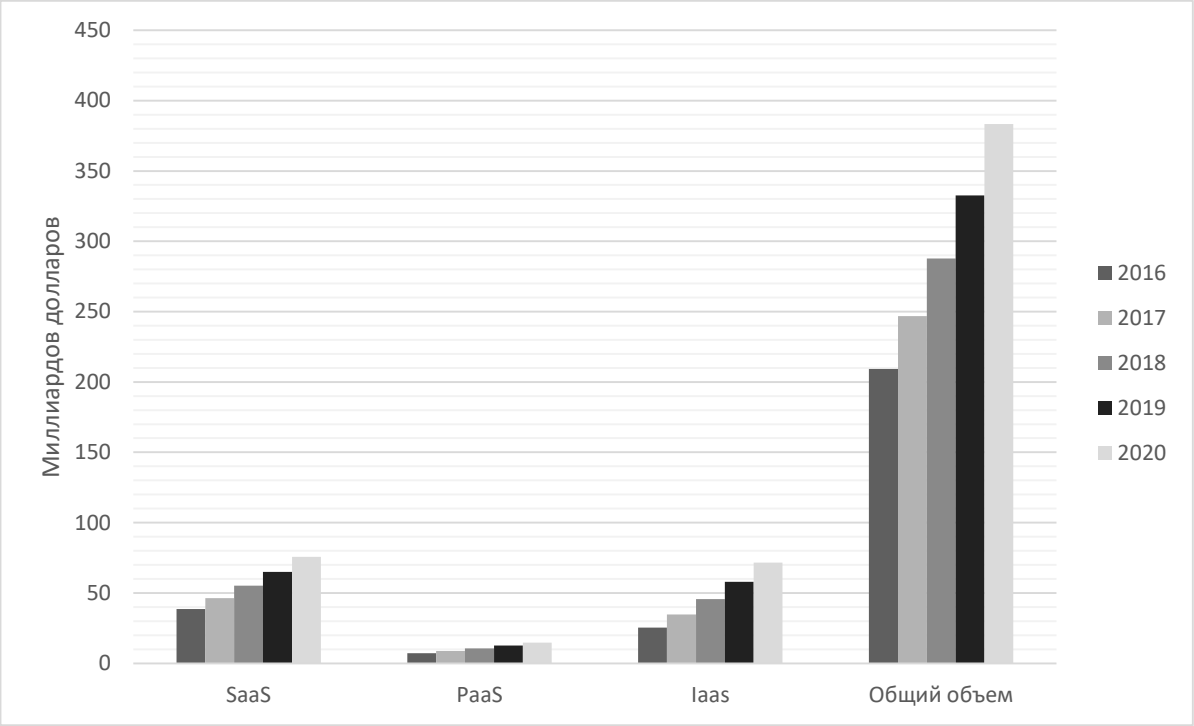

Рисунок 6 – Объем рынка публичных облачных услуг за последние пять лет

Однако выбор правильного облачного сервиса может быть довольно сложным. И у многих предприятий возникают вопросы, связанные с тем, какой же именно тип облачного сервиса подходит под их нужды и конкретную предметную область. Чтобы определиться с тем, какой из типов сервисов будет оптимальным для предприятий занимающихся розничной торговлей, следует провести детальный разбор.

Существует три основных типа облачных сервисов: IaaS, PaaS и SaaS.

Программное обеспечение как услуга (SaaS) позволяет людям использовать облачные веб-приложения. Яркими примерами облачных сервисов SaaS являются Gmail и Hotmail. Другими примерами служб SaaS являются офисные инструменты (Office 365 и Google Docs), программное

обеспечение для управления взаимоотношениями с клиентами (Salesforce), программное обеспечение для управления событиями (Planning Pod) и так далее [13].

SaaS-сервисы обычно доступны с ценовой моделью с оплатой по факту (что означает подписку). Все программное и аппаратное обеспечение предоставляется и управляется поставщиком, поэтому вам не нужно ничего устанавливать или настраивать. Приложение готово к работе, как только вы получите свой логин и пароль. Такой подход идеально подходит конечным пользователям, ведь за размещенные приложения, инструменты разработки и управления, операционную систему, серверы и хранилище, сетевые ресурсы, дата центр и многое другое отвечает провайдер облачных услуг.

Платформа как услуга (PaaS) относится к облачным платформам, которые предоставляют среды выполнения для разработки, тестирования и управления приложениями. Благодаря решениям PaaS разработчики программного обеспечения могут развертывать приложения, от простых до сложных, без необходимости использования всей связанной инфраструктуры (серверов, баз данных, операционных систем, средств разработки и т. д.). Примерами сервисов PaaS являются Heroku и Google App Engine. Поставщики PaaS предоставляют полную инфраструктуру для разработки приложений, а разработчики отвечают за разработку самих программ. Как и SaaS, решения «Платформа как услуга» доступны с ценовой моделью с оплатой по факту [20]. Такой подход идеально подойдет для индивидуальных разработок под конкретные специфичные предприятия со своим уникальным набором бизнес-процессов, для которых нет универсального решения и требуется специализированное программное обеспечение [3].

Инфраструктура как услуга (IaaS) это облачный сервис, который обеспечивает базовую вычислительную инфраструктуру: серверы, хранилище и сетевые ресурсы. Другими словами, IaaS - это виртуальный центр обработки данных [19].

Услуги IaaS могут использоваться для самых разных целей, от

размещения веб-сайтов до анализа больших данных. Клиенты могут устанавливать и использовать любые операционные системы и инструменты, которые им нравятся, в инфраструктуре, которую они получают. Основные поставщики IaaS включают Amazon Web Services, Microsoft Azure и Google Compute Engine [8]. Как и в случае SaaS и PaaS, услуги IaaS доступны по модели «плати за то, что ты используешь». IaaS идеально подходит для: ИТ-администраторов и позволяет более удобным образом обеспечивать поддержку нужной системной архитектуры.

Из представленной информации про каждый тип облачных сервисов следует, что каждый облачный сервис (IaaS, PaaS и SaaS) адаптирован к бизнес-потребностям своей целевой аудитории. С технической точки зрения IaaS дает вам максимальный контроль, но требует обширного опыта для управления вычислительной инфраструктурой, в то время как SaaS позволяет вам использовать облачные приложения без необходимости управления базовой инфраструктурой.

SaaS-решения могут быть использованы для:

- личных целей. Миллионы людей во всем мире используют почтовые службы (Gmail, Hotmail, Yahoo), облачные службы хранения (Dropbox, Microsoft OneDrive), облачные службы управления файлами (Google Docs) и так далее. Люди могут не осознавать этого, но все эти облачные сервисы на самом деле являются сервисами SaaS;
- бизнеса. Компании различных размеров могут использовать решения SaaS, такие как корпоративные почтовые службы (например, Gmail доступен для предприятий), инструменты для совместной работы (Trello), программное обеспечение для управления взаимоотношениями с клиентами (Salesforce, Zoho), программное обеспечение для управления событиями (EventPro, Cvent) и программное обеспечение для планирования ресурсов предприятия (SAP S / 4HANA Cloud ERP).

2.2 Выбор оптимального подхода реализации облачной платформы для управления бизнес-процессами розничной торговли

Даже зная о том, как устроены IaaS, PaaS и SaaS достаточно сложной задачей бывает определить, что же именно использовать. Для того, чтобы определиться, какой из этих типов сервисов лучшим образом подойдет для предприятий розничной торговли определим, для каких целей используется тот или иной подход, а также выделим преимущества и недостатки.

Услуги SaaS предлагают множество преимуществ для частных лиц и предприятий:

- доступ к приложениям из любого места. В отличие от локального программного обеспечения, доступ к которому можно получить только с компьютера (или сети), на котором оно установлено, решения SaaS основаны на облаке. Таким образом, клиент можете получить к ним доступ из любого места, где есть доступ в Интернет;
- можно использовать с любого устройства. Доступ к облачным сервисам SaaS можно получить с любого компьютера. Для доступа необходимо только войти в систему. Многие SaaS-решения имеют мобильные приложения, поэтому к ним также можно получить доступ с мобильных устройств;
- автоматическое обновление программного обеспечения. Клиенту не нужно беспокоиться об обновлении программного обеспечения SaaS, поскольку обновления выполняются поставщиком облачных услуг. Если есть какие-либо ошибки или технические неполадки, поставщик облачных услуг исправит их, а компании-клиенты сосредоточатся на своей работе, а не на обслуживании программного обеспечения;
- бюджетность. По сравнению с локальным программным обеспечением услуги SaaS довольно доступны. Нет необходимости платить за всю ИТ-инфраструктуру. Клиент платите только за услугу

в нужном ему масштабе. Если клиенту нужна дополнительная функциональность, вы всегда можете обновить свою подписку;

– простота в использовании. SaaS-сервисы доступны "из коробки", поэтому их адаптация - это просто. Для доступа чаще всего необходима только регистрация.

Конечно, SaaS-решения также имеют определенные недостатки:

– отсутствие контроля над оборудованием, которое обрабатывает данные клиента;
– только поставщик может управлять параметрами используемого клиентами программного обеспечения.

PaaS-решения в основном используются разработчиками программного обеспечения. PaaS предоставляет среду для разработки, тестирования и управления приложениями. Поэтому PaaS является идеальным выбором для компаний-разработчиков программного обеспечения [11].

Неудивительно, что разработчики программного обеспечения используют такие службы PaaS, как Heroku, Elastic Beanstalk (предлагаемые Amazon Web Services) и Google App Engine.

PaaS предоставляет разработчикам ряд преимуществ [29]:

– сокращение времени разработки. Услуги PaaS позволяют разработчикам программного обеспечения значительно сократить время разработки. Серверные компоненты вычислительной инфраструктуры (веб-серверы, хранилища, сетевые ресурсы и т. Д.) Предоставляются поставщиком, поэтому группам разработчиков не нужно их настраивать, обслуживать или обновлять. Вместо этого разработчики могут сосредоточиться на реализации проектов с максимальной скоростью и качеством;
– поддержка разных языков программирования. Облачные сервисы PaaS обычно поддерживают несколько языков программирования, предоставляя разработчикам возможность реализовывать различные

проекты - от стартапов до корпоративных решений - на одной платформе;

– простое сотрудничество для удаленных и распределенных команд. PaaS предоставляет огромные возможности для совместной работы удаленным и распределенным группам. Аутсорсинг и фрилансинг распространены сегодня, и многие команды разработчиков программного обеспечения состоят из специалистов, которые живут в разных частях мира. Сервисы PaaS позволяют им получать доступ к одной и той же программной архитектуре из любого места и в любое время;
– высокие возможности развития без дополнительного персонала. PaaS предоставляет компаниям-разработчикам все необходимое для создания приложений без необходимости привлечения дополнительного персонала. Все аппаратное и промежуточное программное обеспечение предоставляется, поддерживается и обновляется поставщиком PaaS, что означает, что предприятиям не нужны сотрудники для настройки серверов и баз данных или развертывания операционных систем.

Конечно, облачные сервисы PaaS имеют определенные недостатки:

– пользователь не можете контролировать виртуальную машину, которая обрабатывает его данные;
– PaaS-решения менее гибки, чем IaaS. Например, пользователь не можете создавать и удалять несколько виртуальных машин одновременно.

Решения IaaS могут использоваться для нескольких целей. В отличие от SaaS и PaaS, IaaS предоставляет аппаратную инфраструктуру, которую вы можете использовать различными способами. Это похоже на набор инструментов, которые можно использовать для создания необходимого элемента системы.

Вот несколько сценариев использования IaaS:

- хостинг веб-сайтов или приложений;
- виртуальные дата-центры. IaaS - лучшее решение для построения виртуальных центров обработки данных для крупных предприятий, которым требуется эффективная, масштабируемая и безопасная серверная среда;
- анализ данных. Анализ огромных объемов данных требует невероятных вычислительных мощностей, и IaaS является наиболее экономичным способом их получения. Компании используют инфраструктуру как сервис для анализа и анализа данных.

Инфраструктура как услуга предоставляет следующие основные преимущества для бизнеса:

- никаких расходов на аппаратную инфраструктуру. Поставщики IaaS предоставляют и поддерживают аппаратную инфраструктуру: серверы, хранилища и сетевые ресурсы. Это означает, что предприятиям не нужно вкладывать средства в дорогостоящее оборудование, что существенно экономит средства, поскольку инфраструктура ИТ-оборудования довольно дорогая;
- идеальная масштабируемость Хотя все облачные решения являются масштабируемыми, это особенно относится к инфраструктуре как услуге, поскольку в случае повышенного спроса для вашего приложения доступны дополнительные ресурсы. Приложения также могут быть уменьшены, если спрос низок;
- надежность и безопасность. Обеспечение безопасности данных является обязанностью поставщика IaaS. Аппаратная инфраструктура обычно хранится в специально сконструированных центрах обработки данных, и облачный провайдер гарантирует безопасность данных.

Конечно, облачные решения IaaS имеют и свои недостатки:

– IaaS, как правило, дороже, чем SaaS или PaaS, поскольку клиент фактически арендует аппаратную инфраструктуру;
– все вопросы, связанные с управлением виртуальной машиной, находятся на ответственности пользователя.

Таким образом, исходя из представленных преимуществ и недостатков, можно определить, какой тип сервиса следует разрабатывать для удовлетворения потребностей компаний, занимающихся розничной торговлей.

Очевидным решением для предприятий розничной торговли будет – SaaS, так как большинство предприятий, особенно это касается среднего и малого бизнеса не готовы платить за персонализированную разработку ПО, но при этом, большинство бизнес-процессов у всех подобных предприятий схожи.

При использовании SaaS предприятие не будет беспокоиться о тех проблемах, с которыми бы она могла столкнуться при работе с локальным программным обеспечением, а также, использование SaaS для предприятия будет более выгодным, так как они будут платить лишь за тот набор услуг, который им нужен.

Предприятие в любой момент может отказаться от услуг облачного провайдера и перейти к другому или отказаться вовсе, при этом оно не потеряет начальных вложений, которые бы она потеряла при внедрении локального программного обеспечения.

Ещё одним преимущество использовать именно SaaS для предприятий розничной торговли будет простота в использовании, так как данный тип облачных вычислений не требует специальной квалификации сотрудника и настроить под себя и использовать, если таковая возможность предусмотрена облачным провайдером, сможет обычный пользователь [26].

Для реализации и разработки SaaS сервиса для управления бизнес-процессами предприятий розничной торговли необходимо в первую очередь провести детальное концептуальное и логическое проектирование, изучить

специфику рассматриваемой предметной области, а также провести анализ основных потребностей предприятий.

2.3 Концептуальное моделирование облачной платформы для управления бизнес-процессами розничной торговли

В предыдущем разделе нами было определено, что наиболее предпочтительным вариантом реализации облачной платформы для управления бизнес-процессами розничной торговли будет реализация её как SaaS. Чтобы четко определить предполагаемый функционал разрабатываемой программной системы, необходимо изучить потребности рассматриваемой области, изучить их бизнес процессы и концептуально проработать будущую программную систему.

Представим реализацию основных бизнес процессы предприятия и определим, какие преимущества даст использование разрабатываемой нами облачной платформы для предприятия.

Для иллюстрации бизнес процессов воспользуемся нотацией BPMN.

Нотация моделирования бизнес-процессов (BPMN) - это метод блок-схемы, который моделирует шаги запланированного бизнес-процесса от начала до конца [2].

Являясь ключом к управлению бизнес-процессами, он наглядно отображает подробную последовательность бизнес-операций и информационных потоков, необходимых для завершения процесса.

На высоком уровне BPMN нацелена на участников и другие заинтересованные стороны в бизнес-процессе, чтобы они могли оценить их посредством легкого для понимания визуального представления шагов [22].

На более активном уровне BPMN нацелена на людей, которые будут реализовывать процесс, и дает достаточно подробностей, чтобы обеспечить точную реализацию. Она предоставляет стандартный общий язык для всех заинтересованных сторон, технических или нетехнических: бизнес-

аналитиков, участников процессов, менеджеров и технических разработчиков, а также внешних команд и консультантов. В идеале она устраняет разрыв между намерением процесса и его реализацией, обеспечивая достаточную детализацию и ясность последовательности бизнес-операций.

Представим на рисунках 7 – 10 основные бизнес процессы предприятия розничной торговли с использованием облачной платформы, чтобы проверить гипотезу о том, что платформа удовлетворяет потребностям предприятия.

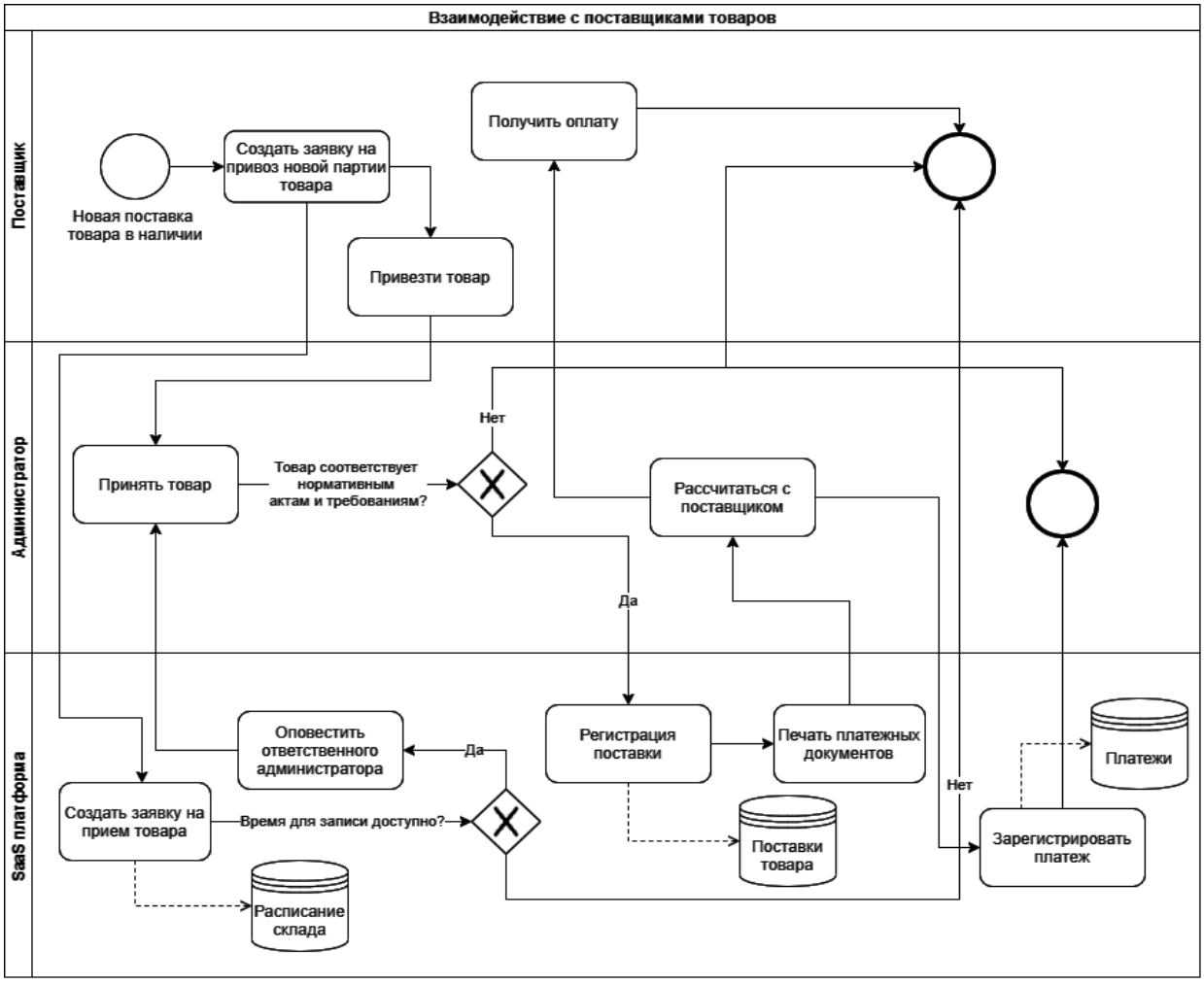

Рисунок 7 – Взаимодействие с поставщиками с использованием SaaS платформы

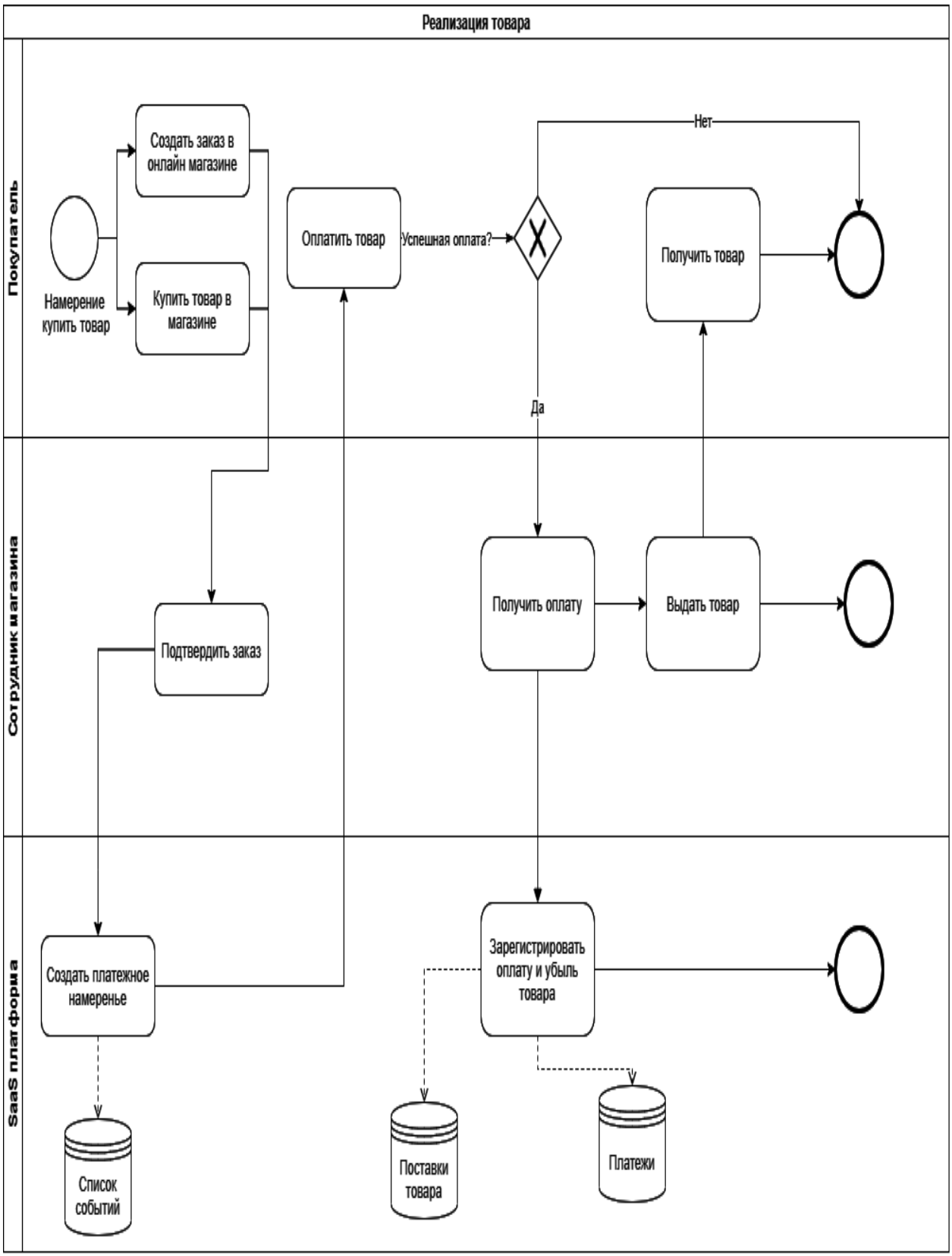

Рисунок 8 – Реализация товара с использованием SaaS платформы

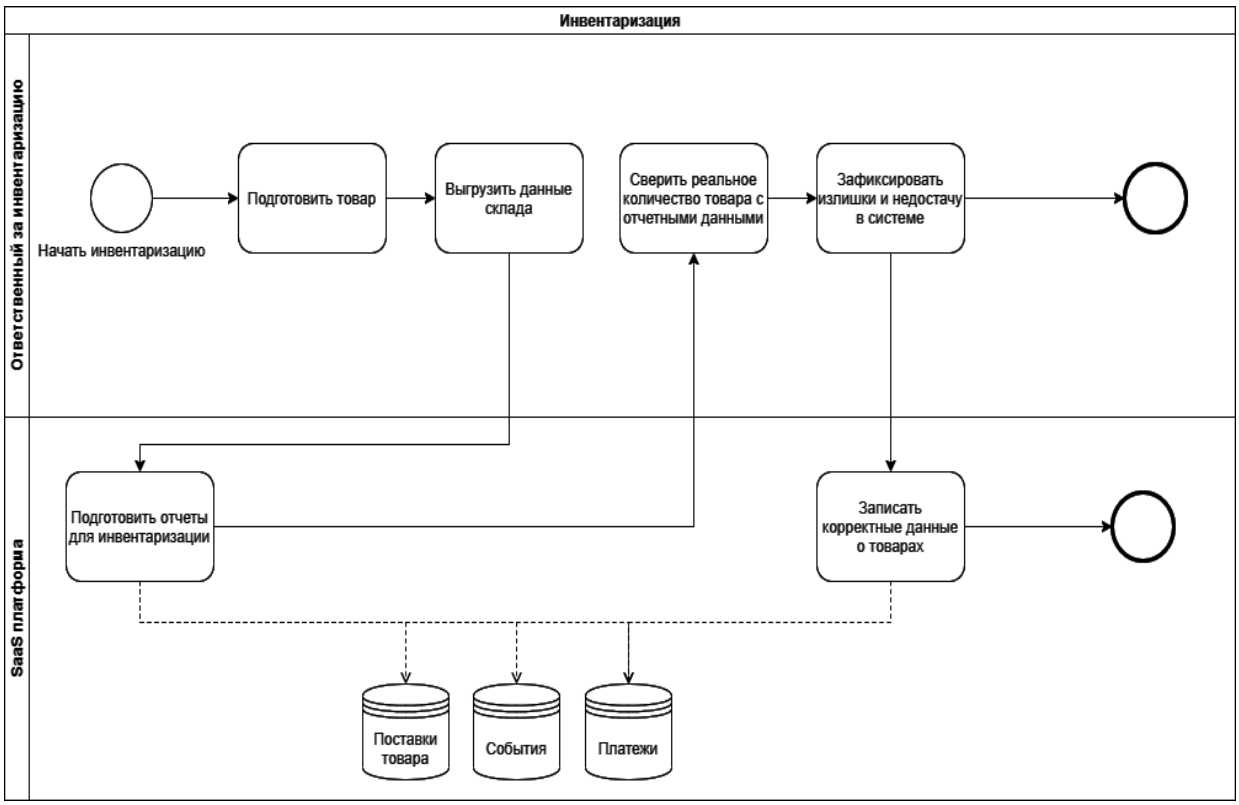

Рисунок 9 – Инвентаризация с использованием SaaS платформы

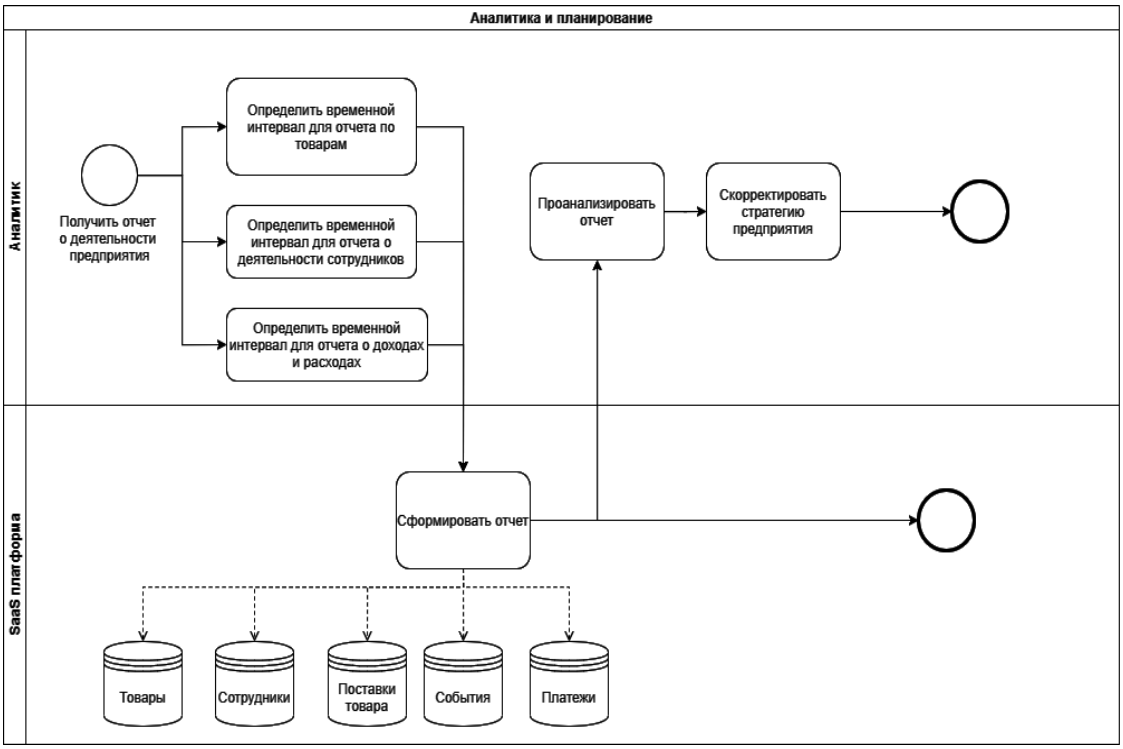

Рисунок 10 – Аналитика и планирование с использованием SaaS платформы

Разрабатываемая программная система позволяет упростить бизнес-процессы предприятия без необходимости в развертывании серверной архитектуры на месте, но при этом дает те же преимущества, как и коробочные ERP системы.

Очевидно, что в зависимости от предприятия представленного и реализованного функционала в облаке может не хватить для удовлетворения всех особых уникальных потребностей предприятия. В связи с этим, следует определить стратегии расширения функционала облачной платформы с целью удовлетворения всех потребностей бизнеса.

Для концептуального проектирования часто используются различные диаграммы, наиболее часто на данном этапе прибегают к проектированию диаграммы вариантов использования (use case diagram) будущей системы [15].

Диаграммы вариантов использования в простейшем виде является представлением взаимодействия пользователя с системой, которая показывает связь между пользователем и различными случаями использования, в которых участвует пользователь. Диаграмма вариантов использования может идентифицировать различные типы пользователей системы и различные варианты использования и часто сопровождается диаграммами других типов. Варианты использования представлены либо кружками, либо эллипсами [10].

В то время как сам сценарий использования может углубиться во все подробности о каждой возможности, диаграмма вариантов использования может помочь обеспечить более общее представление о системе. Ранее было сказано, что «Диаграммы прецедентов - это чертежи вашей системы». Они обеспечивают упрощенное и графическое представление того, что система должна фактически делать.

Из-за их упрощенного характера диаграммы вариантов использования могут быть хорошим средством коммуникации для заинтересованных сторон.

Диаграммы пытаются имитировать реальный мир и предоставляют заинтересованным сторонам представление о том, как будет создаваться система.

Целью диаграмм вариантов использования является просто предоставление высокоуровневого представления о системе и передача требований с точки зрения непрофессионалов для заинтересованных сторон. Дополнительные диаграммы и документация могут использоваться для обеспечения полного функционального и технического представления системы.

Определим базовый функционал, которым должна обладать платформа для управления бизнес-процессами розничной торговли и представим её в виде диаграммы вариантов использования на рисунке 11.

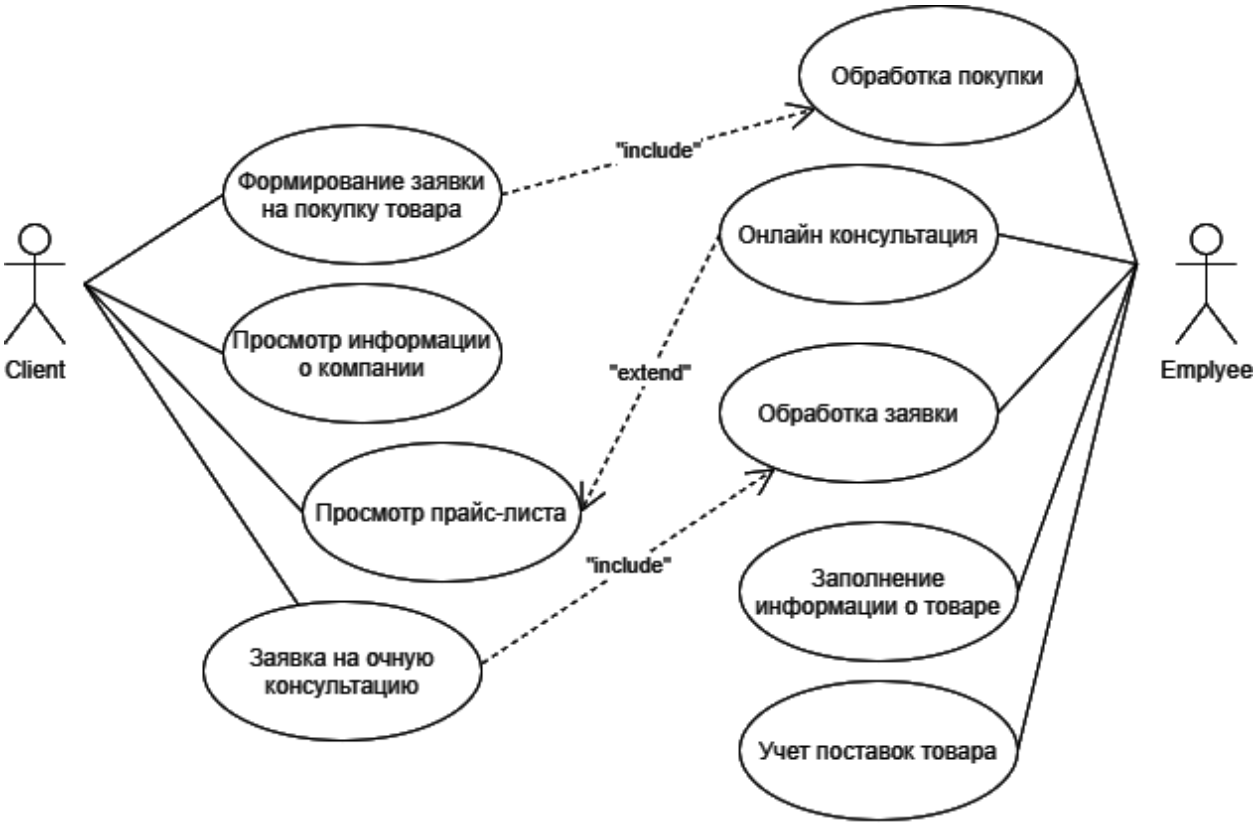

Рисунок 11 – Диаграмма вариантов использования процессов разрабатываемой программной системы

Из рисунка 11 понятен основной функционал будущей программной системы, однако, данная диаграмма не является конечным вариантом и может дополняться в ходе дальнейшего исследования и усовершенствования разрабатываемой системы.

Приступать к разработке программной системы имея на руках лишь диаграмму вариантов использования и BPMN нотации – не оправданное решение и может привести к ряду технических проблем. Чтобы перейти к непосредственной разработке, необходимо и логически рассмотреть структуру будущей облачной программы.

Для реализации задачи логического проектирования часто прибегают к разработке логической модели, которая, как правило, представлена в виде диаграммы классов.

В программной инженерии, диаграмма классов в Unified Modeling Language (UML) представляет собой тип диаграммы статической структуры, которая описывает структуру системы, показывая системы классов, их атрибуты, операции (или методы), и отношения между объектами [5].

Выделяют следующее назначения диаграмм классов:

- демонстрация статической структуры классификаторов в системе;
- обеспечение основных обозначения для других структурных диаграмм, предписанных UML;
- удобство взаимодействия разработчиков и системных архитекторов:
- возможность использования диаграммы бизнес-аналитиками для моделирования систем с точки зрения бизнеса.

Диаграмма классов является основным строительным блоком объектно-ориентированного моделирования. Он используется для общего концептуального и логического моделирования структуры приложения и для подробного моделирования, переводя модели в программный код. Диаграммы классов также могут быть использованы для моделирования данных. Классы в диаграмме классов представляют, как основные элементы, взаимодействия в приложении, так и классы, подлежащие программированию.

Класс в диаграмме классов - описание группы объектов со схожими ролями в системе, которая состоит из:

Структурные особенности (атрибуты) представлять состояние объекта класса и описание структурных или статических особенностей класса

Поведенческие особенности (операции) определяют способ взаимодействия объектов и описывают поведенческие или динамические характеристик класса.

На диаграмме классы представлены прямоугольниками, которые состоят из трех частей:

- верхняя часть содержит название класса. Он напечатан жирным шрифтом и по центру, а первая буква заглавная;
- средняя часть содержит атрибуты класса. Они выровнены по левому краю, а первая буква строчная;
- нижняя часть содержит операции, которые может выполнять класс. Они также выровнены по левому краю, а первая буква строчная.

При проектировании системы ряд классов идентифицируется и группируется в диаграмме классов, которая помогает определить статические отношения между ними. При детальном моделировании классы концептуального дизайна часто делятся на несколько подклассов [27].

Чтобы дополнительно описать поведение систем, эти диаграммы классов могут быть дополнены диаграммой состояний или конечным автоматом UML.

Класс может участвовать в одном или нескольких отношениях с другими классами.

Отношения могут быть одного из следующих типов: наследование (или обобщение), ассоциация, агрегация, зависимость.

Иногда диаграммы классов содержат дополнительную информацию о типах данных, параметрах видимости и прочего для упрощения разработки.

В ходе данной научно-исследовательской работы была разработана диаграмма классов, представляющая логическое устройство будущей программной системы.

Построенная диаграмма классов представлена на рисунке 12.

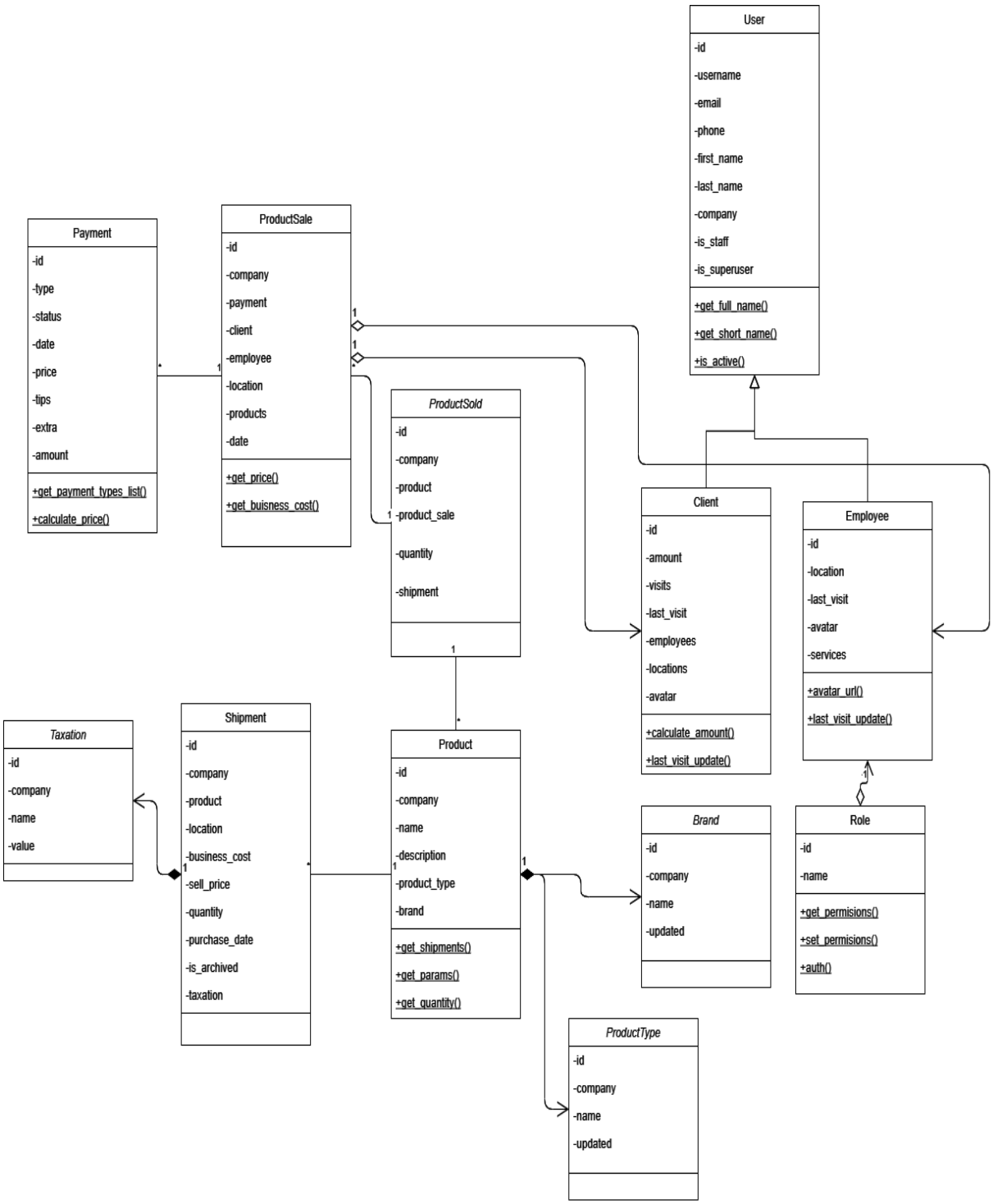

Рисунок 12 – Диаграмма классов разрабатываемой программной системы

Из рисунка 12 видно, что даже с не очень большим набором базовых опций платформа для управления бизнес-процессами розничной торговли имеет достаточно обширную структуру в виде взаимодействующих классов.

Стоит отметить, что также, как и диаграмма вариантов использования,

представленная диаграмма классов будет также дополняться и корректироваться в ходе дальнейшего научного исследования с учетом динамически изменяющихся потребностей современного общества и бизнеса.

Выводы к главе 2

В данной главе мы окончательно завершили все подготовительные этапы перед разработкой программной систем. Нами были проанализированы возможные подход к разработке облачной платформы для управления бизнес процессами розничной торговли и выбрали лучший. Мы выяснили, что из наиболее популярных сегодня облачных услуг: IaaS (Инфраструктура как услуга), PaaS (Платформа как услуга), SaaS (ПО как услуга). Для управления бизнес-процессами предприятий больше подйдет облачное приложения по типу SaaS, так как при данном подходе предприятие не волнуется о многих аспектах поддержки и обновления ПО, а просто платит определенную разработчиком SaaS-приложения сумму за тот функционал, который она использует.

Также, мы определили основные бизнес-процессы предприятий розничной торговли, построили соответствующие диаграммы и провели концептуальное и логическое проектирование предполагаемой программной системы.

В рамках дальнейшей деятельности, перед нами стоит задача реализовать облачную (SaaS) программную платформу, которая будет удовлетворять потребностям бизнеса.

Разработчику SaaS-приложения важно правильно определить, какой стек технологий использовать, так как от этого зависит поддерживаемость разрабатываемого решения и время разработки. Также, необходимо учитывать рекомендации, приведенные в первой главе данной работы и составить четкие требование к разрабатываемой облачной платформе.

Глава 3 Проектирование и разработка облачной платформы управления бизнес-процессами розничной торговли

В рамках данной главы перед нами стоит задача реализовать облачную программную систему, которая будет удовлетворять потребностям бизнеса и соответствовать всем критериям, выделенным в предыдущих главах.

3.1 Описание и обоснование выбора стека технологий для реализации облачной платформы

Разработка приложений под различные платформы и задачи ведется с помощью языков программирования, разработанных для конкретных целей цели. Поэтому, при рассмотрении облачного программирования, необходимо выбирать языки, ориентированные на облачную разработку, вместо языков общего назначения для получения более качественных продуктов.

Чтобы определить, какой язык программирования лучшим образом подойдет для разработки облачных вычислений, следует выделить наиболее часто используемые. Рассмотрим пять из них.

Вот подборка из 5 лучших облачных языков программирования с подробным описанием их характеристик и преимуществ. В этой статье вы найдете лучшие инструменты для разработки облачных вычислений.

Java - это не только язык программирования общего назначения, так как он также позиционирует себя в облачных вычислениях. Его популярность среди разработчиков огромна, так как его используют более 10 миллионов программистов и выполняют в более чем 15 миллиардах терминалов по всему миру. Универсальность языка Java позволяет использовать его при разработке приложений для Android, настольных компьютеров, веб-сайтов и игр. Это делает его подходящим практически для любой задачи программирования.

Среди преимуществ, которые можно упомянуть, они выделяются:

- легкость в изучении;
- продуманная архитектура;
- ориентированность на создание модульных программ и многократное использование уже написанного кода;
- кроссплатформенность, программы на Java легко перенести с одной операционной системы на другую.

Возможность запуска одной и той же программы на многих разных системах обеспечивает бесперебойную работу Java в облачных вычислениях. Благодаря своей надежности, простоте использования, кроссплатформенным возможностям и функциям безопасности, он занимает привилегированное положение среди программистов, разрабатывающих интернет-решения.

C# (Asp.Net) - это язык программирования, разработанный Microsoft для создания веб-сайтов и веб-приложений со многими функциями. Он характеризуется предложением высококачественных решений с динамическими веб-страницами, которые можно прос матривать в разных браузерах. Он позволяет создавать надежные и многоразовые приложения, поэтому данный язык очень популярен и относительно прост в использовании. Преимущества данного языка следующие:

Безопасность приложений (обеспечивается встроенной проверкой подлинности Windows);

- высокоуровневость, простая реализация решений типовых задач;
- простое создание динамических веб-страниц;
- удобство поддержки и управления приложениями;
- встроенные функции кэширования.

Представленные характеристики делают Asp.Net хорошим выбором для реализации облачных решений.

PHP - это язык программирования, широко используемый для веб-разработки и облачных вычислений, что является основным направлением применения данного языка. PHP имеет низкий порог вхождения и очень прост

в освоении, поэтому он популярен, когда дело доходит до автоматизации веб-сайтов и других функций связанных с веб-приложениями.

Этот язык может работать на серверах UNIX и Windows и имеет мощный выходной буфер. PHP может использоваться с большим количеством систем управления базами данных, поэтому он работает на самых популярных веб-серверах и доступен для многих различных операционных систем. Это полностью объектно-ориентированный язык, помогающий создавать большие и сложные веб-приложения.

Python обычно определяют, как язык высокого уровня, созданный для удобства чтения, поэтому на нем очень просто начать программировать. За почти 30 лет своего существования, он активно развивается и до сих пор остается одним из предпочтений разработчиков программного обеспечения!

Python известен как универсальный язык программирования, поддерживающий возможности разработки в различных областях. Благодаря встроенным модулям язык Python широко применяется в создании веб-приложений, API, и для разработки всего, что связано с наукой о данных.

Python сочетает в себе несколько функций, улучшающих программирование, таких как наличие сторонних модулей, обширные библиотеки поддержки, разработка с открытым исходным кодом и сообщества, простота изучения и доступная поддержка, простые в использовании структуры данных, производительность и скорость. Преимущества python делают его одним из предпочтительных языков для разработки облачных вычислений, среди них:

- высокоуровневость;
- кроссплатформенность;
- динамическая типизация;
- паттерны проектирования заложены в основу языка;
- богатое множество библиотек и фремворков для веб-разработки и не только.

Python очень хорошо демонстрирует себя в качестве языка для облачных

технологий. На Python реализовано большое количество популярных облачных сервисов, таких как Youtube, Instagram, Pinterest, Twitter и другие.

Ruby является идеальным языком программирования для облачных вычислений для начинающих, так как он легок в использовании и освоении. Ruby и фреймворки на данном языке (в частности, Ruby on Rails) предоставляют огромную экосистему для разработки качественных веб-приложений, а также для решения различных задач облачных вычислений.

Освоение Ruby открывает множество возможностей в области облачных вычислений, потому что этот язык имеет обширные ресурсы для разработки различных приложений, а также более 60 000 библиотек и фреймворков на выбор.

Облачное программирование продолжает расширяться в этом технологически ориентированном мире. В дополнение к традиционным языкам появились новые, которые предлагают большие преимущества для разработки и выполнения различных приложений. Среди представленных в данной работе языков программирования достаточно сложно сделать выбор, так как каждый из них обладает своими преимуществами для реализации облачных вычислений. В выборе языка для облачных вычислений следует также опираться и на требования к задаче.

Проанализировав популярные языки программирования, мы пришли к выводу, что наиболее подходящим выбором для реализации SaaS платформы для управления бизнес процессами розничной торговли будет Python, в силу своей простоты, продуманной архитектуре и большому количеству библиотек, позволяющих применять его для решения широкого спектра задач.

Для более детального определения всего стека технологий следует решить сопутствующие вопросы – каким образом реализовывать пользовательский интерфейс, в каком виде хранить и передавать данные, а также определиться, какие ещё смежные технологии предстоит использовать.

С точки зрения пользовательского интерфейса для SaaS выбор будет не очень большим, так как здесь главенствующую роль занимают три основные

технологии:

- язык гипертекстовой разметки (HTML) - это стандартный язык разметки для документов, предназначенных для отображения в веб-браузере;
- каскадные таблицы стилей (CSS) - это язык таблиц стилей, используемый для описания представления документа, написанного на языке разметки, например, HTML;
- JavaScript (JS) – высокоуровневый мультипарадигменный динамически типизированный язык программирования. Данный язык использовался для написания динамических веб-страниц, однако сегодня позволяет реализовывать и серверную логику, и мобильные и настольные приложения.

В случае с пользовательским интерфейсом для SaaS выбор как правило заключается в определении требуемых библиотек и фреймворков для JavaScript. Разберем наиболее популярные решения.

Angular.js – фреймворк созданный и поддерживаемый компанией Google, является одним из самых известных фреймворков JavaScript, который дает разработчикам лучшие подходы для объединения JavaScript с HTML и CSS. Это интерфейсная веб-инфраструктура с открытым исходным кодом, которая позволяет разработчикам программного обеспечения собирать интуитивно понятные одностраничные веб-приложения (SPA).

React.js – библиотека созданная компанией Facebook, представляет собой чрезвычайно эффективную библиотеку JavaScript с открытым исходным кодом, используемую для создания интуитивно понятных пользовательских интерфейсов. Эта библиотека, по сути, является структурной библиотекой пользовательского интерфейса, однако она идеально подходит для создания адаптивных одностраничных приложений и межуровневых приложений. React.js быстрее Angular.js и является идеальным решением для людей, которым нужна базовая, универсальная и быстрая среда.

Vue.js - однин из самых знаменитых JavaScript-фреймворков, созданный

Evan You в 2014 году. Это легковесная библиотека с открытым исходным кодом, используемая для удобной сборки пользовательских интерфейсов. Огромное количество сайтов используют данный фреймвор, включая сайты многих важных организаций. Vue призван быть универсальным и использоваться для постепенной разработки пользовательского интерфейса.

В рамках представленной работы для реализации пользовательского интерфейса разрабатываемой SaaS платформы было решено использовать React.js, так как он представляет грамотную структуру компонентов и не нагромождён большим количеством компонентов, которые не будут использоваться. При необходимости расширения достаточно просто будет установить необходимый модуль.

С точки зрения хранения данных также предстоит сделать выбор между двумя наиболее популярными сегодня подходами – реляционный подход или нереляционный подход, часто называемый NoSQL («Не только SQL» или «Не SQL»).

Реляционная база данных представляет собой цифровую базу данных на основе реляционной модели данных (в виде таблиц и связей между ними). Системы программного обеспечения, используемое для поддержания реляционных баз данных является реляционная система управления базами данных (СУБД). Многие системы реляционных баз данных имеют возможность использовать SQL (язык структурированных запросов) для запросов и обслуживания базы данных.

NoSQL - это нереляционная система хранения и обработки данных, которая не требует фиксированной схемы, избегает объединений и легко масштабируется. База данных NoSQL используется для распределенных хранилищ данных с нечеткой структурой.

Так как разрабатываемая SaaS платформа для управления бизнес процессами предприятий розничной торговли имеет четкую структуру, то и для хранения данных будет лучшим решением использовать именно реляционный подход.

Так как в качестве серверного языка был выбран Python, то в качестве СУБД лучше всего будет выбрать PostgreSQL, так как она является свободной объектно-реляционной базой данных и имеет активно обновляемую библиотеку для взаимодействия с Python [25].

Наконец, для программирования интерфейса на языке Python лучшим вариантом будет использование специальных фреймворков. Наиболее популярными сегодня являются Django, Flask, Aiohttp.

Так как для реализации пользовательского интерфейса было решено использовать библиотеку React.js, то необходимо использовать те фреймворки, которые поддерживают разработку Rest-API. Все три представленных фреймворка имеют такую поддержку, однако реализованы каждый по-своему. Рассмотрим преимущества и недостатки каждого из представленных фреймворков.

Django – свободный веб-фреймворк на Python, который часто используется для реализации различных e-commerce решений. Однако, благодаря Django REST Framework может быть использован для реализации микросервисов и REST-API. На сегодняшний день это самый популярный веб-фреймворк для Python и начиная с 3й версии поддерживает асинхронную работу «из коробки». Django имеет множество преимуществ, из-за которых разработчики останавливают свой выбор именно на нем:

- один управляющий скрипт (manage.py), который можно использовать для выполнения большинства специфических для фреймворка действий (таких как запуск сервера разработки, создание пользователя-администратора, сбор статических файлов и т. д.),
- поддержка как синхронной, так и асинхронной обработки запросов;
- приверженность паттернам проектирования и объектно-ориентированному подходу;
- настраиваемое объектно-реляционное отображение (ORM);
- собственный стиль кодирования;

- хорошая документация;
- гибко настраиваемая система маршрутизации URL;
- соответствие стандарту WSGI;
- популярность, в связи с которой можно приписать большое активное сообщество, наличие большой кодовой базы и внешних компонентов, упрощающих разработку.

К недостаткам Django можно отнести его ориентированность на монолитные приложение и строгую структуру.

Flask – микросервисный веб-фреймворк на Python, является вторым по популярности после Django, имеет большое сообщество, а также имеет большое количество расширений. Flask в себя включает лишь основные инструменты взаимодействия с HTTP и работу с шаблонизатором, однако путем расширений позволяет расширить свой функционал:

- синхронная обработка запросов;
- гибкость в выборе архитектуры проекта компонентов;
- хорошая документация;
- система маршрутизации Werkzeug;
- соответствует стандарту WSGI [30];
- поддерживает базовую статическую маршрутизацию;
- имеет большое количество сторонних модулей, расширяющих функциональность.

К недостаткам Flask можно отнести отсутствие ORM и скудную базовую функциональность.

Aiohttp - исключительно асинхронный HTTP клиент / сервер для asyncio и Python, чаще всего используемый для реализации микросервисов и работы с websocket.

Преимущества:

- асинхронная обработка запросов, клиентские и серверные веб-сокеты;
- поддерживает функциональные и классовые представления;

- поддерживает Postgres, MySQL, Redis асинхронные драйверы,
- гибко настраиваемая система маршрутизации,
- множество сторонних модулей, расширяющих функциональность.

Aiohttp имеет и ряд недостатков:

- сложная, плохо структурированная документация;
- отсутствие шаблонного движка;
- отсутствие поддержки WSGI.

Проанализировав существующие фреймворки, было решено использовать Django в связке с Django REST Framework, так как на сегодня это самое популярное и проверенное решение для веб-приложений на Python [14].

Подводя итог, в рамках данного раздела был определен весь стек технологий, который планируется использовать для разработки будущей SaaS платформы. Представленный стек соответствует современному надёжному облачному приложению и может быть успешно использован для разработки облачных систем.

3.2 Постановка требований к разрабатываемой облачной платформе

Прежде чем переходить к разработке требований к реализуемому программному обеспечению, стоит понимать, что практическая реализация облачной платформы это один из самых сложных этапов в представленной научно-исследовательской деятельности.

Ошибки, допущенные при разработке программного обеспечения, непосредственно будут видны пользователю, поэтому их следует избегать как на этапе определения требований, так и на этапе разработки и кодирования.

Так как в предыдущей работе уже был выбран наиболее удачный стек технологий (Python, Django, Django REST Framework, PostgreSQL, HTML, CSS, JavaScript, React.js) для реализации разрабатываемой облачной платформы, то необходимо определить требования к взаимодействию этих

технологий, используемых на серверной и клиентской части, а также определить к требованию к дизайну, функциональности пользовательского интерфейса, к безопасности программного обеспечения и процессам, реализуемым серверной программой.

3.2.1 Разработка дизайна и постановка требований к пользовательскому интерфейсу облачной платформы

Очевидно, что к клиентским технологиям из выбранного стека относятся HTML, CSS, JavaScript, React.js. Поэтому будем формировать требования согласно выбранному стеку. В целом, стоит отметить, что особых ограничений выбранный стек не накладывает и является одним из наиболее популярным и очевидным выбором при разработки современных web-приложений.

Первым и главным требованием, предъявляемым к пользовательскому интерфейсу, является то, что наше приложение должно быть SPA (Single-Page Application, одностраничное web-приложение), чтобы переход пользователя с локального программного обеспечения был наименее болезненным.

Разрабатываемый интерфейс должен реализовывать основные бизнес процессы предприятий розничной торговли: планирование деятельности предприятия, учет поставок, учет бизнес-встреч, учёт доходов и расходов, складской учет, а также доступ к этим процессам должен быть простым и понятным.

Учет и планирование деятельности предприятия будет осуществляться на удобном календаре, где для каждой точки (отделения магазина) можно просматривать и планировать деятельность: назначать встречи, планировать поставку товаров, следить за статусами поставки и. т. д. Дизайн страницы планирования представлен на рисунке 13. Дизайн страницы используемых услуг представлен на рисунке 14.

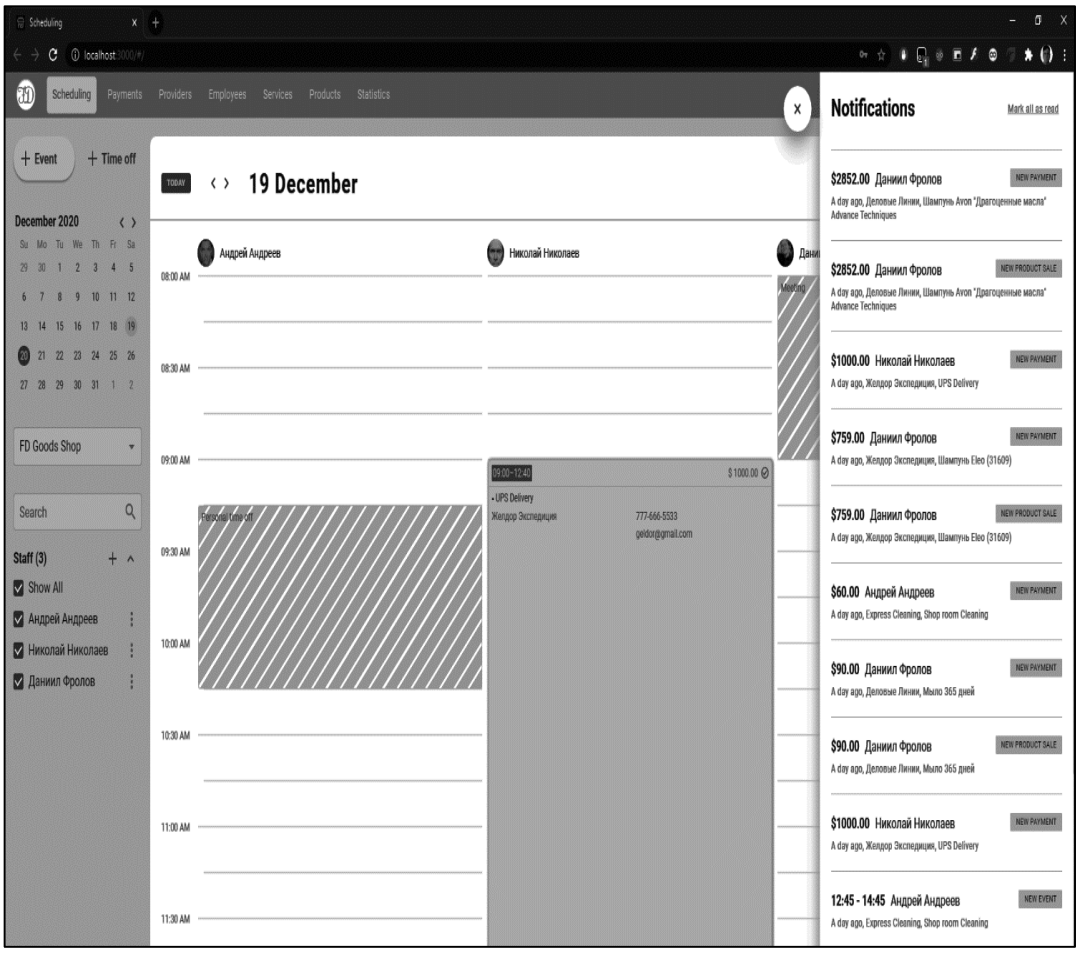

Рисунок 13 – Окно планирования деятельности предприятия розничной торговли и учета бизнес встреч

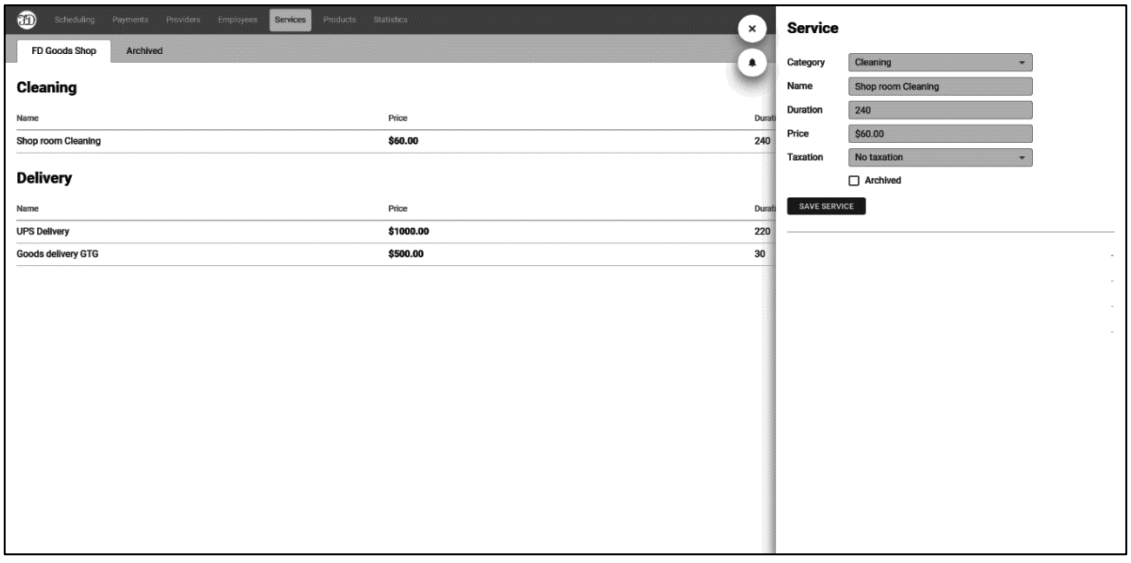

Рисунок 14 – Окно просмотра создания и редактирования используемых услуг

Для планирования поставок и прочих услуг для предприятия, занимающегося розничной торговлей необходима возможность создания новых поставщиков или исполнителей услуг.

Окно для просмотра, добавления и редактирования исполнителей услуг представлены на рисунке 15.

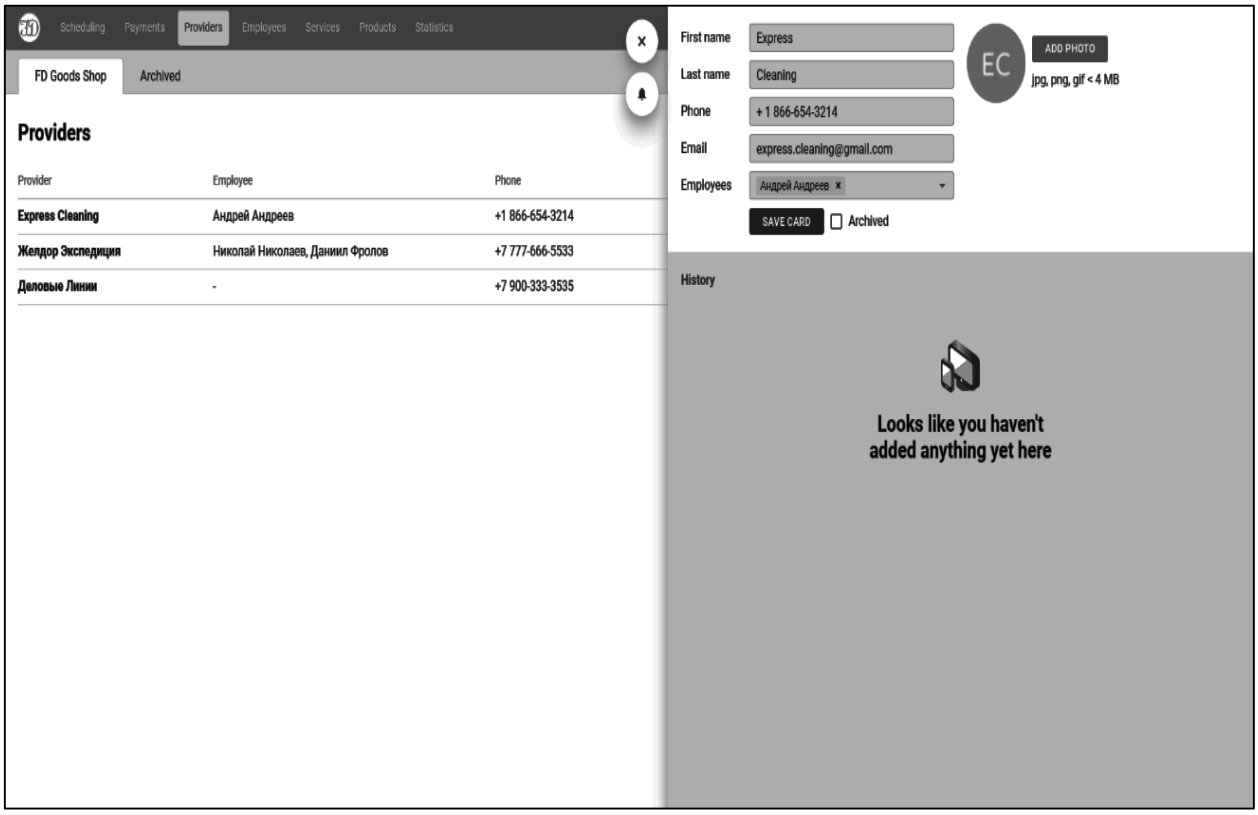

Рисунок 15 – Окно для просмотра, добавления и редактирования списка исполнителей услуг

Помимо исполнителей услуг в нашей системе должна быть возможность указать ответственного работника, а также, работник потребуется в фиксировании продажи.

Окно создания и редактирования сотрудников представлены на рисунке 16.

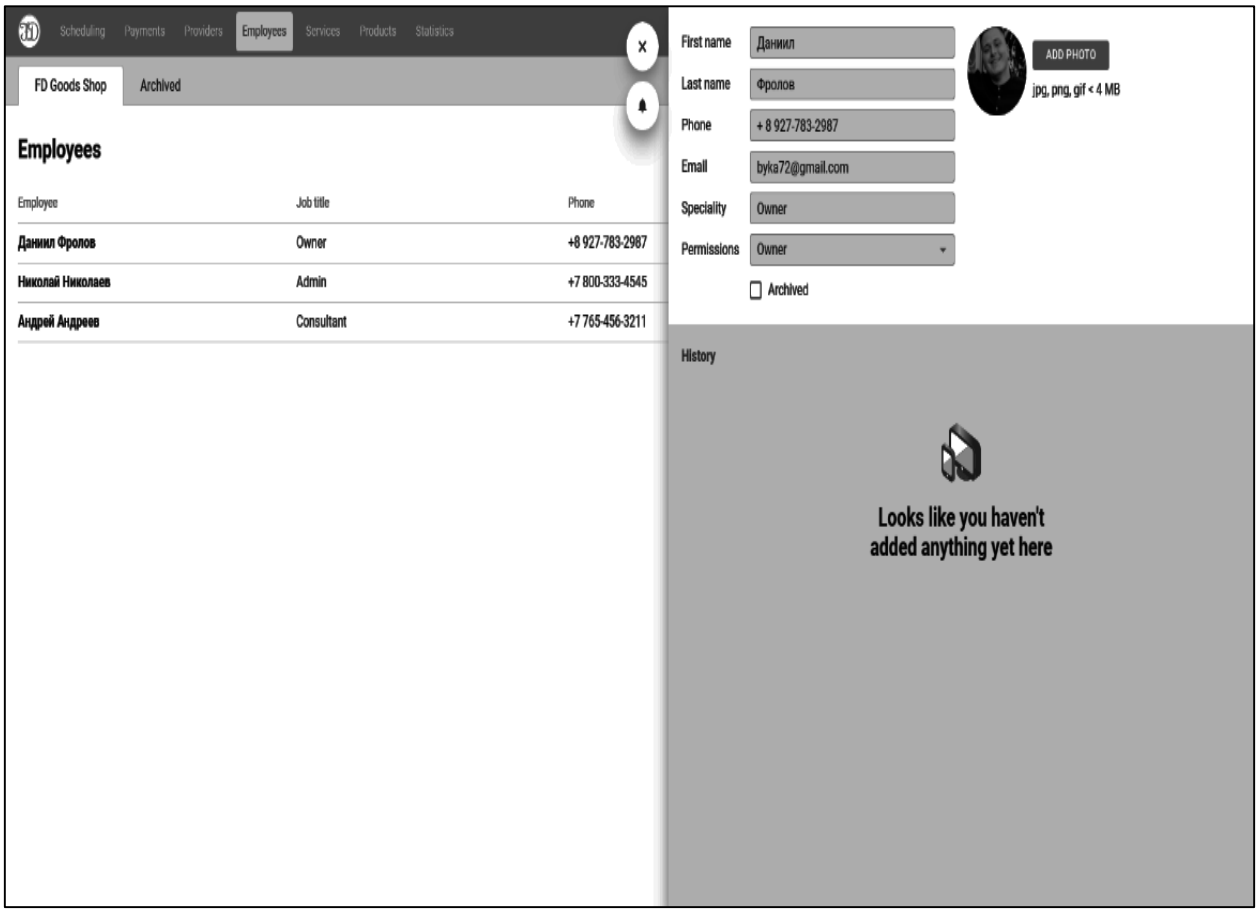

Рисунок 16 – Окно для просмотра, добавления и редактирования работников

Очень важной частью бизнес-процессов предприятия, занимающегося розничной торговлей занимает складской учет.

Важно, чтобы у пользователя данной системы была возможность создать продукт, указав его бренд и тип, а также возможность создать партии продукта и указать для них стоимость закупки, стоимость продажи и дату поступления партии.

Окна складского учёта представлены на рисунках 17-21.

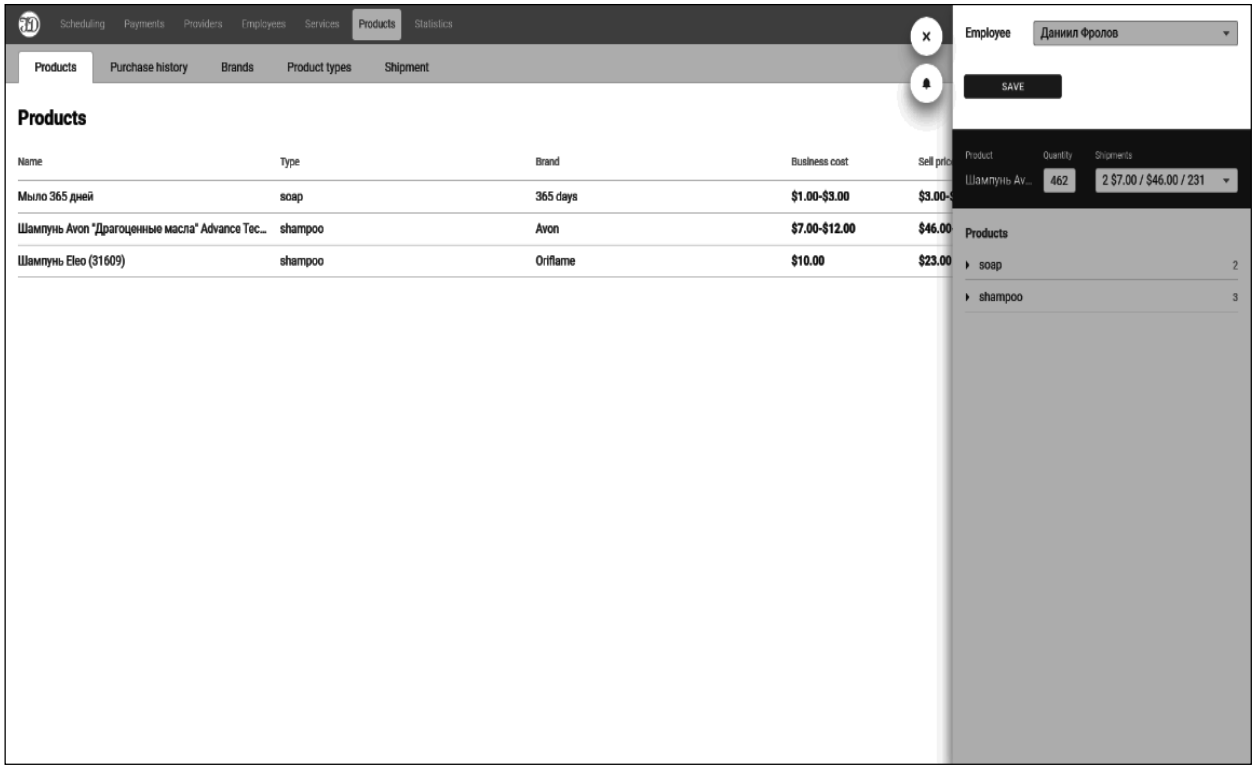

Рисунок 17 – Окно для просмотра и создания продажи продукта

Рисунок 18 – Окно для создания и редактирования продукта и его поставок

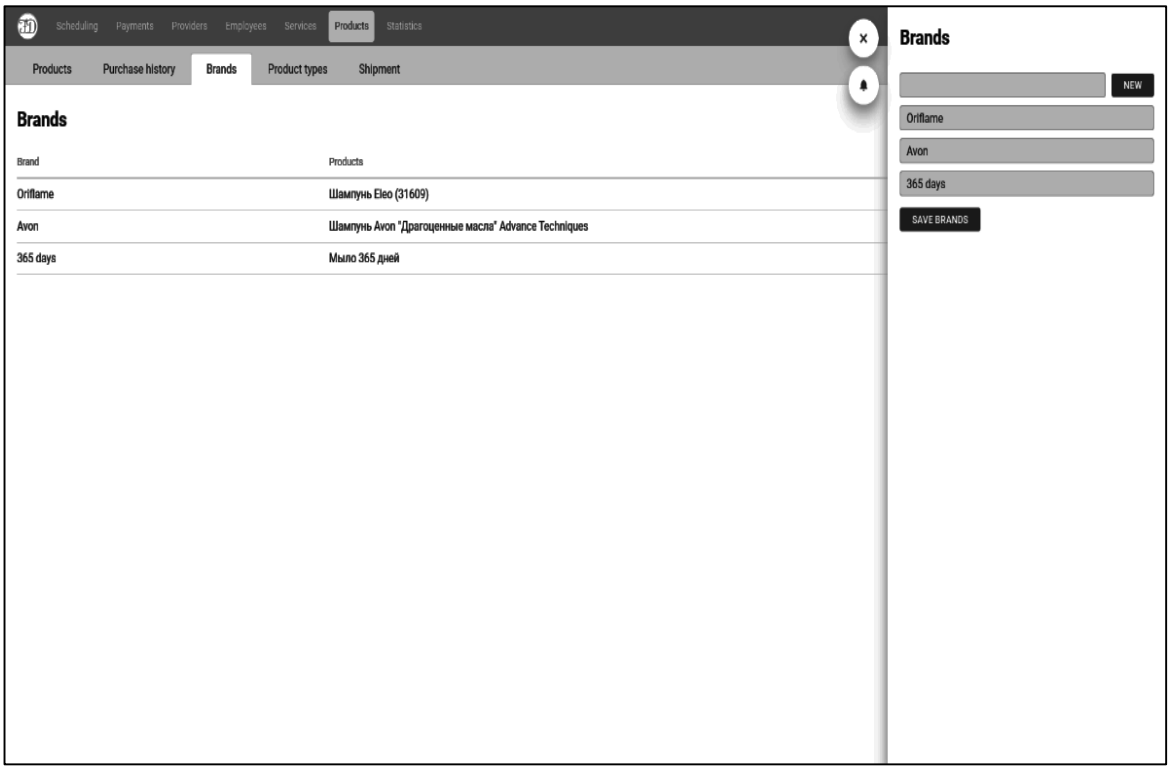

Рисунок 19 – Окно для просмотра, добавления и редактирования брендов

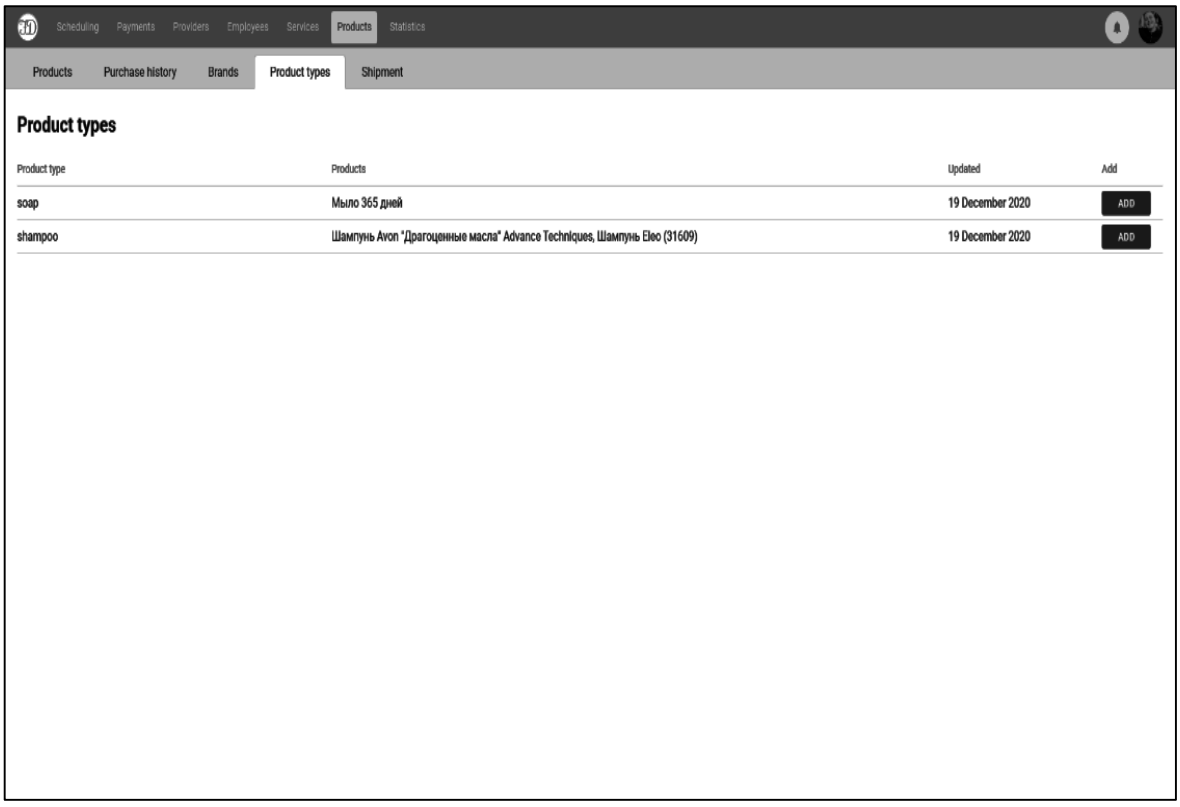

Рисунок 20 – Окно для просмотра, типов продуктов

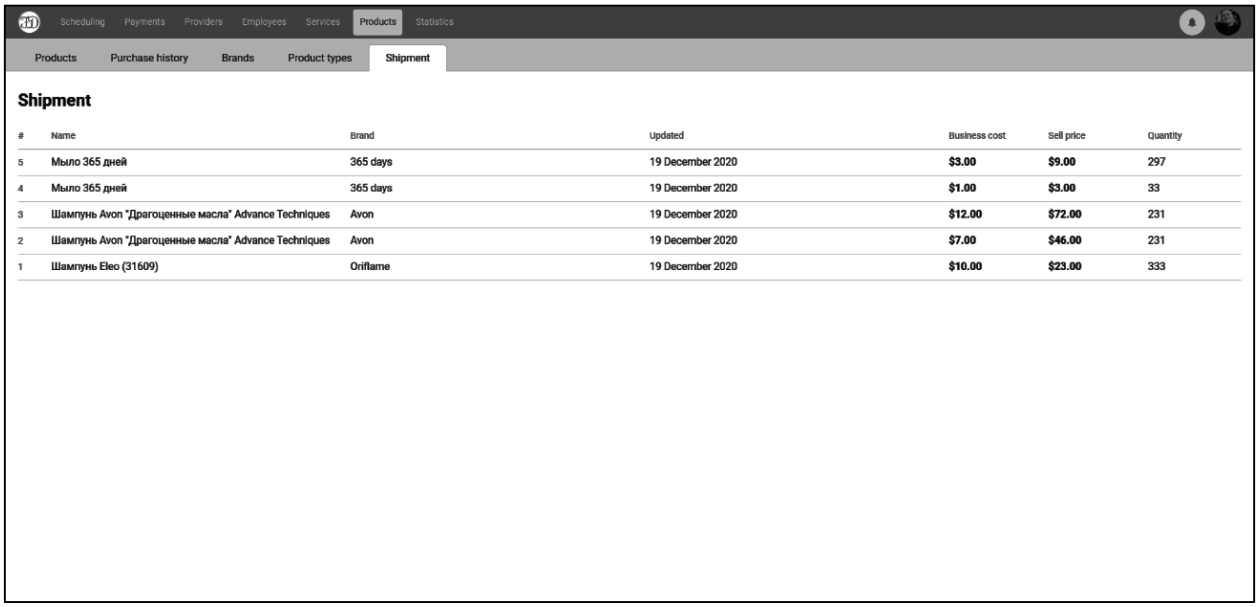

Рисунок 21 – Окно для просмотра списка поставок

Последним, но не мало важным, компонентом системы является контроль доходов и расходов предприятия, а также просмотр статистики. Окна фиксирования оплаты услуги и фиксирования оплаты заказов представлены на рисунках 22 и 23 соответственно.

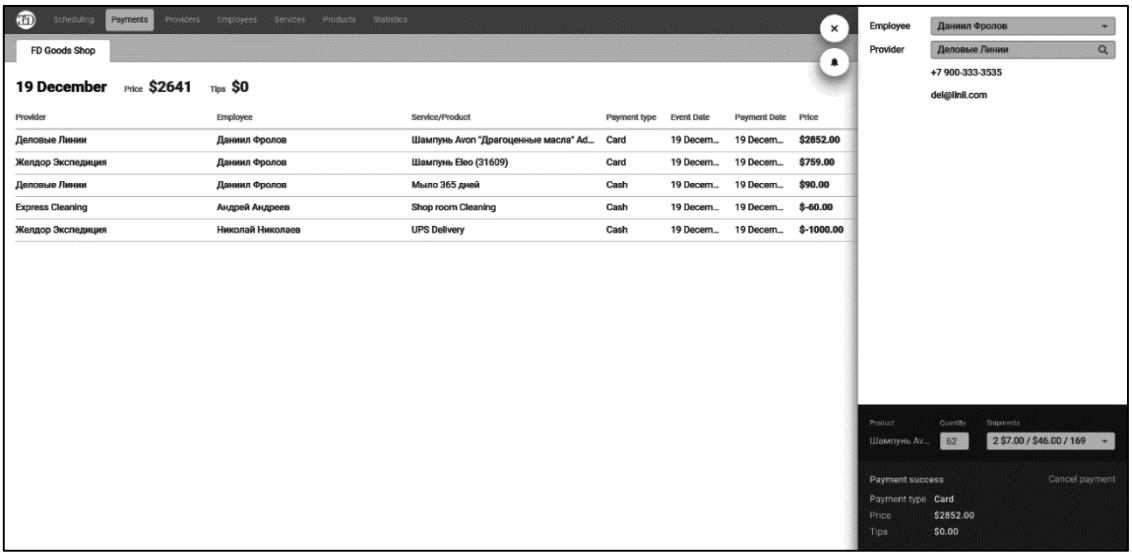

Рисунок 22 – Окно для фиксации и контроля доходов и расходов предприятия

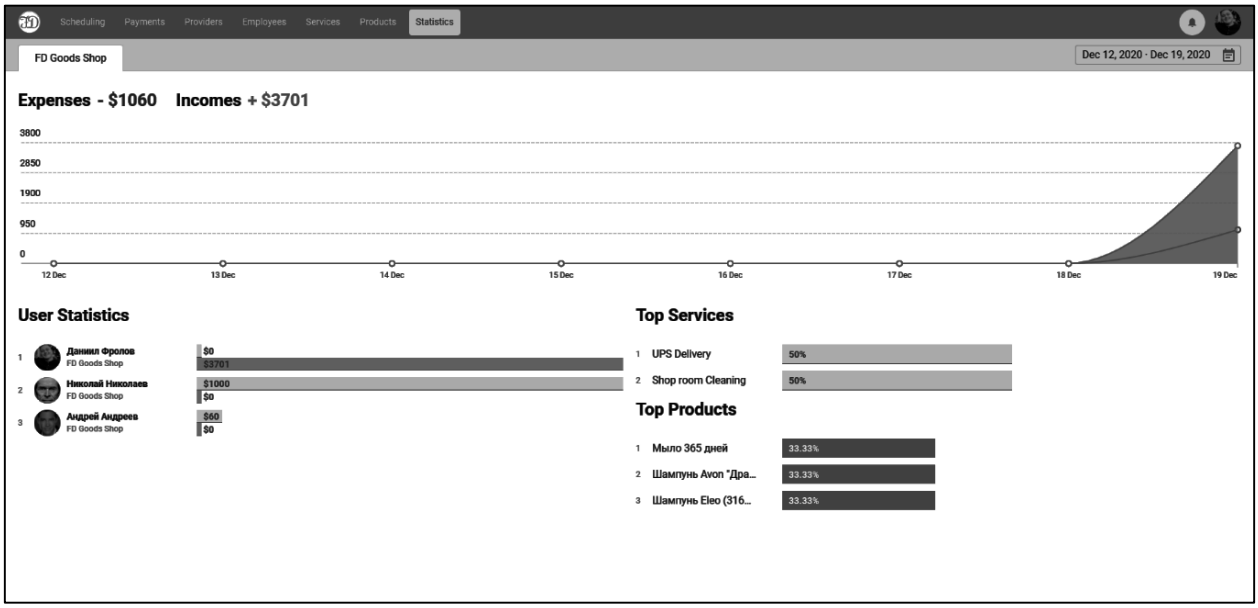

Рисунок 23 – Окно для анализа и просмотра расходов и доходов предприятия

В розничной торговле также важно оперативное реагирование на события, поэтому в системе следует предусмотреть сервис уведомлений, в котором будут фиксироваться все важные события на предприятии. Окно просмотра уведомлений представлено на рисунке 24.

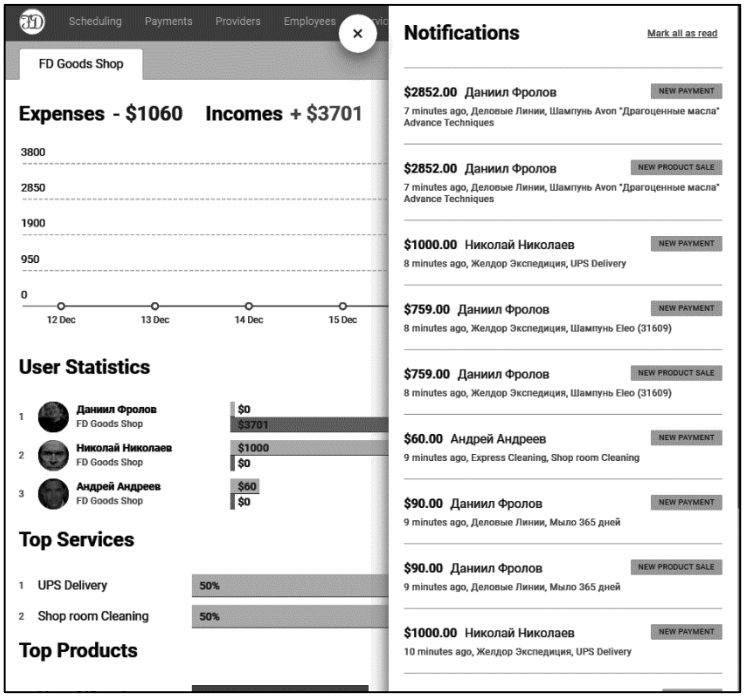

Рисунок 24 – Окно для анализа и просмотра расходов и доходов предприятия

После проработки дизайна и общего функционала системы следует определить требования для взаимодействия серверной и клиентской части приложения, а также определить требования безопасности.

3.2.2 Постановка требований к взаимодействию компонентов облачной платформы

Существуют различные способы реализации клиент-серверного взаимодействия, взаимодействия web-страница может генерироваться на сервере с динамическими данными, либо клиент может взаимодействовать с сервером посредством API.

Реализация клиент-серверного взаимодействия через API имеет ряд преимуществ перед генерацией web-страниц на сервере. В случае с API мы не ограничиваем себя одним клиентским интерфейсом, а реализуем универсальное серверное приложение, которое при желании может использоваться как web, так и мобильными клиентами, что в современных реалиях особенно актуально.

Во-вторых, мы можем реализовать внешний API для взаимодействия с нашей системой для сторонних пользователей, тем самым делая нашу систему расширяемой и поддерживающей различные интеграции в сторонние проекты.

Однако, существуют и различные способы реализации API серверного приложения – REST, GraphQL, RPC (JSON-RPC, XML-RPC, SOAP).

GraphQL - это язык запросов и манипулирования данными с открытым исходным кодом для API , а также среда выполнения для выполнения запросов с существующими данными. GraphQL был разработан внутри Facebook в 2012 году, а затем был публично выпущен в 2015 году. 7 ноября 2018 года проект GraphQL был перенесен из Facebook в недавно созданную GraphQL Foundation, размещенную некоммерческой организацией Linux Foundation.

REST (Representational State Transfer, передача состояния

представления) - это стиль проектирования архитектуры программного обеспечения, который определяет набор ограничений, которые будут использоваться для создания web-сервисов. Web-службы, соответствующие архитектурному стилю REST, называют веб-службами RESTful. RESTful системы обеспечивают взаимодействие между компьютерными системами в Интернете. Веб-службы RESTful позволяют запрашивающим системам получать доступ и управлять представлениями веб-ресурсов с помощью унифицированного и заранее определенного набора операций без сохранения состояния. RPC - протокол вызова удалённых процедур, использующий определенный формат, например XML или JSON для кодирования своих сообщений и передачи их по сети. Является прародителем SOAP, отличается исключительной простотой в применении.

SOAP (Simple Object Access Protocol) - это спецификация протокола обмена сообщениями для обмена структурированной информацией при реализации веб-служб в компьютерных сетях.

Наиболее современными и актуальными сегодня являются REST и GraphQL [16]. Не смотря на всё наиболее растущую популярность GraphQL этот стандарт всё же больше подходит для реализации внешнего API, так как имеет ряд спорных и не решённых вопросов относительно его реализации. В связи с чем, для реализации облачной платформы следует сделать выбор в сторону REST, как стандарта взаимодействия между клиентом и сервером [12]. При клиент-серверном взаимодействии огромную роль играет безопасность, так как запросы и данные отправляются по сети. Если в случае с данными нет особых проблем при использовании протокола HTTPS, то вот об идентификации пользователя и определении, что именно авторизованный пользователь выполняет запрос стоит подумать. В качестве инструмента для идентификации пользователя и предоставления доступа следует использовать JSON Web Token (JWT) протокол, так как это один из наиболее простых и безопасных протоколов взаимодействия в современных клиент-серверных приложениях.

3.3 Разработка облачной платформы для управления предприятием розничной торговли

3.3.1 Разработка серверной логики программной системы

В предыдущей главе нами была спроектирована диаграмма классов для нашей разрабатываемой системы. Воспользуемся данной диаграммой для реализации нашего серверного приложения. В качестве базы данных (БД) для хранения данных облачной платформы воспользуемся реляционной системой управления базами данных (СУБД) PostgreSQL, так как это свободная, современная, активно развивающаяся СУБД с большими возможностями и хорошей отказоустойчивостью.

В предыдущих главах мы уже определили стек технологий для реализации серверной логики – Python 3, Django, Django REST Framework. Рассмотрим пример того, как разработанная ранее диаграмма реализуется с учетом данного стека.

Django предоставляет очень удобную и функциональный ORM (Object-Relational Mapping), которая позволяет нам проектировать структуру БД без написания непосредственных SQL запросов, а путем написания классов отражающих структуру БД. ORM Django – очень мощный инструмент и помимо упрощения проектирования БД предоставляет набор инструментов для выборки и фильтрации данных, контроль миграций (изменений) структуры БД и многое другое. Такой подход очень удобен и позволяет не зависеть от конкретной СУБД, поэтому, в случае чего, мы сможем без труда перенести нашу облачную платформу на любую другую СУБД.

Чтобы создать отображение сущности БД в Django необходимо унаследовать свой класс с описанием полей и методов сущности от специального класса Model. Пример одного из класса-сущности, реализующего объктное представление сущности из БД представлен на рисунке 25.

```python
class Product(UniqueTogetherFullCleanMixin, models.Model):
    """ Product model """
    company = models.ForeignKey('Company', on_delete=models.CASCADE)
    name = models.CharField(_('Name'), max_length=120)
    description = models.TextField(_('Description'), blank=True)
    product_type = models.ForeignKey('ProductType', on_delete=models.CASCADE)
    brand = models.ForeignKey('Brand', on_delete=models.CASCADE)

    class Meta:
        verbose_name = _('Product')
        verbose_name_plural = _('Products')
        unique_together = (('company', 'id'), ('company', 'name'),)
        ordering = ('-company', '-id',)

    @cached_property
    def shipments(self):
        if self.shipment_set.all().exists():
            return self.shipment_set.all()
        else:
            return None

    # ...
```

Рисунок 25 – Пример одного из ORM представлений в разрабатываемой облачной системе (Продукт)

Детальное описание каждого модуля нашей облачной платформы заняло бы большое количество места, поэтому приведем лишь краткий обзор разработанных модулей и пакетов. Структура модулей и пакетов серверного приложения представлена на рисунке 26.

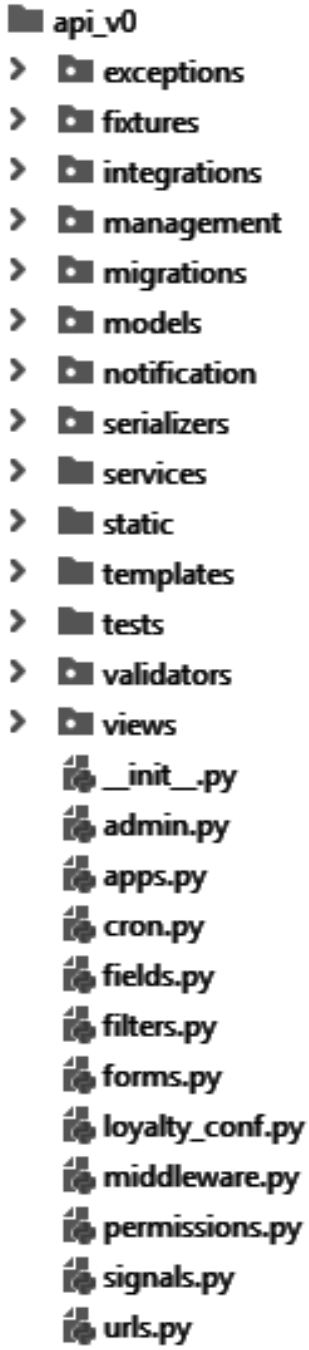

Рисунок 26 – Структура компонентов серверного приложения

В пакете «exceptions» представлены собственные исключения разработанной системы и их обработчики.

Пакет «fixtures» содержит специальные файлы для заполнения БД при тестировании.

Пакет «integrations» содержит интеграции со сторонними сервисами и

серверными приложениями. Например, сейчас там находится интеграция с сервисом Google Calendar для выгрузки расписания встреч и поставок.

Пакет «management» содержит специальные утилиты. Например, обновление времени «бесконечных» очередей событий «по запросу».

Пакет «migrations» содержит набор файлов миграций (изменений структуры БД). Это стандартный пакет для работы с версионностью и изменением структуры БД, о котором было сказано выше.

Пакет «notification» содержит сервисы для рассылки различных видов уведомлений: стандартное уведомление системы, email уведомление, sms уведомление.

Пакет «serializers» содержит сериализаторы моделей в форматы, поддерживаемые REST (JSON, XML).

Пакет «services» содержит сервисы с бизнес-логикой и реализует service layer pattern (Шаблон проектирования «уровень сервсов»).

Пакет «static» содержит статические ресурсы приложения. Это стандартный пакет для работы со статическими файлами в фреймворке Django.

Пакет «templates» содержит HTML шаблоны с генерируемой информацией и поддержкой DTL (Django Template Language), например, шаблоны писем для email рассылок. Это стандартный пакет для работы со шаблонами web-страниц.

Пакет «tests» содержит интеграционные и юнит тесты разработанного приложения.

Пакет «validators» содержит модули, отвечающие за проверку корректности получаемых от клиентов данных.

Пакет «views» содержит модули с представлениями, отвечающими за получение и обработку запросов от клиентов.

Модуль «admin» содержит настройки станартной административной панели Django.

Модуль «cron» содержит классы настройки повторяющихся задач и

использует демона (программа в UNIX системах) cron.

Модуль «fields» содержит собственные реализации дополнительных типов полей для ORM и сериализаторов.

Модуль «forms» содержит переопределение стандартных форм авторизации, а также реализации собственных форм.

Модуль «middleware» содержит собственные механизмы промежуточного программного обеспечения для перехвата запроса пользователя и выполнения дополнительных действий. Например, реализация авторизации через JWT.

Модуль «models» содержит ORM сущности, представляющие структуру БД данного приложения.

Модуль «permissions» содержит определение дополнительных механизмов разграничения доступа к различным частям системы.

Модуль «signals» содержит сигналы и обработчики событий, произошедших с данными. Например, обновление какого-либо поля или определенной модели.

Модуль «urls» содержит регистрацию конечных точек разрабатываемого API и сопоставление конечных точек с обработчиками.

После того, как серверное приложение было разработано, требуется проверить его работоспособность.

Одним из крупных преимуществ Django REST Framework является его автоматическая генерация документации и графическое представление API «из коробки». Пример графического представления API представлен на рисунках 27-29.

Рисунок 27 – Пример автоматически документируемого веб-интерфейса корневой конечной точки API ссерверного приложения

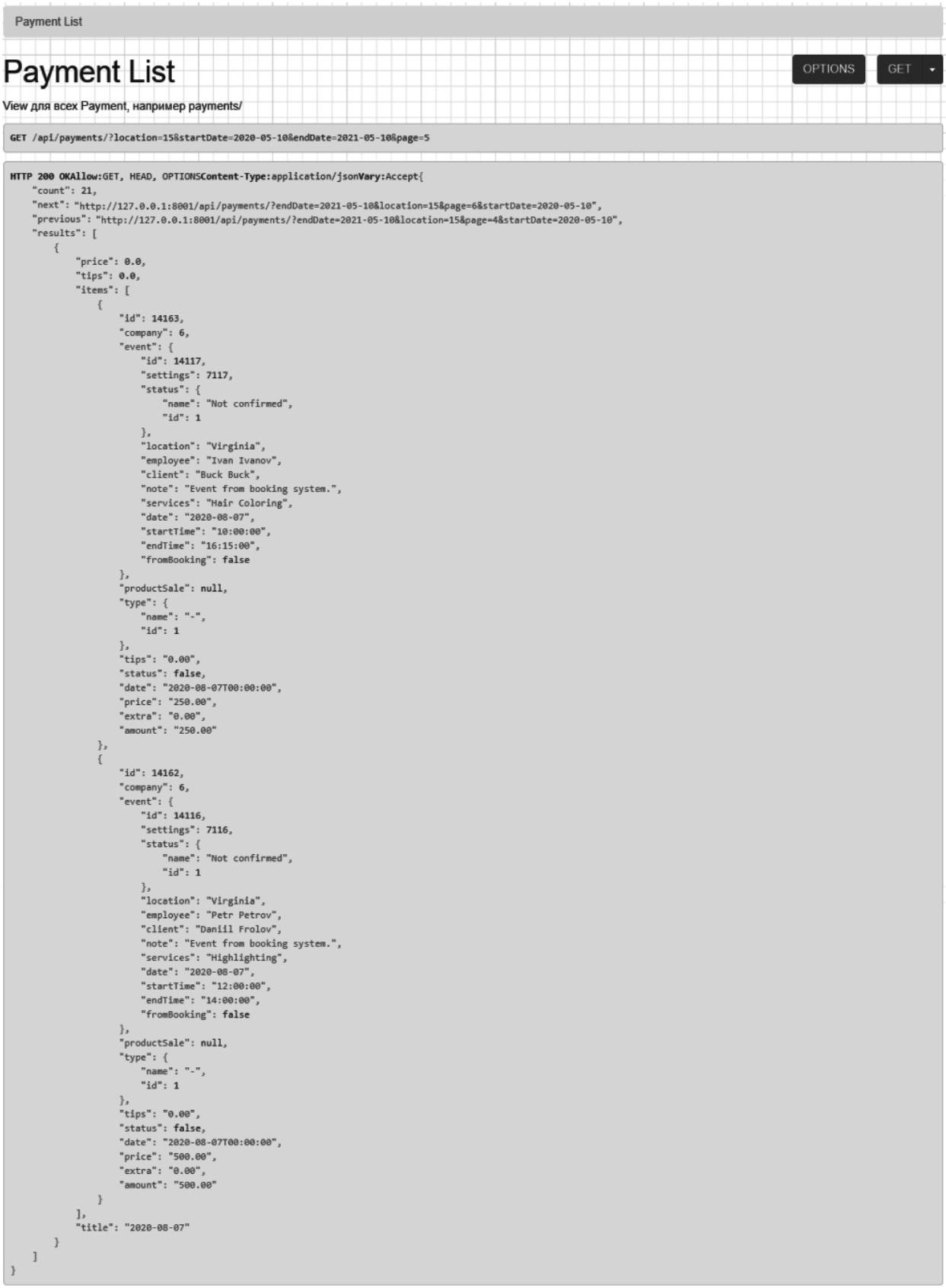

Рисунок 28 – Пример автоматически документируемого веб-интерфейса списка сущностей с пагинацией

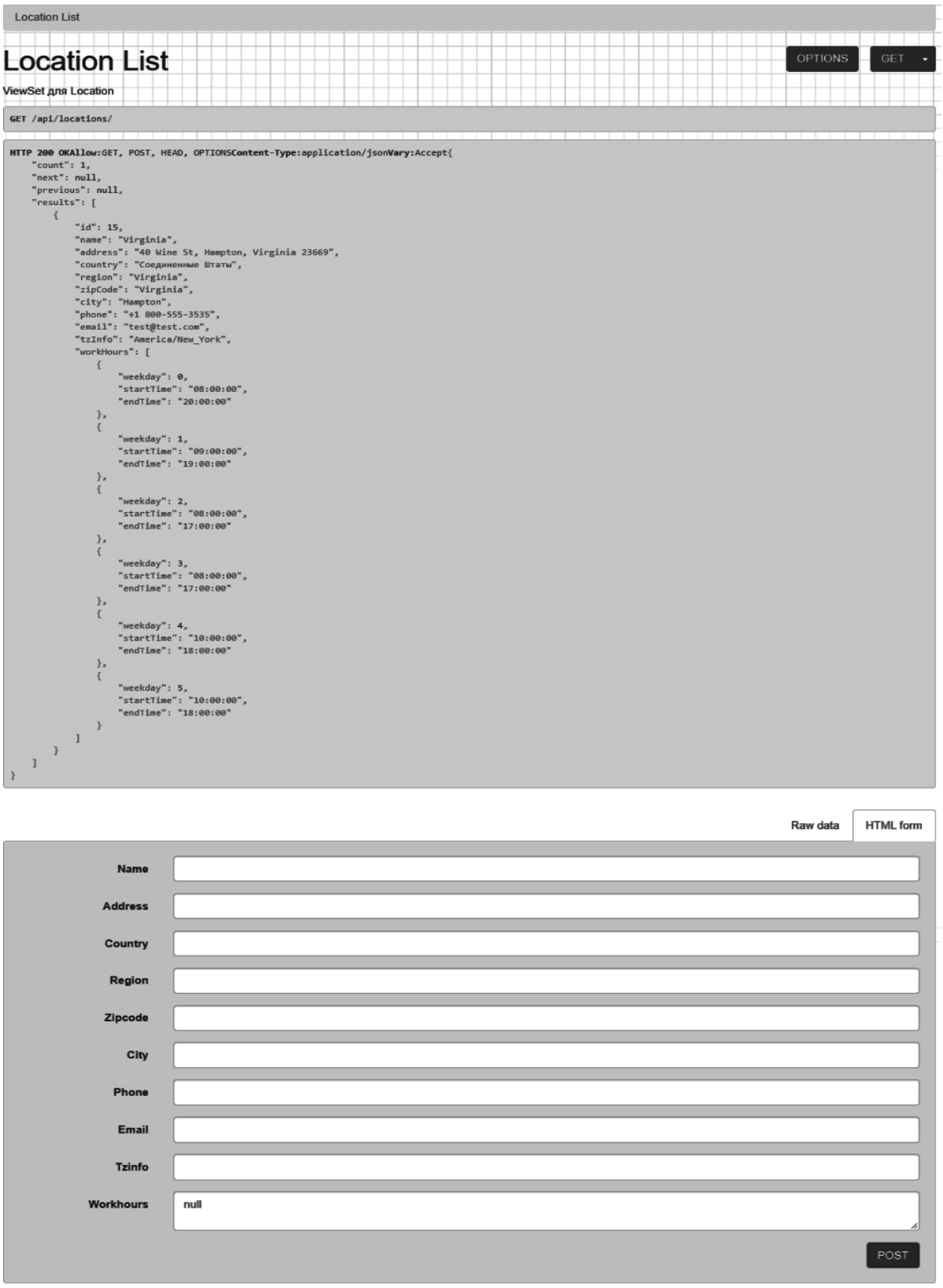

Рисунок 29 – Пример автоматически документируемого веб-интерфейса для работы с конкретной сущностью

После успешно реализованной серверной части нашей облачной платформы, реализующей все необходимые бизнес-процессы и соответствующей ранее выдвинутым требованиям, можно переходить к реализации клиентского web-приложения.

3.3.2 Разработка клиентской логики программной системы

Клиентская часть приложения не менее важна чем серверная составляющая, так как это именно то, что первым делом встретит пользователь, поэтому ошибки в клиентском приложении также недопустимы, так как они могут привести к тому, что клиент так и не сможет использовать нужный ему функционал.

Нами было разработано удобное и отзывчивое SPA приложение с использованием стека определённого ранее и полностью соответствующее выдвинутым ранее требованиям. Структура компонентов клиентского приложения представлена на рисунке 30.

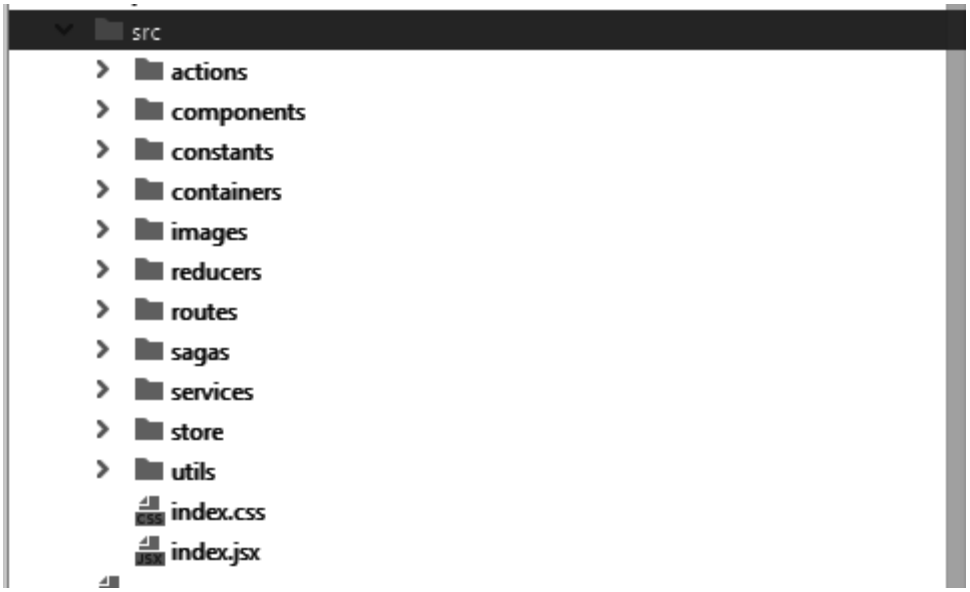

Рисунок 30 – Структура компонентов клиентского приложения

Детальное описание каждого компонента клиентского приложения в рамках данной научно-исследовательской работы нецелесообразно, поэтому приведем краткий разбор структуры компонентов клиентского приложения.

Модуль «actions» содержит набор всех actions («событий»), в котором каждый action вынесен в отдельный файл.

Модуль «components» содержит набор всех динамических компонентов приложения.

Модуль «constants» содержит постоянные данные хранимые на клиенте для ускорения и упрощения взаимодействия с серверным приложением.

Модуль «containers» содержит специализированные компоненты, которые имеют доступ к дереву состояний приложения.

Модуль «images» все изображения клиентского приложения.

Модуль «reducers» содержит специализированные компоненты отвечающие за определение состояния приложения и реакции на выполнения actions.

Модуль «routes» содержит настройки роутинга и интеграцию клиентского приложения с серверным.

Модуль «sagas» содержит реализации паттерна проектирования saga.

Модуль «services» содержит реализацию клиентской бизнес-логики.

Модуль «store» - подключенное хранилище состояний.

Модуль «utils» - пере используемые вспомогательные функции и сервисы.

Файл «index.css» основные стили проекта.

Файл «index.jsx» - точка входа клиентского приложения.

Результатом реализации клиентского приложения стала полностью законченная SaaS платформа, успешно взаимодействующая с серверным приложением и реализующая основные бизнес-процессы предприятий розничной торговли. Скриншоты разработанной SaaS платформы представлены на рисунке 31.

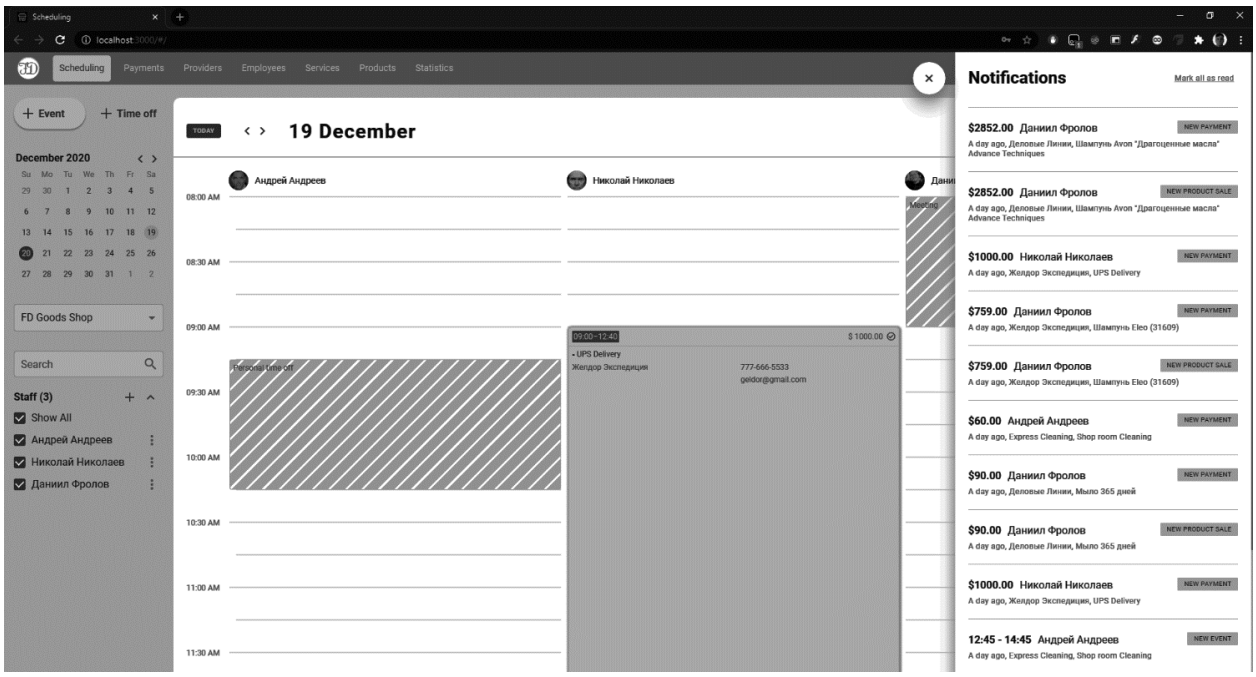

Рисунок 31 – Скриншот 2 разработанной облачной SaaS платформы для управления бизнес процессами розничной торговли

После реализации клиентской и серверной части облачной платформы логичным дальнейшим шагом будет – развертывание данной платформы на реальном сервере. Однако, следует провести сравнение разработанного решения с отечественными и иностранными аналогами, чтобы определить, какую научную и практическую ценность оно может привнести.

Выводы к главе 3

В данной главе нами была спроектирована и разработана облачная платформа, реализующая бизнес-процессы предприятий розничной торговли.

В дальнейшем, перед нами стоит задача провести более детальный анализ разработанной облачной платформы и рассмотреть вектор развития системы, а также провести сопоставление полученных результатов научно-исследовательской деятельности с заложенными гипотезами.

Глава 4 Применение облачной платформы для управления предприятиями розничной торговли

4.1 Анализ ресурсных затрат на реализацию облачной платформы

Согласно теоретической информации, представленной в предыдущих частях научно-исследовательской работы, временные и денежные затраты на внедрение облачной платформы минимальны, по сравнение с разработкой и развертыванием полноценной ERP-системы для конкретного предприятия [24]. Однако, затраты на разработку и поддержание работоспособности облачной платформы для множества предприятий – весьма затратный процесс. В связи с чем провайдеры таких платформ часто прибегают к плановой модели оплаты за использование их платформы.

При плановой модели оплаты предприятие платит за те ресурсы, которые она использует. При чем, чем более гибко устроена система, тем более рациональны траты предприятия, так как они могут платить лишь за тот функционал, который им действительно нужен в конкретный момент времени и не переплачивать за избыточный функционал коробочной ERP системы.

Рассчитаем затраты, потраченные на разработку представленной платформы.

Определим размер амортизационных отчислений. Первоначальная стоимость персонального компьютера магистранта – 70000 (семьдесят тысяч) рублей. Срок полезного использования данного ПК – 5 лет, из которых 2 года он использовался для проведения данной научно-исследовательской деятельности. Следовательно, норма амортизации будет составлять:

$$\text{Н} = \frac{1}{5} * 100\% = 20\ \%.$$

Амортизационные отчисления за год составят:

$$\text{А}_\text{г} = 70000 * 0{,}2 = 14000 \text{ руб.}$$

Итоговые амортизационные отчисления составят:

$A_м = 14000 * 2 = 28000$ руб.

Для разработки платформы использовалось лицензионное программное обеспечение от компании JetBrains, стоимость которого составляет 199$ или 15 000 рублей. Итого затраты на амортизацию ПК и программное обеспечение для разработки составляют 58 000 рублей.

Помимо амортизационных затрат для развертывания облачной платформы потребовались услуги сторонних организаций, а именно хостинг провайдера. В качестве хостинг провайдера был выбран сервис Яндекс.Облако. Причин такого выбора несколько:

– Яндекс.Облако предлагает прозрачное ценообразование.

– Возможность расширения и увеличения количества используемых ресурсов.

– Возможности получить грант на первые два месяца использования сервиса.

Срок использования данного сервиса составил 4 месяца с тарифным планом 6500 руб./мес. Данный пункт следует также отнести к постоянным расходам, так как необходим для работы платформы. К услугам сторонних организаций также можно отнести затраты на регистрацию доменного имени, которые составили 200 рублей.

Функционирование же платформы требует не менее 6517 руб./мес.

Для определения размера монетизации облачной платформы стоит также учитывать и размер заработной платы, потраченной на реализацию данного программного обеспечения. Количество рабочих дней, потраченных на разработку облачной платформы составило 79. Над проектом работали два исполнителя – научный руководитель и инженер (магистрант) в соотношении 19 и 60 дней соответственно. Оклад научного руководителя составляет 60 тыс. руб./мес., оклад инженера 40 тыс. руб./мес. Годовой фонд рабочего времени составил 247 рабочих дней.

Соответственно, затраты на заработную плату составляют:

$$З_{нр} = 60\,000 * \frac{12}{247} * 19 = 55\,384{,}6 \text{ руб.}$$

$$З_{и} = 40\,000 * \frac{12}{247} * 60 = 116\,599{,}2 \text{ руб.}$$

Итого: 171 983,8 рублей – затраты на заработную плату исполнителям.

Подытожим все затраты в таблице 2 для более наглядного представления

Таблица 2 – Затраты на разработку облачной платформы

Наименование трат	Размер затрат (руб.)
Амортизация оборудования	28 000
Программное обеспечение для разработки	30 000
Услуги сторонних организаций	26 200
Заработная плата исполнителей	171 984
Итого:	256 184

Как видно из таблицы 1, итоговые затраты на разработку облачной платформы составляют 256 184 рублей. Постоянные расходы на функционирование облачной платформы составляют 6517 рублей в месяц.

Разработанная платформа может выдерживать большие нагрузки до нескольких тысяч предприятий, пользующихся данной платформой. С учетом окупаемости затрат в течении 12 месяцев, доход платформы должен составлять 27 866 рублей в месяц. Даже при использовании платформы 10 предприятиями для достижения уровня безубыточности, размер оплаты одним предприятием составит менее 3000 рублей в месяц, что делает разработанное решение очень перспективным. Следует сравнить затраты на использование представленной платформы с локальными решениями, чтобы сделать вывод о рентабельности использования предприятием представленной платформы.

4.2 Сравнение результатов исследований и разработанной платформы с аналогами

Чтобы убедиться в том, что наша разработанная платформа удовлетворяет всем условиям рынка, а также является полезным продуктом в управлении бизнесом, следует сравнить его с отечественными и иностранными аналогами [9].

Из отечественных аналогов самыми популярными являются «Мой склад» и «Yclients». Из иностранных аналогов стоит выделить «Salesmate» Примеры интерфейса данных облачных сервисов представлены на рисунках 32-34 соответственно.

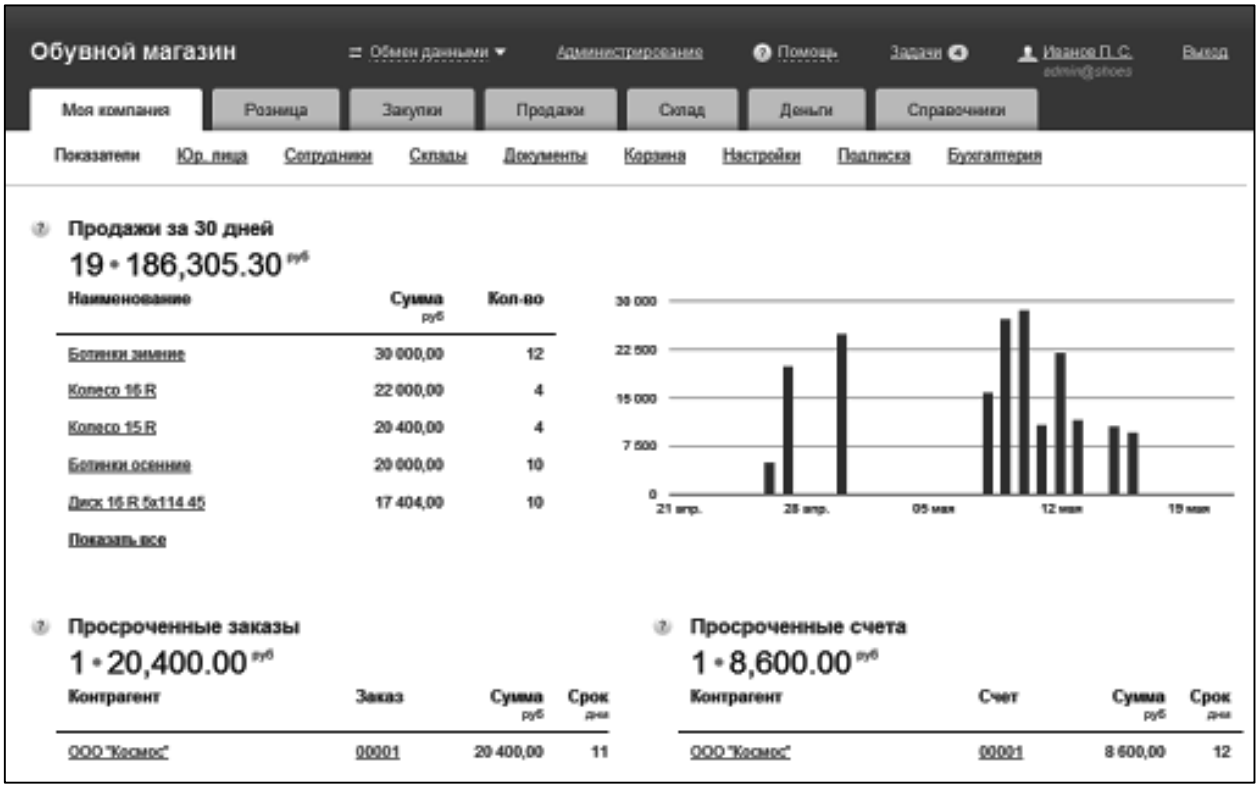

Рисунок 32 – Интерфейс облачного SaaS сервиса «Мой склад»

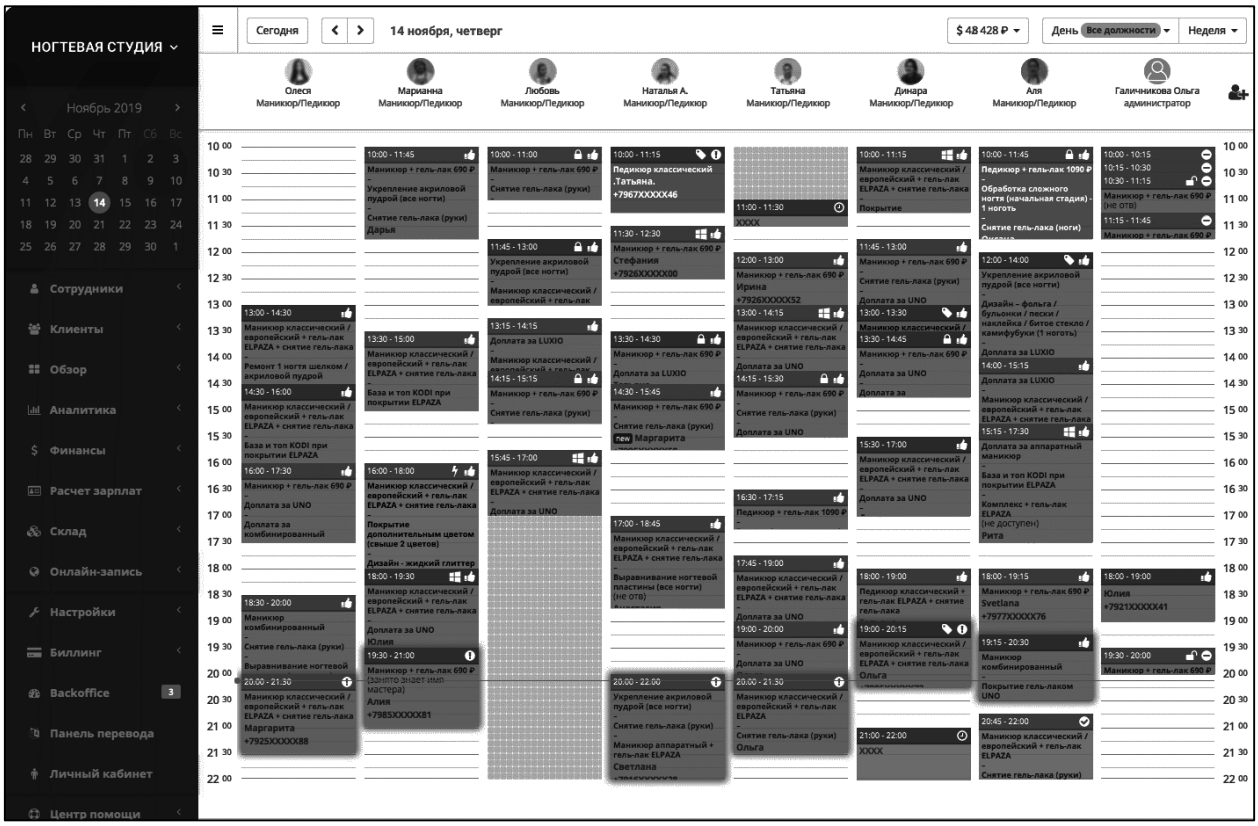

Рисунок 33 – Интерфейс облачного SaaS сервиса «Yclients»

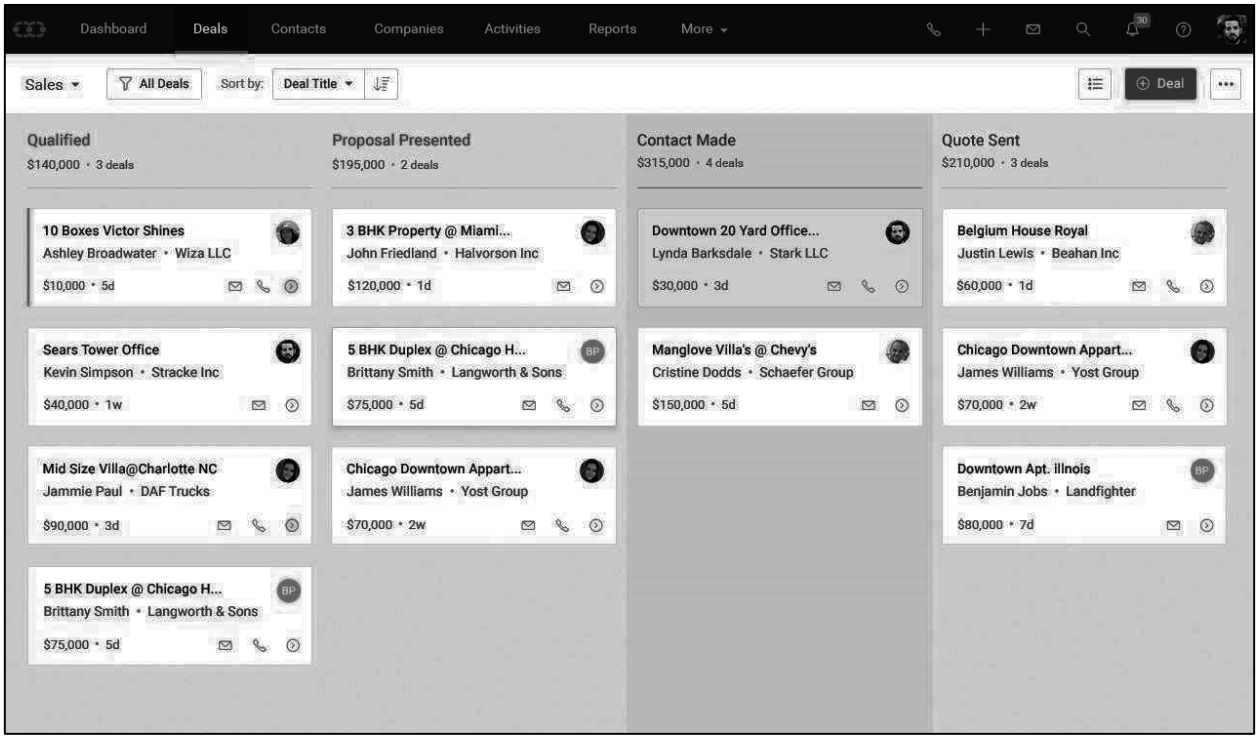

Рисунок 34 – Интерфейс облачного SaaS сервиса «Salesmate»

После изучения конкурентов и их функционала, мы сделали вывод, что

функционал их во многом похож и также реализован в нашей облачной платформе. Однако, в случае с сервисом «Мой склад» мы упираемся в ограниченность системы и упором исключительно на складской учёт без дополнительных важных модулей для планирования деятельности предприятия, занимающегося розничной торговлей. В случае с «Yclients», напротив, представлен слишком широкий функционал и данная система стремиться быть универсальной как для предприятий сферы услуг, так и для предприятий занимающихся розничной торговлей в связи с чем вынужденно идти на компромиссы в плане функционала и не предоставляет выбора и настройки только нужных компонентов. Сервиса «Salesmate» во-первых, ориентирован на ранок Европы и США и поэтому не учитывает специфику российского рынка, а во-вторых, сервис нацелен на предприятия среднего и малого бизнеса.

Во сравнении со представленными конкурентами мы можем сделать вывод, что разработанное нами решение может занять свою нишу в сфере облачных сервисов для управления предприятием розничной торговли, однако стоит более детально продумать бизнес стратегию развития нашей платформы, определить узкие места конкурентов и реализовать их в нашей системе, чтобы помочь предприятиям, занимающимся розничной торговлей повысить качество своей деятельности.

Популярным программным решением для управления предприятием сегодня является программное обеспечение от компании «1С». Для управления предприятиями розничной торговли у данной компании есть специальная редакция их программного обеспечения «1С: Предприятие 8» - «1С: Управление торговлей 8».

Сравним стоимость затрат на внедрение «1С: Управление торговлей 8» со стоимостью использования разработанной платформы.

Для внедрения «1С: Управление торговлей 8» потребуется покупка соответствующих лицензий. Представим стоимость лицензий для внедрения «1С: Управление торговлей 8» на 20 пользователей в таблице 3.

Таблица 3 – Расчет затрат на внедрение «1С: Управление торговлей 8» на 20 пользователей

Наименование ПО	Количество лицензий	Стоимость (руб.)	Итого (руб.)
Windows Server 2012 R2 standard	2	36 000	72 000
Windows Server 2012 User CAL	20	1400	28 000
SQL CAL 2014 runtime	20	8600	172 000
RDS CAL 2012	20	4800	96 000
Microsoft SQL 2014 Standard Runtime	1	20 000	20 000
Сервер 1С Предприятие x64	1	86 000	86 000
Лицензия 1С: Управление торговлей 8	20	7200	72 000
Итого:	546 000 руб.		

Стоимость серверов согласно источнику [1] в среднем составит 300 000 рублей, а стоимость работ по внедрению в районе 100 000 рублей.

Итого, средние затраты на внедрение «1С: Управление торговлей 8» с лицензией на 20 человек составят 946 000 рублей.

Помимо прочего присутствуют также и постоянные затраты на поддержание созданной инфраструктуры.

Даже при условии, что разработанной в рамках нашей научно-исследовательской деятельности платформой будет пользоваться только одно предприятие на 20 человек, срок за который предприятие сможет окупить затраты на внедрение «1С: Управление торговлей 8» составит $\frac{946\,000}{27\,866} \approx$ 34 месяца.

При более продуманной монетизации и при большем количестве предприятий, пользующихся услугами облачной платформы окупаемость внедрения коробочного решения по типу «1С: Управление торговлей 8» составит нецелесообразный срок, при этом использование облачной SaaS

платформы избавит предприятие от множества других не менее важных проблем, рассматриваемых в прошлых частях данной научно-исследовательской работы. Также при внедрении решения по типу «1С: Управление торговлей 8» потребуются затраты на содержание оборудование, оплату ежемесячного сопровождения, что ставит под сомнение сам факт выгодности такого подхода.

Таким образом, представленные выше расчёты и сравнения подтверждают, что использование разработанной облачной платформы для управления бизнес-процессами предприятия розничной торговли не только удобнее для предприятия, но и выгоднее.

4.3 Интеграция предприятий розничной торговли в разработанную облачную платформу

После реализации нашей платформы мы провели тестирование нашей облачной SaaS платформы на специальной фокус-группе, состоящей из предприятий города Тольятти, занимающихся розничной торговлей.

Мы связались с представителями предприятий: ООО Торговый Дом «Спецавтопласт», ООО «Рыболов Альянс», ООО «ВПМ», ООО «Барс», ООО «Звезда», ООО «Новые мебельные технологии», и предложили им использовать нашу облачную платформу в качестве инструмента для управления своими бизнес-процессами. Стоит отметить, что все представленные предприятия занимаются розничной торговлей, но при этом, в совершенно различных направлениях: производство и продажа защитной пленки, производство и продажа мебели, розничная торговля металлом, розничная торговля рыбной продукцией, производство и продажа пластиковых изделий.

Мы попросили данную фокус-группу оценить каждое окно и инструмент интерфейса нашей облачной продукции от 1 до 10, где 10 - полностью удовлетворяет, 1 - совершенно не удовлетворяет. Результаты оценивания

представлены в таблице 4.

Таблица 4 – Результаты оценивания разработанной SaaS платформы фокус-группой

Окно интерфейса	ООО Торговый Дом «Спецавто пласт»	ООО «Рыболов Альянс»	ООО «ВПМ»	ООО «Барс»	ООО «Звезда»	ООО «Новые мебельные технологии»
Scheduling (Планирование расписания и поставок)	9	9	5	10	10	8
Payments (Фиксация оплат и их список за определенный период)	7	8	9	9	9	10
Providers (Поставщики услуг)	6	8	7	7	8	9
Employees (Учёт сотрудников)	10	10	10	10	10	10
Services (Услуги)	6	6	5	7	7	4
Products (Продукты)	10	10	7	10	10	10
Statistics (Статистика)	7	6	7	6	8	7
Система уведомлений	9	8	9	9	9	8
Общее впечатление о платформе	8	8	7	9	9	8

С целью более наглядного представления данных и выявления проблемных мест нашей облачной платформы, по полученным данным мы составили графики. Графики представлены на рисунках 35-36.

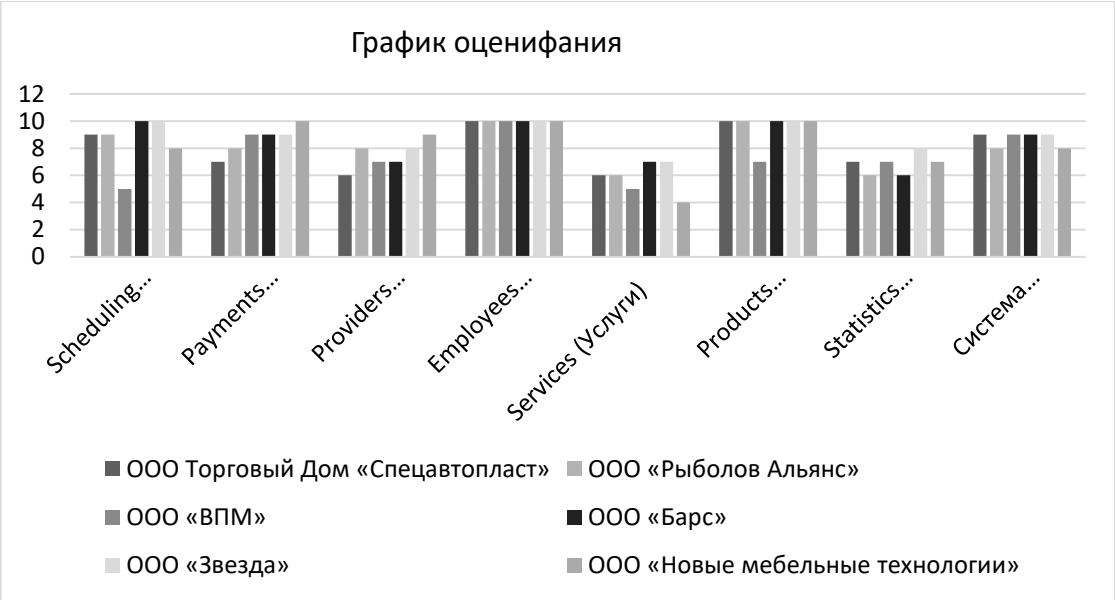

Рисунок 35 – График оценивания разработанного программного обеспечения

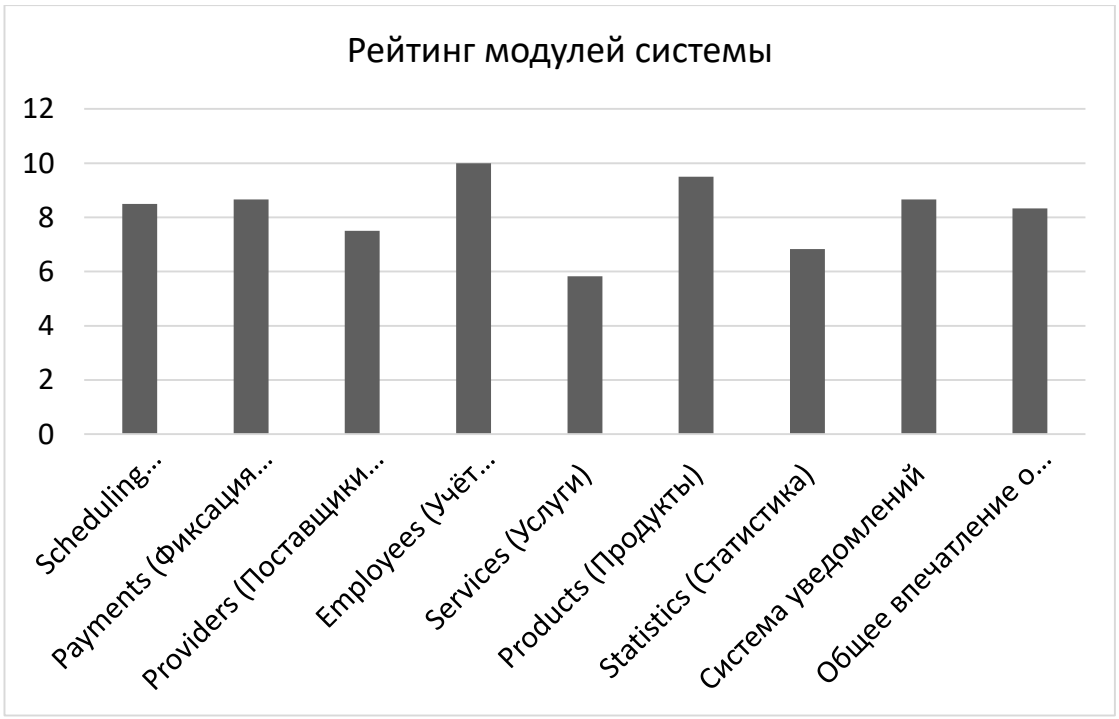

Рисунок 36 – График с рейтингом модулей разработанного программного обеспечения

Как видно из представленной таблицы и графиков, все участники фокус-группы оказались в основном довольны опытом использования нашей

облачной платформы и многие лично высказались о том, что системы удобная и они бы не отказались от использования подобной программы в повседневной деятельности. Однако, по графикам также видно, что у большинства компаний возникли трудности с некоторыми элементами интерфейса, больше всего с окном управления услугами.

Мы учли все пожелания фокус-группы и будем работать над улучшениями нашей облачной платформы.

Выводы к главе 4

В данной главе нами были подсчитаны ресурсные затраты, потраченные на разработку облачной платформы, а также произведена интеграция ряда предприятий в разработанную облачную платформу.

Мы провели опрос интегрированных в нашу облачную платформу предприятий и выяснили, что в основном, предприятия были удовлетворены опытом использования разработанной нами облачной платформы.

Нами было выяснено, что облачные технологии – будущее развитие программного обеспечения и что предприятия розничной торговли получат огромную пользу при отказе от локальных решений и переходе в облака.

Конечно, разработанная SaaS платформа может не покрывать все бизнес-процессы конкретно взятого предприятия, однако, в разработанной нами программной системе имеется возможность расширение и доработка дополнительных модулей.

Заключение

Предприятия различного уровня и размера пытаются увеличить свою прибыль путем внедрения информационных технологий в свои бизнес процессы. В тяжелых условиях конкурентной борьбы каждый предприниматель стремиться наименьшими усилиями и затратами увеличить свою прибыль, но, как правило, различные интегрированные ERP, CRM системы – не всегда лучший вариант и помимо больших затрат, они требуют периодического вмешательства специалистов поддержки и разработчиков для поддержания системы в актуальном для бизнеса состоянии. В решении данного вопроса на помощь бизнесу приходят облачные вычисления.

Облачные вычисления – способ обеспечения удобного сетевого доступа по запросу к определенному общему набору вычислительных ресурсов и программному обеспечению. Потребители облачных вычислений могут снизить свои расходы, связанные с реализацией инфраструктуры информационных технологий.

В рамках научно-исследовательской деятельности было рассмотрено использование облачных вычислений для управления бизнес-процессами предприятий розничной торговли. Были выделены и охарактеризованы следующие преимущества облачных технологий:

– Масштабируемость.

– Экономичность.

– Немедленная доступность.

– Представление.

– Безопасность.

В ходе рассмотрения типов облачных приложений, таких как IaaS, PaaS, SaaS было выяснено, что для предприятий розничной торговли больше всего подходят SaaS решения, ввиду следующих преимуществ:

– Возможность использования с любого устройства с доступом в интернет.

– Автоматическое обновление программного обеспечения.
– Бюджетность.
– Простота в использовании.
– Отсутствие необходимости контроля над оборудованием.

После того, как нами был выбран оптимальный тип облачного сервиса, мы связались с представителями предприятий: ООО Торговый Дом «Спецавтопласт», ООО «Рыболов Альянс», ООО «ВПМ», ООО «Барс», ООО «Звезда», ООО «Новые мебельные технологии», и определили основные бизнес-процессы, которые следует предусмотреть в разрабатываемой программной системе.

Собрав необходимые сведенья от предприятий, мы приступили к проектированию и разработке SaaS платформы. Были определены лучшие практики разработки современных облачных решений, которые были отражены в публикациях, и спроектированы вспомогательные диаграммы.

Нами была разработана модульная SaaS платформа, которая реализует бизнес-процессы предприятий розничной торговли. Данная платформа может включать и отключать модули с реализацией определенных бизнес-процессов «по требованию» и расширяться, посредством добавления новых модулей.

В завершении работы мы подсчитали ресурсные затраты, потраченные на разработку программного обеспечения, подтвердили свои положения, заложенные в начале научно-исследовательской деятельности.

Подводя итоги научно-исследовательской деятельности, можно сделать вывод, что выдвинутые нами тезисы и положения подтвердились.

Список используемой литературы и используемых источников

1. 1С: Предприятие 8 [Электронный ресурс]. URL: https://v8.1c.ru/price/ (дата обращения: 25.05.2021).

2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем [Электронный ресурс]. URL: https://docs.cntd.ru/document/9041994 (дата обращения: 25.05.2021).

3. Зиангирова Л. Ф. Технологии облачных вычислений : учебное пособие. Саратов : Вузовское образование, 2016. 300 с.

4. Как посчитать выгоды от миграции в «облако» [Электронный ресурс]. URL: https://www.pvsm.ru/it-infrastruktura/119475 (дата обращения: 25.05.2021).

5. Моделирование программных систем [Электронный ресурс]. URL: http://www.informicus.ru/default.aspx?SECTION=6&id=73&subdivisionid=1 (дата обращения: 25.05.2021).

6. Петров Д.Л. Алгоритмы миграции данных в высокомасштабируемых облачных системах хранения // Автореферат диссертации на соискание ученой степени к.т.н. СПб: СПбГЭТУ «ЛЭТИ», 2011. 18 с.

7. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. СПб.: Питер, 2021. 544 с.

8. Сафонов В. О. Платформа облачных вычислений Microsoft Windows Azure : учебное пособие. Москва, Саратов : ИНТУИТ, Ай Пи Ар Медиа, 2020. 329 с.

9. Ahmad A. A., Andras P. Scalability analysis comparisons of cloud-based software services, Advances in Intelligent Systems and Computing. 2019. Vol. 8. PP. 115-128.

10. Andy M., Jon P., Peter F. Enterprise Cloud Computing: A Strategy Guide for Business and Technology Leaders - and the Rest of Us, Meghan-Kiffer Press, 2010. 264 p.

11. Christopher M. Cloud Computing Law, Oxford University Press, 2013. 448 p.

12. Codecademy: What is REST? [Электронный ресурс]. URL: https://www.codecademy.com/articles/what-is-rest (дата обращения: 29.10.2020).

13. Dataversity: SaaS vs. PaaS vs. IaaS: Where the Market is Going? [Электронный ресурс]. URL: https://www.dataversity.net/saas-vs-paas-vs-iaas-where-the-market-is-going/ (дата обращения: 25.05.2021).

14. Django project: Django documentation [Электронный ресурс]. URL: https://docs.djangoproject.com/en/3.1/ (дата обращения: 25.05.2021).

15. Gemino A., Parker D., Use case diagrams in support of use case modeling: Deriving understanding from the picture, Journal of Database Management, 2009. 427 p.

16. GraphQL: Introduction to GraphQL [Электронный ресурс]. URL: https://graphql.org/learn/ (дата обращения: 25.05.2021).

17. Gustavo A. A. Santana, Data Center Virtualization Fundamentals: Understanding Techniques and Designs for Highly Efficient Data Centers with Cisco Nexus, UCS, MDS, and Beyond, Cisco Press, 2013. 960 p.

18. Igor F., Hui-Lan Lu, Dor S., Cloud Computing: Business Trends and Technologies, Wiley, 2016. 376 p.

19. Jessica G., Infrastructure as a Service 36 Success Secrets - 36 Most Asked Questions on Infrastructure as a Service - What You Need to Know, Emereo Publishing, 2013. 78 p.

20. Kris J., Cloud computing: SaaS, PaaS, IaaS, virtualization, business models, mobile, security and more, Burlington, MA: Jones & Bartlett Learning, 2013. 321 p.

21. Mahon E. Transitioning the Enterprise to the Cloud: A Business Approach Kindle Edition, Cloudworks Publishing Company, 2015. 179 p.

22. Matthew P. Virtualization Essentials, 2nd Edition, Sybex; 2016. 336 p.

23. Michael J. Kavis, Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS), Wiley, 2014. 229 p.

24. Oduh I. U., Misra S., Damasevicius R., Maskeliunas R. Cloud Based Simple Employee Management Information System: A Model for African Small and

Medium Enterprises, Journal of Cloud Computing. 2018. Vol. 8. PP. 1-17.

25. Python Documentation: PEP 3333 - Python Web Server Gateway Interface v1.0.1 [Электронный ресурс]. URL: https://www.python.org/dev/peps/pep-3333/ (дата обращения: 25.05.2021).

26. Ray J Rafaels, Cloud Computing: From Beginning to End, CreateSpace Independent Publishing Platform, 2015. 152 p.

27. Seema P., N., Subash C., SaaS CloudQual: A Quality Model for Evaluating Software as a Service on the Cloud Computing Environment, Innovations in Computer Science and Engineering. 2017. Vol. 8. PP. 73-80.

28. Thomas E., Ricardo P., Zaigham M., Cloud Computing: Concepts, Technology & Architecture, Prentice Hall, 2013. 487 p.

29. Vladimir S., Trustworthy Cloud Computing, John Wiley & Sons, 2016. 352 p.

30. WSGI Documentation: Learn about WSGI [Электронный ресурс]. URL: https://wsgi.readthedocs.io/en/latest/learn.html (дата обращения: 25.05.2021).