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

Кафедра Прикладная математика и информатика
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Управление корпоративными информационными процессами
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Моделирование и разработка сервис-ориентированных приложений»

Обучающийся А. Ю. Макаров
(И.О. Фамилия) (личная подпись)

Научный руководитель к.п.н., О. Ю. Копша
(ученая степень, звание, И.О. Фамилия)

Тольятти 2024

Оглавление

Введение

В современных условиях развития информационных технологий предприятия сталкиваются с проблемой необходимости быстрой адаптации информационных систем к постоянно меняющимся бизнес-требованиям.

Одной из ключевых научно-практических проблем является оперативное и экономически эффективное внедрение ИТ-решений, интегрирующих компоненты, разработанные на основе технологий различных поставщиков. Для решения данной задачи требуется обеспечение технической совместимости, стандартизация интерфейсов и протоколов взаимодействия, а также соблюдение принципов экономической целесообразности при разработке и реализации таких решений. В этом контексте сервис-ориентированная архитектура предоставляет концептуальную основу, позволяющую адресовать вышеуказанные вызовы и способствовать эффективной интеграции разнообразных ИТ-компонентов в единую систему.

В рамках данного исследования будет рассмотрено разработка эффективных методов моделирования и реализации сервис-ориентированных приложений, обеспечивающих высокую гибкость, масштабируемость и интегрируемость систем. Актуальность проблемы обусловлена потребностью в создании гибких, масштабируемых и легко интегрируемых систем

Основные задачи исследования включают следующие пункты:

– проанализировать теоретические аспекты методов и способов моделирования Формирование модели оптимизации процессов управления;

– сформировать модель сервис-ориентированного приложения для оптимизации процессов управления в системе управления заказов при помощи декомпозиции и моделирования бизнес-процессов при помощи BPMN, UML, диаграммы классов и модели структуры базы данных;

– разработать модель сервис-ориентированного приложения, предназначенного для сервиса управления заказами;

– оценить эффективность реализованного приложения и перспективы развития.

Объект исследования – являются сервис-ориентированные архитектуры и подходы к разработке приложений в контексте современных программных систем.

Предметом исследования выступают методы и технологии моделирования и разработки сервис-ориентированных приложений, а также принципы их внедрения и оценки эффективности.

Гипотеза исследования заключается в том если применять современные методы моделирования и разработки при создании сервис-ориентированных приложений, то это позволит повысить их гибкость, масштабируемость и эффективность интеграции в существующие информационные системы.

Положения, выносимые на защиту:

Теоретический анализ сервис-ориентированных архитектур (SOA) подтверждает их значительные преимущества в сравнении с традиционными монолитными архитектурами, включая улучшенную гибкость, масштабируемость и облегчённую интеграцию разнородных систем в современных информационных системах. Обоснование исследования в том, что были рассмотрены основные принципы сервис-ориентированной архитектуры, такие как разделение ответственности, стандартизация интерфейсов и независимость сервисов. Анализ существующей литературы и сравнительный анализ показали, что сервис-ориентированной архитектура обеспечивает более эффективное управление сложностью систем, позволяет быстрее адаптироваться к изменениям бизнес-требований и способствует повторному использованию компонентов.

Применение современных методов и инструментов моделирования, таких как UML и BPMN, в сочетании с микро-сервисными подходами, обеспечивает эффективное проектирование и разбиение сервис-ориентированных приложений, что способствует повышению их интегрируемости и адаптивности к изменениям бизнес-процессов. Исследование различных методов

моделирования выявило, что использование UML и BPMN позволяет создавать чёткие и понятные модели бизнес-процессов и архитектурных решений. Сравнение монолитных и сервис-ориентированных архитектур показало, что сервис-ориентированный подход предоставляет более гибкую и масштабируемую структуру, что подтверждается успешным применением данных методов в практике разработки.

Разработка и реализация прототипа сервис-ориентированного приложения на основе выбранных инструментов и технологий демонстрируют практическую применимость предложенных методов моделирования и подтверждает их эффективность в создании гибких и масштабируемых информационных систем. В ходе разработки прототипа были использованы современные фреймворки и инструменты, обеспечивающие контейнеризацию и оркестрацию сервисов. Реализация показала, что предложенные методы позволяют создавать модули, легко масштабируемые и управляемые, что подтверждается успешной интеграцией сервисов и их взаимодействием в рамках сервис-ориентированной архитектуры.

Было проведено тестирование, а также анализ перспектив развития сервис-ориентированного подхода, подтверждают высокую эффективность предложенных решений и их потенциал для дальнейшего развития в условиях роста требований к информационным системам. Оценка производительности с использованием инструментов нагрузочного тестирования и мониторинга показала, что разработанное приложение обладает высокой скоростью обработки запросов и устойчивостью к нагрузкам. Анализ успешных кейсов внедрения сервис-ориентированной архитектуры в различных отраслях и рассмотрение современных тенденций (контейнеризация, без-серверные архитектуры) свидетельствуют о перспективности выбранного направления и необходимости дальнейших исследований для улучшения и расширения функциональности сервис-ориентированных приложений.

Научная новизна исследования заключается в разработке и обосновании новых методов моделирования и разработки сервис-ориентированных

приложений, учитывающих современные тенденции микро-сервисной архитектуры и контейнеризации. Предложенные подходы способствуют повышению эффективности разработки и эксплуатации программных систем.

Теоретическая значимость работы выражается в проведении сравнительного анализа сервис-ориентированной архитектуры с традиционными монолитными архитектурами, где показаны преимущества сервис-ориентированной архитектуры в обеспечении управляемости, адаптивности и повторного использования компонентов. На основе микро-сервисных подходов и стандартов моделирования, таких как UML и BPMN, предложены методы, способствующие улучшению проектирования и адаптации сервисов к изменениям бизнес-процессов.

Практическая значимость работы заключается в возможности применения предложенных методов и технологий для создания гибких и масштабируемых систем, что особенно актуально в условиях растущих требований к информационным системам.

Глава 1 Теоретические аспекты сервис-ориентированных архитектур

1.1 Основные принципы сервис-ориентированной архитектуры

В условиях динамично изменяющейся бизнес-среды современные предприятия предъявляют высокие требования к гибкости и адаптивности своих информационных систем. Адаптивность бизнеса достигается через непрерывное совершенствование и модификацию ИТ-инфраструктуры, что позволяет организациям быстро реагировать на изменения рыночных условий, внедрять инновации и поддерживать конкурентоспособность. Одной из ключевых научно-практических проблем является быстрое и экономически эффективное внедрение ИТ-решений, которые интегрируют компоненты, созданные на основе технологий различных поставщиков. Это требует обеспечения технической совместимости, стандартизации интерфейсов и протоколов взаимодействия, а также соблюдения экономической целесообразности при разработке и внедрении таких решений. Сервис-ориентированная архитектура (SOA, *Service-Oriented Architecture*) предоставляет концептуальную основу для решения данной проблемы, сервис-ориентированная архитектура представляет собой архитектурный стиль, который способствует разработке гибких и масштабируемых ИТ-систем посредством использования сервисов как основных строительных блоков [1]. Этот подход соответствует концепции бизнес стабильности, так как обеспечивает возможность быстрой трансформации бизнес-решений и адаптации к новым требованиям. Использование сервисов в рамках сервис-ориентированной архитектуры позволяет предприятиям осуществлять оперативную реструктуризацию и реорганизацию своих бизнес-процессов. Сервисы разрабатываются как слабосвязанные, автономные компоненты, которые могут быть повторно использованы в различных бизнес-контекстах. Это способствует снижению затрат на разработку и поддержку ИТ-систем, повышению эффективности процессов и ускорению вывода новых продуктов и

услуг на рынок. На рисунке 1 изображена базовая структура сервис-ориентированной архитектуры.

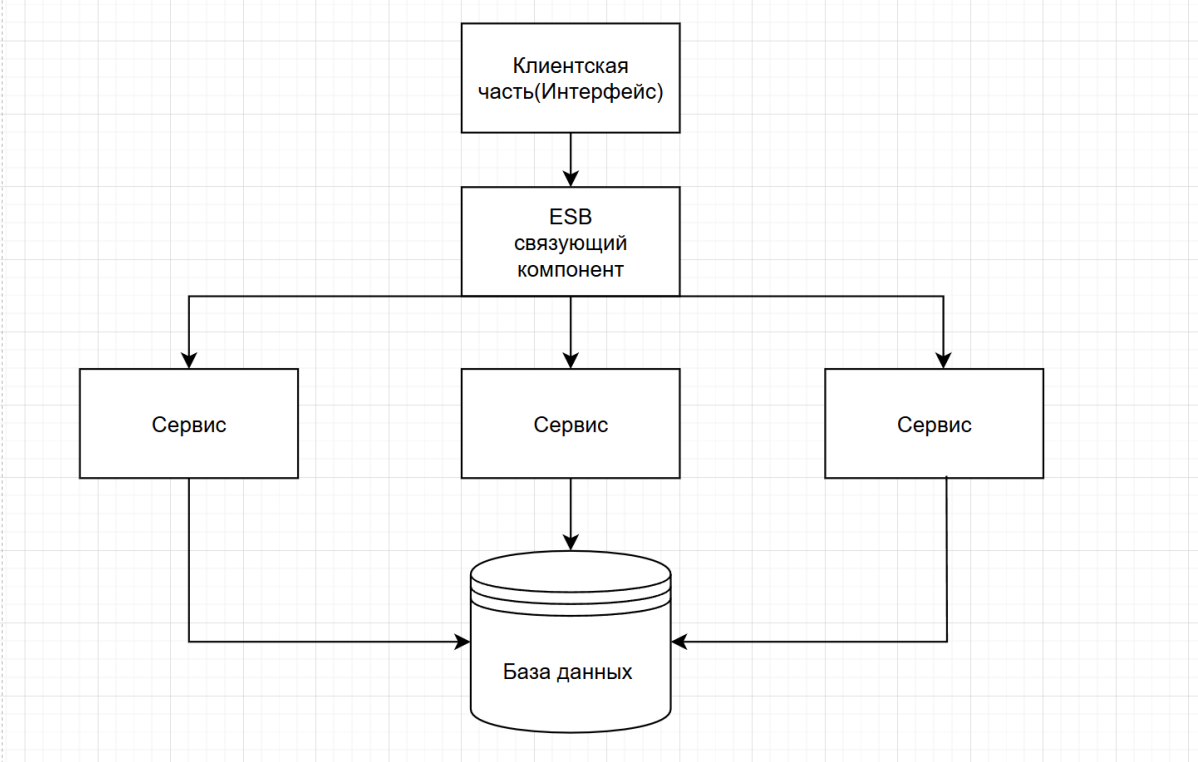

Рисунок 1 – Базовая структура сервис-ориентированной архитектуры

Сервисы взаимодействуют между собой через независимые, стандартизованные интерфейсы, обеспечивая высокую степень кроссплатформенности между различными компонентами системы. Применение открытых стандартов и протоколов, таких как SOAP, REST, WSDL, XML и JSON, позволяет интегрировать компоненты, созданные на основе технологий разных поставщиков, и создавать распределенные бизнес-приложения, ориентированные на сервисы. Это способствует масштабируемости систем, позволяя наращивать или сокращать бизнес-функции и ИТ-системы в соответствии с потребностями предприятия [4].

Основные характеристики сервисной ориентации бизнеса включают:

– бизнес-ориентированность сервисов: Сервисы разрабатываются исходя из требований и целей бизнеса, фокусируясь на поддержке и улучшении бизнес-процессов, а не ограничиваясь возможностями существующих ИТ-систем [2];

– слабая связность и повторное использование: Сервисы являются независимыми компонентами, которые могут быть повторно использованы в различных бизнес-процессах как внутри предприятия, так и между организациями, повышая эффективность и снижая избыточность[5];

– абстракция и описание сервисов: Сервисы описываются через четко определенные интерфейсы и метаданные, включая операции, семантику, динамические характеристики, политики безопасности и другие свойства, скрывая внутреннюю реализацию от потребителей;

– контрактное взаимодействие и гарантии качества: Взаимодействие между поставщиками и потребителями сервисов основано на формальных соглашениях или контрактах, определяющих обязательства сторон и гарантируя установленный уровень качества сервиса (SLA)[10];

– публикация и обнаружение сервисов: Сервисы регистрируются в сервисных реестрах или каталогах с использованием метаданных, что позволяет потребителям осуществлять поиск и динамически связываться с необходимыми сервисами.

сервис-ориентированная архитектура предоставляет предприятиям возможность разрабатывать бизнес-системы и ИТ-системы с точки зрения сервисов, доступных через интерфейсы, и результатов действий этих сервисов. Сервис является логическим представлением набора операций, направленных на достижение определенных бизнес-результатов. Он обладает автономностью, может состоять из других сервисов, и потребители сервиса не обязаны знать его внутреннюю структуру, что обеспечивает гибкость и масштабируемость системы в целом.

В контексте цифровой трансформации и перехода к облачным технологиям сервис-ориентированная архитектура играет ключевую роль в интеграции разнородных систем и приложений. Она служит основой для развития микро-сервисной архитектуры, где приложения состоят из набора небольших, независимых сервисов, которые могут развёртываться и масштабироваться автономно. Это позволяет организациям быстро внедрять инновации, улучшать взаимодействие с клиентами и партнёрами, а также эффективно реагировать на изменения рыночных условий [3].

Тем не менее, успешное внедрение сервис-ориентированной архитектуры требует внимания к управлению сложностью сервисной экосистемы, обеспечения безопасности, соблюдения стандартов и управления сервисами. Необходимо применять современные инструменты мониторинга и анализа производительности сервисов, а также внедрять лучшие практики в области разработки и эксплуатации сервис-ориентированных систем. Основными бизнес-драйверами для внедрения сервис-ориентированной архитектуры являются повышение оперативности бизнеса, снижение затрат на разработку и поддержку ИТ-систем, улучшение взаимодействия между подразделениями компании и внешними партнёрами, а также создание новых возможностей для предоставления услуг и продуктов на рынке. Таким образом, сервис-ориентированная архитектура выступает как стратегический инструмент, способствующий повышению эффективности бизнеса и достижению конкурентных преимуществ в современном цифровом мире [8].

Принципы сервис-ориентированной архитектуры формируют теоретическую базу для разработки модульных и адаптивных систем, характеризующихся автономностью и легкостью замены отдельных сервисов [29].

– в контексте сервис-ориентированной архитектуры, модульность подразумевает декомпозицию приложения на автономные сервисы, каждый из которых реализует специфическую бизнес-функцию. Такая структурная

организация способствует оптимизации управления системой, а также упрощает процессы обновления и тестирования компонентов;

– одним из ключевых аспектов сервис-ориентированной архитектуры является проектирование сервисов с учетом их потенциального применения в различных системах и сценариях эксплуатации. Это не только снижает издержки на разработку, но и обеспечивает согласованность бизнес-логики в рамках всей организации [30];

– кроссплатформенность в сервис-ориентированной архитектуры достигается посредством обеспечения совместимости сервисов, работающих на различных платформенных средах, через стандартизированные протоколы взаимодействия, такие как HTTP, XML или JSON. Данная характеристика облегчает интеграцию как внутри организации, так и с внешними партнерами [13];

– принцип абстракции в сервис-ориентированной архитектуры предполагает скрытие внутренней реализации сервисов за четко определенными интерфейсами. Это позволяет осуществлять обновления сервисов без необходимости изменения внешней структуры системы, тем самым повышая управляемость и гибкость архитектурных решений;

– сервис-ориентированная архитектуры основывается на концепции децентрализованного управления, где каждый сервис обладает автономностью в управлении своими версиями и жизненным циклом. Это минимизирует взаимозависимость между компонентами системы, способствуя повышению устойчивости и масштабируемости;

– в условиях распределенной среды, характерной для сервис-ориентированной архитектуры, обеспечение безопасности требует комплексного подхода, включающего механизмы аутентификации и шифрования данных. Такой подход необходим для защиты сервисов, функционирующих на различных уровнях доверия;

– для поддержания надежной работы распределенных систем в рамках сервис-ориентированной архитектуры необходимо внедрение механизмов

мониторинга производительности и управления ресурсами. Это позволяет своевременно обнаруживать и устранять потенциальные проблемы, предотвращая возможные сбои и обеспечивая стабильность функционирования системы [22];

В таблице 1 указаны принципы проектирования качественных сервисов, которые реализуются с помощью характеристик веб-сервисов или сервис-ориентированной архитектуры [32].

Таблица 1 – Основные принципы реализации сервисов с помощью сервис-ориентированной архитектуры

Включено веб-сервисами	Технологически нейтральный	Независимость от конечной платформы.
	Стандартизированный	Стандартизированные протоколы.
	Расходный материал	Обеспечение автоматического обнаружения и использования.
Включено SOA	Многоразовый	Использование Сервиса, а не повторное использование путем копирования кода/реализации.
	Абстрагированный	Услуга абстрагируется от реализации.
	Опубликовано	Точная, опубликованная спецификация функциональности интерфейса сервиса, а не реализация.
	Формальный	Формальный договор между конечными точками возлагает обязательства на поставщика и потребителя.
	Соответствующий	Функциональность, представленная на уровне детализации, распознается пользователем как значимая услуга.

В целом приведенные тезисы подчеркивают ключевые характеристики веб-сервисов и сервис-ориентированной архитектуры. Для веб-сервисов это технологическая нейтральность, использование открытых стандартов и протоколов, а также возможность автоматического обнаружения и интеграции. Для сервис-ориентированной архитектуры – повторное использование функционала без копирования кода, абстракция реализации за чёткой спецификацией интерфейсов, формализованный договор об обязанностях поставщика и потребителя, а также предоставление сервисов, значимых и понятных для конечного пользователя. Вместе эти принципы создают гибкую, масштабируемую и стандартизированную среду для взаимодействия различных систем и приложений [35].

1.2 Особенности и преимущества использования сервисов в программировании

Сервис-ориентированная архитектура демонстрирует ряд значительных преимуществ по сравнению с традиционными монолитными архитектурными моделями, в которых все процессы функционируют как единое, неразделимое целое. Рассмотрим некоторые из ключевых преимуществ сервис-ориентированной архитектуры:

Сокращение временных издержек при выводе продуктов на рынок**,** в контексте сервис-ориентированной архитектуры разработчики могут осуществлять повторное использование существующих сервисов в различных бизнес-процессах, что способствует значительной экономии времени и ресурсов. Благодаря данной архитектурной парадигме возможно ускоренное создание приложений, избегая необходимости написания кода и интеграции компонентов с нуля, что в конечном итоге сокращает время выхода продукта на рынок [6].

Эффективное обслуживание и поддержка систем, модульная природа SOA облегчает процесс разработки, обновления и отладки отдельных сервисов

по сравнению с управлением крупными блоками кода в монолитных приложениях. Это позволяет изолировать изменения и модификации на уровне отдельных сервисов без воздействия на общую функциональность бизнес-процессов, обеспечивая тем самым устойчивость и гибкость системы.

Повышенная адаптивность к технологическим изменениям**,** сервис-ориентированная архитектура характеризуется высокой степенью адаптивности к динамично развивающимся технологическим трендам. Архитектурная гибкость позволяет эффективно и экономично модернизировать существующие приложения. Например, медицинские организации могут интегрировать функциональные возможности устаревших систем электронных медицинских карт в современные облачные приложения, что обеспечивает сохранение и использование ценных данных без значительных дополнительных затрат [7].

Кроссплатформенность и стандартизация, одним из научно обоснованных преимуществ сервис-ориентированной архитектуры является улучшенная кроссплатформенность между различными системами и платформами благодаря использованию стандартных протоколов и интерфейсов. Это способствует бесшовной интеграции различных компонентов и облегчает взаимодействие между разнородными системами, что является критически важным в условиях современной информационной экосистемы [11].

Масштабируемость и управляемость**.** сервис-ориентированная архитектура обеспечивает высокую масштабируемость системы за счет возможности добавления или удаления отдельных сервисов без необходимости кардинальных изменений в архитектуре. Это упрощает управление ресурсами и позволяет эффективно реагировать на изменяющиеся требования бизнеса, поддерживая устойчивое развитие информационных систем [14].

Архитектурные различия: Сервис-ориентированная архитектура и микро-сервисная архитектура

Объем и масштаб охвата. Сервис-ориентированная архитектура характеризуется широким охватом функциональных возможностей

предприятия, обеспечивая эффективное взаимодействие различных бизнес-подразделений на единой платформе обмена данными. В отличие от этого, микро-сервисная архитектура фокусируется на более узком спектре задач, предоставляя детализированное разделение функциональности.

Например, в системе электронной коммерции управление запасами может быть реализовано как отдельный сервис-ориентированной архитектуры -сервис. Однако применение микро-сервисного подхода позволит декомпозировать управление запасами на более мелкие и специализированные сервисы, такие как проверка доступности товаров, обработка заказов и бухгалтерский учет, что способствует повышению гибкости и масштабируемости системы.

Реализация и интеграция сервисов, внедрение сервис-ориентированной архитектуры предполагает интеграцию различных типов сервисов в единую прикладную среду. Этот процесс обычно осуществляется посредством корпоративной сервисной шины (Enterprise Service Bus, ESB), которая обеспечивает взаимодействие следующих типов сервисов:

- функциональные сервисы: поддерживают конкретные бизнес-операции, обеспечивая выполнение специализированных задач;
- корпоративные сервисы: предоставляют определенные бизнес-функции другим сервисам внутри корпоративной экосистемы;
- прикладные сервисы: используются разработчиками для создания и развертывания приложений, обеспечивая необходимую функциональность;
- инфраструктурные сервисы: управляют нефункциональными аспектами системы, такими как аутентификация, безопасность и мониторинг [17].

В противоположность этому, микро-сервисная архитектура представляет собой более детализированную и независимую реализацию принципов SOA. Микро-сервисы функционируют автономно, предоставляя узконаправленные функции и не разделяя ресурсы с другими сервисами, что способствует повышению изоляции и независимости компонентов системы.

Хранилище данных, в среде сервис-ориентированной архитектуры используется единый уровень хранения данных, который совместно используется подключенными сервисами. Различные корпоративные приложения получают доступ к одним и тем же данным, что способствует повторному использованию информации и оптимизации ценности репозиториев данных. Микро-сервисная архитектура, напротив, предусматривает наличие отдельного хранилища данных для каждого сервиса. Такая независимость данных ставит во главу угла изоляцию и автономность микро-сервисов, что позволяет минимизировать зависимости и повысить устойчивость системы к изменениям [15].

Процессы развертывания, развертывание сервисов в рамках сервис-ориентированной архитектуры может быть сложным из-за взаимосвязи между сервисами. Например, изменение или добавление нового сервиса требует модификации всего приложения, что увеличивает сложность и время развертывания. Кроме того, приложения сервис-ориентированной архитектуры не всегда могут эффективно использовать преимущества контейнеризации, которая обеспечивает абстрагирование приложений от операционных систем и аппаратного обеспечения [12].

В микро-сервисной архитектуре развертывание осуществляется проще, поскольку каждый сервис является независимым приложением, предназначенным для масштабирования в облачной среде. Сервисы могут быть размещены в контейнерах и развернуты в облачных инфраструктурах независимо друг от друга, что значительно упрощает процессы масштабирования и управления.

1.3 Основные стандарты и технологии в области сервис-ориентированной архитектуры

Веб-сервисы основываются на ряде фундаментальных веб-стандартов, которые обеспечивают их функциональность и взаимодействие в распределенных средах. К основным из этих стандартов относятся:

– XML (Extensible Markup Language): Расширяемый язык разметки, предназначенный для хранения и передачи структурированных данных. XML является развивающимся и открытым стандартом, разработанным рабочей группой «XML Working Group» под эгидой консорциума «W3C» в 1996 году. Спецификация XML определяет набор правил, включающих элементы, атрибуты, сущности и другие компоненты, а также их порядок, по которым создаются документы на данном языке. Сам по себе XML является языком, который описывает сам себя, однако для документов в определенной предметной области могут быть наложены дополнительные ограничения. Эти ограничения обычно оформляются с помощью схем документов, представляемых на специальном языке, таком как XML Schema Definition Language. XML-документы хранятся в текстовом формате и обычно используют универсальный стандарт кодировки «Unicode», включающий символы практически всех алфавитов мира. В древовидную структуру XML можно уложить информацию из хранилища данных с любой структурой. Таким образом, с помощью XML легко определять новые форматы данных, которые являются простыми, переносимыми, независимыми от конкретной операционной системы и приложений, создавших эти данные. Кроме того, данные XML-документа являются строго структурированными, поддерживают возможности поиска и преобразования для последующего визуального отображения, что делает их подходящими как для автоматической обработки, так и для восприятия человеком [16]. Язык XML играет важную роль в веб-сервисах, являясь основой для таких технологий, как SOAP и WSDL, а также определяя формат данных, используемый для обмена информацией между потребителем сервиса и самим сервисом. Посредством XML структуру документа, его содержимое и способы отображения можно представить в виде трех различных компонентов. Это предоставляет возможность отображать один и тот же документ различными способами, например, в зависимости от типа клиента или типа запрашиваемого документа. Также необходимо замечание, что все средства создания и потребления веб-сервисов содержат библиотеки

для обработки XML-документов. Это могут быть библиотеки, поддерживающие XML Document Object Model (DOM), SAX для извлечения и обработки информации из XML-документов;

– SOAP (Simple Object Access Protocol): Протокол обмена сообщениями на базе XML, обеспечивающий взаимодействие в среде веб-сервисов. SOAP является основанным на языке XML стандартом для обмена сообщениями в распределенных средах, таких как Всемирная паутина. Он предназначен для удаленного вызова методов и может использоваться с различными транспортными протоколами, включая HTTP и SMTP [33]. Протокол SOAP был разработан корпорациями «IBM», «Lotus Development Corporation», «Microsoft», «DevelopMentor» и «Userland Software». Он позволяет приложениям взаимодействовать между собой через интернет, используя для этого XML-документы, называемые сообщениями SOAP. Протокол SOAP совместим с любой объектной моделью, поскольку включает только те функции и методы, которые абсолютно необходимы для формирования коммуникационной инфраструктуры. Это делает SOAP независимым от платформы и конкретных приложений, а для его реализации может быть использован любой язык программирования. SOAP поддерживает любой транспортный протокол и любые методы кодирования данных, позволяющие приложениям посылать информацию любого типа, например, изображения или документы. С точки зрения приложения, при использовании SOAP для вызова методов веб-сервисов требуется генерация запроса и обработка XML-документа, содержащего ответ. Можно использовать XSLT, SAX, DOM или XDOM для преобразования XML-документа в вид, более подходящий для конкретного приложения. По мере увеличения числа средств обработки веб-сервисов необходимость в непосредственной обработке XML-документов будет уменьшаться;

– WSDL (Web Services Description Language): Язык описания веб-сервисов, основанный на XML, который содержит всю информацию, необходимую для доступа к данному сервису. Интерфейс для доступа к веб-

сервису описывается с помощью WSDL и представляет собой коллекцию портов, которые, в свою очередь, являются коллекцией абстрактных операций и сообщений. Абстракция операций и сообщений позволяет связывать их с различными протоколами и форматами данных, такими как SOAP, HTTP, GET/POST или MIME [24];

– UDDI (Universal Description, Discovery, and Integration): Универсальный интерфейс распознавания, описания и интеграции, который регистрирует и публикует определения веб-сервисов. Структура UDDI определяет модель данных в программных интерфейсах на основе XML и SOAP для регистрации и обнаружения коммерческой информации, включая веб-сервисы. Реестр UDDI был создан независимым консорциумом производителей, основанным компаниями «Microsoft», «IBM» и «Ariba», занимающимся разработкой стандарта для описания, регистрации и локализации веб-сервисов. «Microsoft», «IBM», «Hewlett-Packard» и «SAP» на начальном этапе предоставляли общедоступный UDDI-сервис, который концептуально копирует систему имен доменов Интернета, осуществляющую преобразование имен в IP-адреса. На практике реестр UDDI более похож на сервис реплицируемой базы данных, доступной через Интернет. UDDI представляет собой каталог веб-сервисов. Информацию об определенном веб-сервисе можно обнаружить, пытаясь найти адрес WSDL-файла, указывающего на веб-сервис поставщика. Для регистрации и поиска информации UDDI использует SOAP. Предприятие может осуществлять регистрацию сведений о веб-сервисе в UDDI, сначала генерируя WSDL-файл, который описывает веб-сервисы, поддерживаемые SOAP-процессором, а затем с помощью API UDDI регистрируя информацию в хранилище. После отправки предприятием этих сведений, совместно с другой контактной информацией, элемент реестра будет содержать адрес, указывающий на WSDL-файл сайта SOAP-сервера, описывающий данный веб-сервис. SOAP-процессор другого предприятия запрашивает реестр для получения WSDL. Для отправки конкретной операции по определенному протоколу клиент может сгенерировать соответствующее

сообщение. В данном случае клиент и сервер должны договориться о том, чтобы протокол был одинаковым (в данном случае "SOAP поверх HTTP"), и совместно использовать одинаковые понятия или семантические описания сервиса, представленные в WSDL. При повсеместном распространении основных стандартов общее понимание будет гарантировано посредством WSDL [9];

Вышеперечисленные стандарты являются фундаментальными компонентами экосистемы веб-сервисов, обеспечивая их совместимость, кроссплатформенность и эффективное взаимодействие в различных бизнес-процессах и технических решениях. Их использование способствует развитию современной информационной инфраструктуры, позволяя предприятиям и приложениям эффективно обмениваться данными и функциональностью в глобальном масштабе.

Выводы к главе 1

В данной главе, прежде всего, осуществлён всесторонний теоретико-методологический анализ основополагающих концепций и ключевых принципов сервис-ориентированной архитектуры, которая, как представляется, формирует концептуально-интеллектуальный каркас для проектирования высоко адаптивных, масштабируемых и, в целом, эволюционирующих информационно-технологических систем. Следует отметить, что особый акцент сделан на детальном рассмотрении преимуществ сервис-ориентированной парадигмы, включающих, помимо прочего, модульное построение, расширенную возможность повторного использования компонентов, стандартизированные интерфейсы взаимодействия, а также технологическую нейтральность, обеспечивающую независимость от конкретных платформенных решений.

При этом, в контексте комплексного анализа были изучены фундаментальные аспекты реализации сервисов: поддержка слабой связности элементов, обеспечение полноформатной кроссплатформенности,

многоуровневое абстрагирование внутренней логики, а также организация формализованного контрактного взаимодействия посредством унифицированных и широко признанных протоколов. Подобный подход, как представляется, позволяет сформировать системно-организованный и методологически обоснованный фундамент для дальнейшего совершенствования архитектурных моделей в рамках современного ИТ-ландшафта.

Глава 2 Формирование модели оптимизации процессов управления

2.1 Описание системы управления заказами

В условиях стремительного развития электронной коммерции и цифровой трансформации бизнес-процессов системы управления заказами выступают стратегически важным инструментом для обеспечения конкурентоспособности предприятий. Целями разработки системы управления заказами являются повышение надёжности, точности и скорости обработки заказов, а также достижение гибкости и масштабируемости для поддержки увеличивающихся объёмов данных, количества пользователей и расширения ассортимента товаров. Удобство использования системы для конечных пользователей, включая интуитивно понятный интерфейс и высокую скорость работы, также занимает ключевую позицию среди приоритетов [31].

Одной из задач системы является автоматизация бизнес-процессов, направленных на минимизацию человеческого фактора, что способствует снижению операционных рисков и затрат. Кроме того, система должна обеспечивать высокую степень интеграции с корпоративными платформами, такими как CRM (Customer Relationship Management) для управления взаимоотношениями с клиентами и ERP (Enterprise Resource Planning) для согласованного управления ресурсами компании. Это позволяет добиться единства данных и оперативности в принятии управленческих решений [18].

Система управления заказами должна также решать следующие задачи:

– обеспечение точности и своевременности выполнения заказов**.** Мониторинг статусов заказов в режиме реального времени и автоматическое уведомление пользователей о процессе выполнения;

– управление товарным ассортиментом**.** Данная задача включает добавление, обновление и удаление товаров с учётом их наличия на складах и других параметров;

– организация обработки платежей**.** Данная задача подразумевает поддержку различных платёжных систем, безопасное проведение транзакций и работа с возвратами;

– логистика доставки**.** Данная задача включает выбор оптимальных способов доставки, отслеживание посылок, управление складами и взаимодействие с логистическими компаниями;

– управление пользовательским взаимодействием**.** Задача подразумевает создание удобного механизма регистрации, авторизации и управления учётными записями;

– обеспечение безопасности и защиты данных**.** Задача включает внедрение современных технологий шифрования и управления доступом;

Таким образом, данная система должна не только улучшить качество взаимодействия с клиентами, но и способствовать укреплению конкурентных позиций предприятия за счёт оптимизации ключевых бизнес-процессов.

Система управления заказами включает комплекс функциональных областей, каждая из которых обеспечивает выполнение специфических задач, критически важных для поддержания её целостной и эффективной работы. Эти области взаимосвязаны, что позволяет системе интегрировано обрабатывать данные и предоставлять пользователям удобный функционал. К основным функциональным областям относятся:

а) каталог товаров:

1) включает инструменты для создания и управления структурированным списком товаров, доступным пользователям,

2) предоставляет функционал для фильтрации по категориям, ценовым диапазонам, брендам и другим характеристикам,

3) предоставляет детализированную информацию о каждом товаре, включая описания, технические характеристики, изображения, видео обзоры и отзывы покупателей,

4) реализует возможность динамического обновления данных о товарах с учётом наличия на складах и изменений цен;

б) корзина покупок:

1) позволяет пользователям добавлять товары для последующего оформления заказа, изменять их количество и удалять позиции,

2) автоматически рассчитывает итоговую стоимость заказа, включая налоги, скидки и стоимость доставки,

3) обеспечивает сохранение данных о товарах в корзине для авторизованных пользователей при повторных посещениях сайта;

в) управление заказами:

1) организует процессы оформления заказов, включая выбор параметров доставки и платежа,

2) отслеживает статус заказов на всех этапах: от оформления до доставки,

3) хранит историю заказов для возможности повторного оформления,

4) включает модуль обработки отмен заказов и возвратов;

г) платежи:

1) обеспечивает интеграцию с популярными платёжными системами, и банковские карты,

2) гарантирует защиту данных о транзакциях через протоколы шифрования,

3) управляет возвратами средств в случае отмены заказа или претензий со стороны пользователей;

д) доставка:

1) интегрируется с логистическими и курьерскими службами для автоматического расчёта стоимости и сроков доставки,

2) обеспечивает выбор различных способов доставки: стандартная, экспресс, самовывоз.

3) включает функционал отслеживания посылок и уведомлений о статусе доставки;

е) управление пользователями:

1) предоставляет механизмы регистрации, авторизации и аутентификации пользователей,

2) управляет профилями клиентов, включая настройки уведомлений, адреса доставки и предпочтения,

3) обеспечивает разграничение прав доступа для различных ролей, таких как администраторы, менеджеры и клиенты.

Каждая из перечисленных функциональных областей вносит значительный вклад в обеспечение как удобства конечных пользователей, так и общей эффективности бизнес-процессов. Благодаря интеграции этих областей система управления заказами формирует надёжную основу для успешной цифровой трансформации предприятий и повышения их конкурентоспособности на рынке.

2.2 Анализ бизнес-процессов с использованием BPMN

В современном динамичном бизнес-среде, характеризующейся высокой конкуренцией и быстро меняющимися требованиями потребителей, оптимизация бизнес-процессов становится критически важной для обеспечения эффективности и устойчивости организации. В этой связи, для глубокого понимания и анализа процессов системы управления заказами, применяется стандартизированная нотация BPMN (Business Process Model and Notation). Данная нотация предоставляет универсальный язык для моделирования бизнес-процессов, позволяя детализовать каждый этап и визуализировать взаимодействия между различными участниками процесса [20].

Моделирование процесса оформления заказа представляет собой комплексный анализ, направленный на детальное рассмотрение всех этапов взаимодействия пользователя с системой. Это включает в себя путь пользователя от первоначального выбора товара до окончательной доставки продукта. Использование нотации BPMN в этом контексте обеспечивает возможность создания точной и понятной модели процесса, что является фундаментом для последующего анализа и оптимизации. Визуализация

процесса с помощью диаграмм BPMN способствует выявлению ключевых точек взаимодействия между различными компонентами системы, а также идентификации потенциальных узких мест и неэффективных мест. Это позволяет определить области, требующие улучшения, и разработать стратегии для повышения эффективности и качества обслуживания клиентов.

Процесс оформления заказа включает в себя следующие основные шаги:

– выбор товара и добавление его в корзину: Пользователь взаимодействует с каталогом товаров, используя функционал поиска и фильтрации для нахождения интересующих его позиций. Возможность сравнения товаров и чтения отзывов способствует осознанному выбору;

– проверка наличия товара на складе: После добавления товара в корзину система автоматически проверяет его доступность на складе в режиме реального времени. Это предотвращает возможные ситуации, связанные с отсутствием товара после оформления заказа;

– оформление заказа пользователем: На этом этапе пользователь вводит данные для доставки, выбирает способ оплаты и подтверждает заказ. Ввод корректной информации является критическим для успешной обработки заказа;

– обработка платежа и подтверждение заказа: Система инициирует платежный процесс, взаимодействуя с платежными шлюзами и обеспечивая безопасность транзакции. По завершении пользователь получает подтверждение заказа;

– организация доставки товара: После успешной оплаты начинается процесс подготовки товара к отправке, включая упаковку и передачу логистическим службам. Система может автоматически выбирать наиболее подходящего перевозчика на основе различных критериев;

– уведомление пользователя о статусе заказа: Пользователь регулярно информируется о состоянии своего заказа через электронную почту, SMS или уведомления в приложении, что повышает уровень доверия и удовлетворенности.

Каждый из этих этапов детализируется на диаграмме BPMN, где отображаются различные элементы процесса:

- события: Начальные, промежуточные и конечные события, обозначающие старт, переходы и завершение процесса;
- действия (Задачи): Конкретные операции, выполняемые участниками процесса;
- шлюзы: Механизмы, управляющие потоком процесса, включая ветвления и слияния потоков;
- участники процесса: Различные роли и отделы, вовлеченные в процесс, представленные в виде "пулов" и "дорожек".

Использование BPMN позволяет создать наглядную и подробную модель процесса, которая служит основой для выявления неэффективности, дублирования операций и потенциальных рисков. Это, в свою очередь, позволяет разработать рекомендации по оптимизации процесса, улучшению взаимодействия между участниками и повышению общей производительности системы.

На рисунке 2 изображён один из бизнес-процессов смоделированный через BPMN диаграмму.

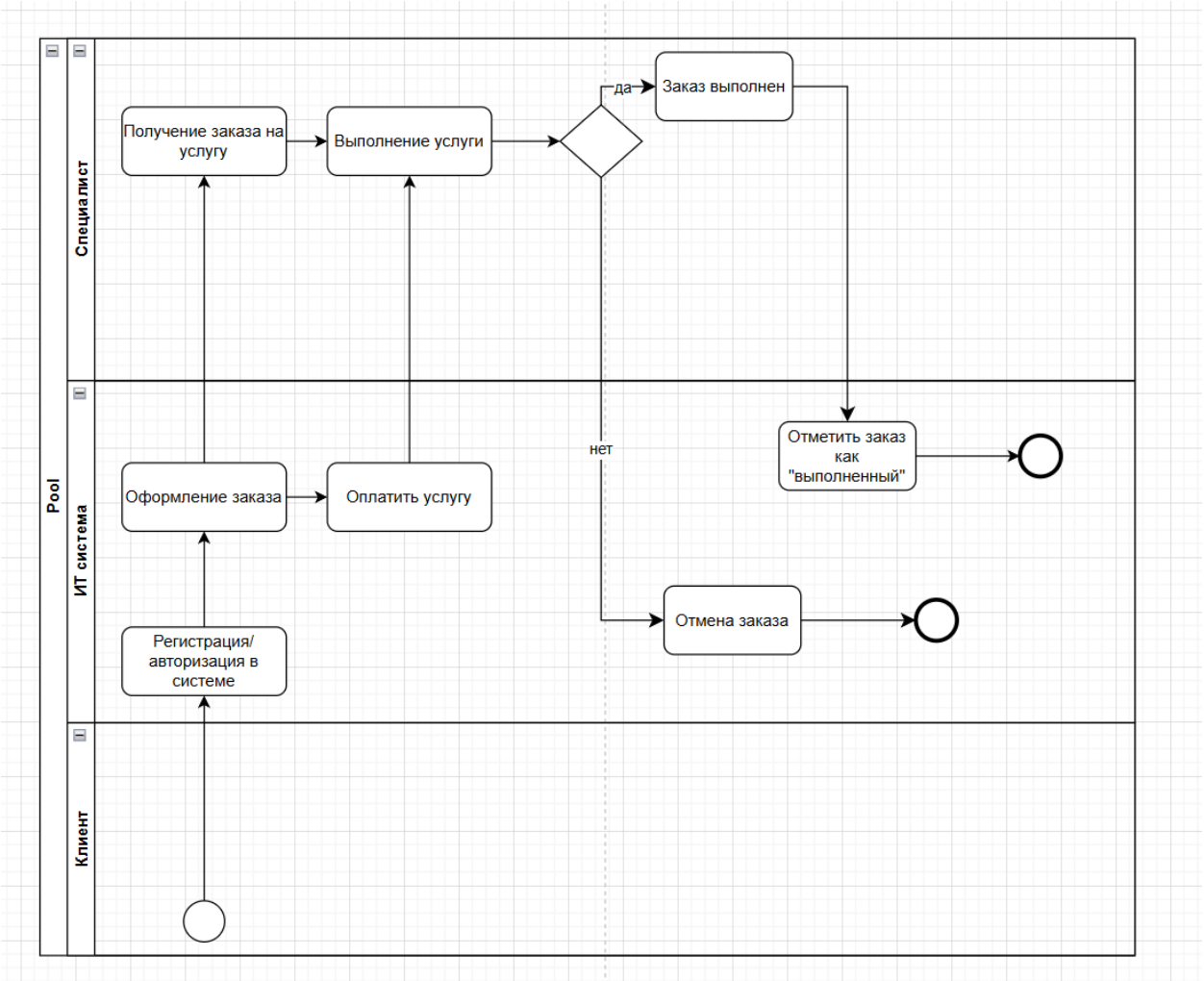

Рисунок 2 – Описание одного из бизнес-процессов в системе управления заказов

На основе созданных моделей бизнес-процессов осуществляется выявление и идентификация сервисов, необходимых для поддержки и автоматизации ключевых этапов процесса оформления заказа. Данный подход соответствует принципам сервис-ориентированной архитектуры (SOA), которая направлена на создание модульной и гибкой системы, где каждый сервис выполняет определенную бизнес-функцию и может быть независимо развит и масштабирован. Выделение сервисов на основе бизнес-процессов обеспечивает согласованность между бизнес-требованиями и технической реализацией, а также повышает адаптивность системы к изменениям внешней среды и внутренним потребностям организации.

В результате анализа были определены следующие основные сервисы:

– сервис каталога товаров: Отвечает за управление информацией о товарах, включая описание, категории, цены, изображения и наличие на складе. Обеспечивает доступность данных для других сервисов и пользовательского интерфейса, а также поддерживает функции поиска и фильтрации;

– сервис корзины покупок: Управляет процессом формирования заказа, включая добавление и удаление товаров из корзины, расчет стоимости с учетом скидок и налогов, а также сохранение состояния корзины между сеансами пользователя;

– сервис обработки заказов: Координирует процесс оформления и выполнения заказа, начиная от его создания до завершения. Отвечает за изменение статусов заказа, взаимодействие с другими сервисами и обеспечение целостности данных;

– сервис платежей: Обеспечивает безопасную обработку финансовых транзакций, включая интеграцию с платежными шлюзами, верификацию платежей и обработку возвратов. Поддерживает различные методы оплаты и обеспечивает соответствие стандартам безопасности, таким как PCI DSS;

– сервис доставки: Управляет логистическими операциями, включая выбор курьерской службы, генерацию накладных, отслеживание посылок и управление взаимодействием с внешними логистическими провайдерами;

– сервис управления пользователями: Обеспечивает функциональность регистрации, аутентификации и авторизации пользователей, а также управление профилями и настройками. Отвечает за безопасность доступа и защиту персональных данных в соответствии с нормативными требованиями.

На рисунке 3 изображена схема компонентов.

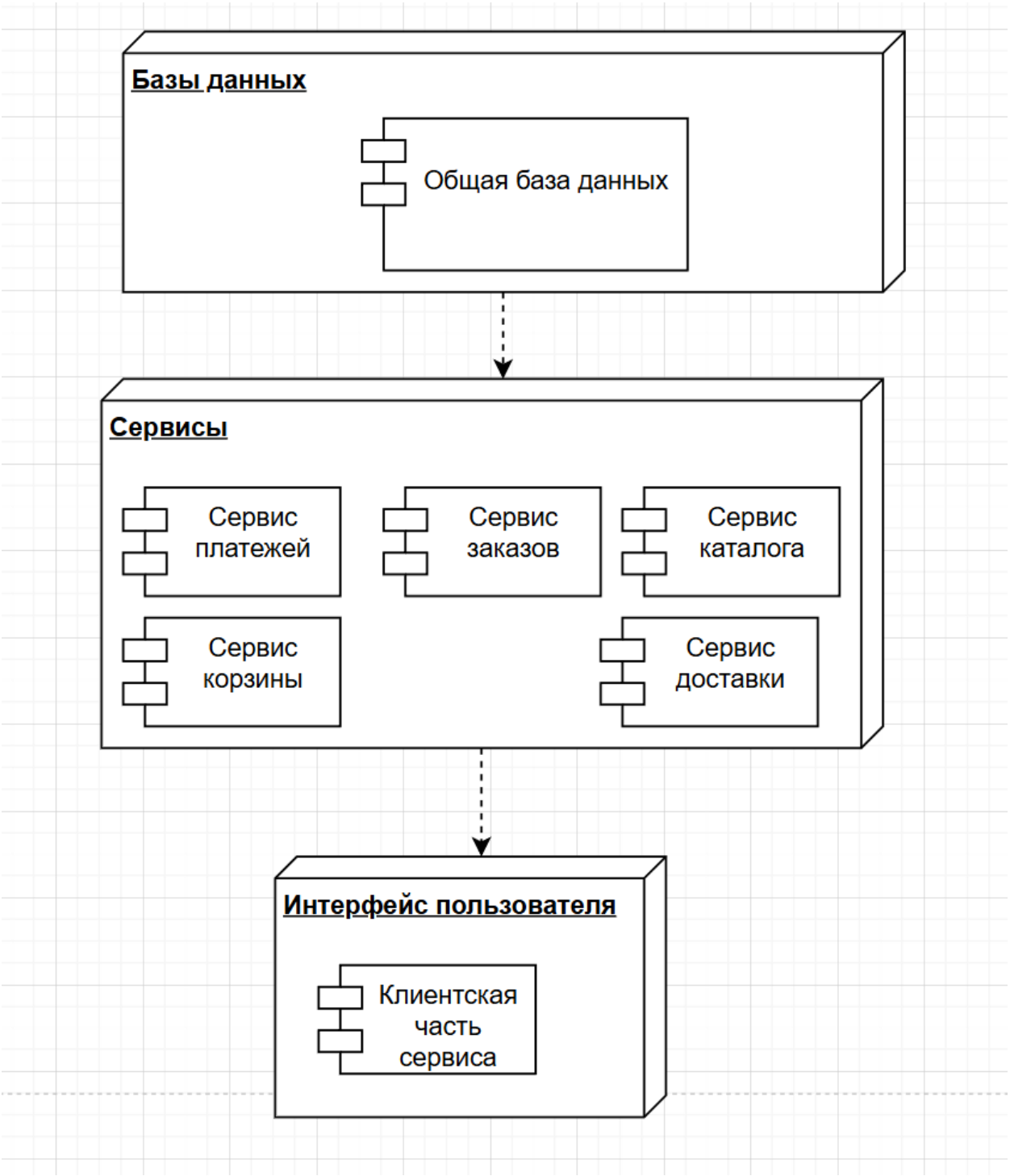

Рисунок 3 – Компоненты ИС

Каждый из этих сервисов интегрируется в общую архитектуру системы посредством четко определенных интерфейсов и протоколов взаимодействия, таких как RESTful API или SOAP. Это обеспечивает:

– модульность, возможность независимого развития и развёртывания сервисов без влияния на всю систему;

– гибкость, способность быстро адаптироваться к изменениям бизнес-требований и внедрять новые функции;

– масштабируемость, возможность масштабировать отдельные сервисы в соответствии с нагрузкой и требованиями производительности;

– надежность, повышение устойчивости системы к сбоям за счет распределения функциональности между сервисами.

Интеграция сервисов также подразумевает реализацию механизмов оркестрации и хореографии процессов, обеспечивая согласованность и последовательность операций в распределенной системе. Использование брокеров сообщений и очередей способствует повышению надежности и асинхронности взаимодействия между сервисами.

Кроме того, применение современных подходов, таких как сервис-ориентированная архитектура и контейнеризация позволяет улучшить управляемость и автоматизацию развёртывания системы.

В заключение, проведенный анализ бизнес-процессов с использованием BPMN и последующее выявление сервисов на их основе позволяют создать эффективную, гибкую и масштабируемую систему управления заказами. Такая система способна удовлетворять текущие и будущие потребности бизнеса, обеспечивая высокий уровень клиентского сервиса и конкурентоспособность на рынке.

2.3 Использование UML и других диаграмм для моделирования системы

Моделирование информационных систем с использованием UML (Unified Modeling Language) играет ключевую роль в проектировании сложных программных систем, таких как система управления заказами. UML предоставляет стандартный набор инструментов, которые обеспечивают формализацию и визуализацию различных аспектов системы, начиная с функциональных требований и заканчивая физической архитектурой [25].

Диаграммы вариантов использования являются базовым инструментом для описания функциональных требований системы. Они позволяют визуализировать, какие действия могут выполнять акторы (пользователи или внешние системы) при взаимодействии с системой, а также устанавливают границы функциональности разрабатываемого программного обеспечения [23].

В контексте системы управления заказами основные варианты использования включают следующие сценарии:

- регистрация нового пользователя и аутентификация уже зарегистрированного клиента;
- просмотр каталога товаров, включая фильтрацию и поиск по категориям;
- добавление товаров в корзину покупок и её редактирование;
- оформление заказа, включая выбор метода доставки и способа оплаты;
- отслеживание статуса заказа через личный кабинет или уведомления;
- управление каталогом товаров и заказами со стороны администратора.

Каждый вариант использования описывается в терминах акторов (например, "Покупатель", "Администратор") и их взаимодействия с системой. Диаграммы вариантов использования обеспечивают высокоуровневое представление системы, что важно для согласования требований между технической командой и бизнес-участниками.

Диаграммы классов в UML позволяют представить статическую структуру системы, включая основные сущности (классы), их атрибуты, методы и связи. Это инструмент структурного моделирования, обеспечивающий целостность системы и ясность в определении её компонентов [26].

На рисунке 4 изображена подробная диаграмма классов.

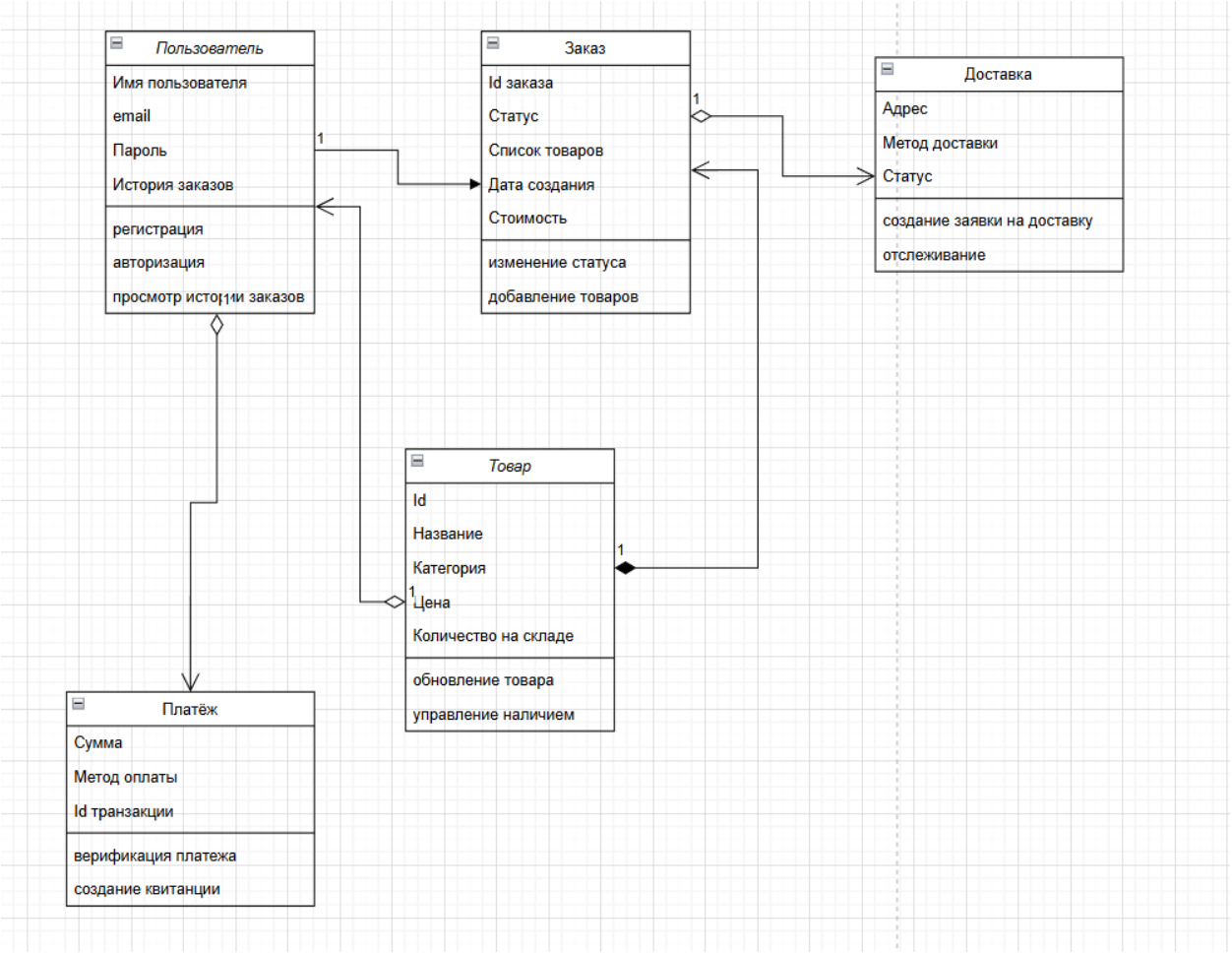

Рисунок 4 – диаграмма классов

Для системы управления заказами основные классы включают:

– класс "Пользователь",

– атрибуты: имя пользователя, электронная почта, пароль, история заказов;

– методы: регистрация, авторизация, просмотр истории заказов;

– класс "Товар",

– атрибуты: уникальный идентификатор, название, категория, цена, количество на складе;

– Методы: обновление характеристик, управление наличием;

– Класс "Заказ",

– Атрибуты: идентификатор заказа, статус, список товаров, дата создания, общая стоимость.

– Методы: изменение статуса, добавление товаров;
– Класс "Платёж",
– Атрибуты: сумма, метод оплаты, идентификатор транзакции;
– Методы: верификация платежа, генерация квитанции;
– Класс "Доставка",
– Атрибуты: адрес, метод доставки, статус;
– Методы: создание заявки на доставку, отслеживание.

Диаграммы классов устанавливают логические взаимосвязи между элементами системы, например, связь "один ко многим" между классами "Пользователь" и "Заказ".

Диаграммы последовательностей позволяют моделировать динамическое поведение системы, визуализируя последовательность обмена сообщениями между объектами и сервисами для выполнения определённых функций. Это важно для понимания временной структуры взаимодействий и согласования логики работы.

Для системы управления заказами ключевые сценарии включают:

а) пользователь добавляет товар в корзину:
   1) сообщение: запрос на проверку наличия товара,
   2) ответ: подтверждение или отказ;

б) оформление заказа:
   1) сообщение: запрос на расчёт общей стоимости,
   2) ответ: подтверждение и генерация заказа;

в) Обработка платежа:
   1) сообщение: запрос на проведение транзакции,
   2) ответ: статус выполнения платежа;

г) доставка:
   1) сообщение: передача данных логистическому сервису,
   2) ответ: обновление статуса доставки.

Диаграммы последовательностей подробно отображают взаимодействия между пользователем, системой, базой данных и внешними сервисами. На рисунке 5 изображена use case диаграмма одного из бизнес-процессов.

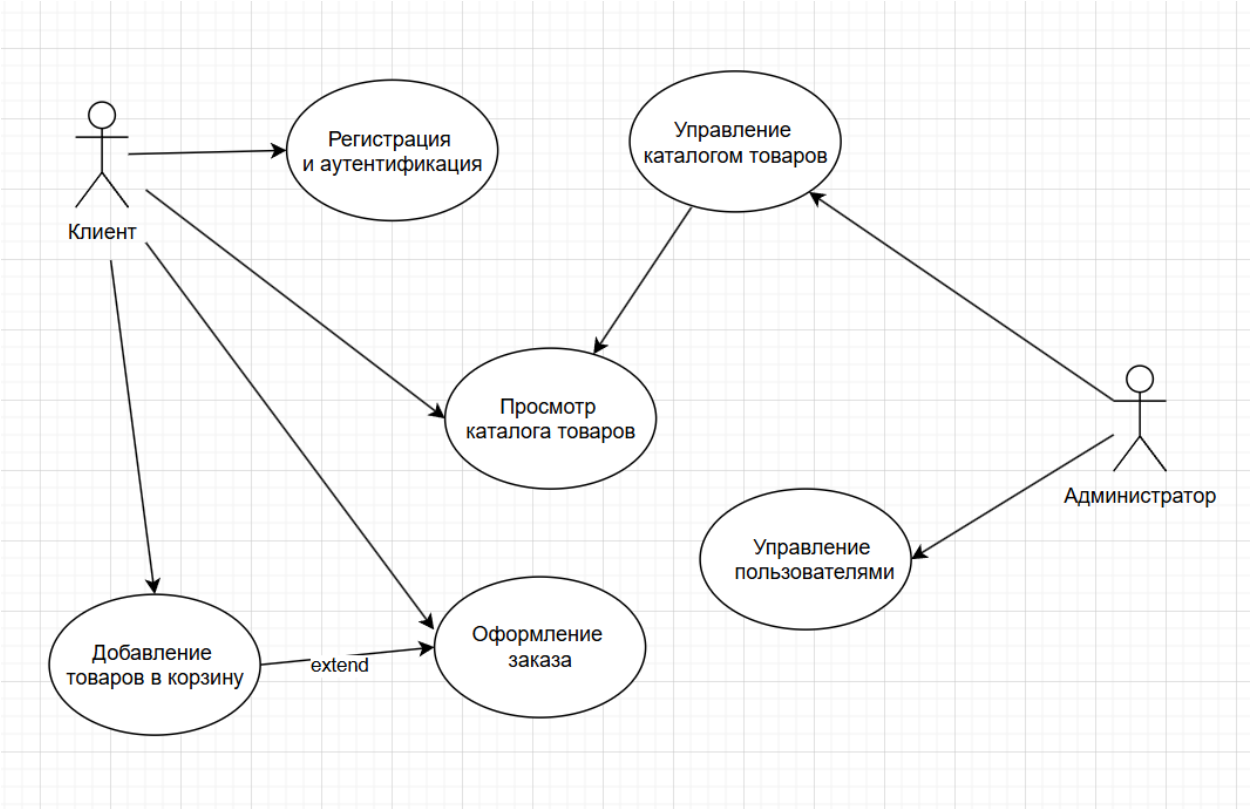

Рисунок 5 – Use case диаграмма одного из бизнес-процессов в системе управления заказов

Диаграммы развёртывания представляют собой средство визуализации физической архитектуры системы. Они показывают, как программные компоненты размещены на узлах аппаратной инфраструктуры, что важно для обеспечения масштабируемости и производительности.

Для системы управления заказами основные элементы диаграмм развёртывания включают:

- узлы (Nodes)**:** серверы приложений, базы данных, хранилища файлов;
- артефакты (Artifacts)**:** исполняемые файлы, библиотеки, конфигурационные файлы;
- связи (Connections)**:** сетевые соединения между серверами, API для интеграции с внешними системами.

Пример: Сервер приложения взаимодействует с сервером базы данных для хранения информации о заказах, а также с внешними сервисами платежей через API. Таким образом, использование UML и других диаграмм позволяет разработчикам и бизнес-аналитикам комплексно анализировать и проектировать систему управления заказами, согласовывая её функциональные и нефункциональные требования. Это обеспечивает высокую степень прозрачности и точности в реализации проекта, снижая риск ошибок и обеспечивая соответствие ожиданиям участников [19].

2.4 Модель данных и проектирование базы данных

Проектирование схемы базы данных представляет собой один из наиболее критически значимых этапов разработки системы управления заказами, поскольку от структурной организации данных напрямую зависит эффективность как хранения, так и последующей обработки информации. В данном контексте, на основе тщательно проведенного анализа требований системы и детального изучения бизнес-процессов, осуществляется разработка схемы базы данных. Эта схема включает в себя разнообразные таблицы, предназначенные для систематического хранения информации о товарах, пользователях, заказах, платежах и процессах доставки. [21]

Основными таблицами, включенными в базу данных, являются:

– users (Пользователи): Данная таблица содержит детализированные данные о пользователях системы, включая их имена, контактную информацию, адреса доставки, а также историю совершенных заказов. Кроме того, в таблице могут храниться данные о предпочтениях пользователей и их поведении на платформе, что способствует персонализации сервисов;

– products (Товары): Эта таблица включает исчерпывающую информацию о товарах, представленных в системе. В частности, она содержит названия товаров, их описания, цены, классификацию по категориям, а также данные о наличии на складе. Дополнительно могут быть включены такие параметры, как рейтинги товаров, отзывы пользователей и изображения;

– orders (Заказы): Таблица заказов предназначена для отслеживания всех операций, связанных с оформлением заказов. Она связывает пользователей с конкретными товарами, указывая даты размещения заказов, их текущие статусы (например, обработка, отправка, доставлено) и другие связанные параметры;

– payments (Платежи): в данной таблице регистрируются все платежные транзакции, связанные с заказами. Здесь фиксируются суммы платежей, даты проведения транзакций, а также используемые методы оплаты (например, кредитные карты, электронные кошельки, банковские переводы);

– deliveries (Доставки): Таблица доставки управляет информацией, касающейся процесса доставки заказанных товаров. Включает данные о адресах доставки, ожидаемых сроках, текущих статусах доставки (например, в пути, на складе, доставлено) и информации о курьерских службах.

На рисунке 6 описана подробная структура данных базы данных сервис-ориентированного приложения управления заказами.

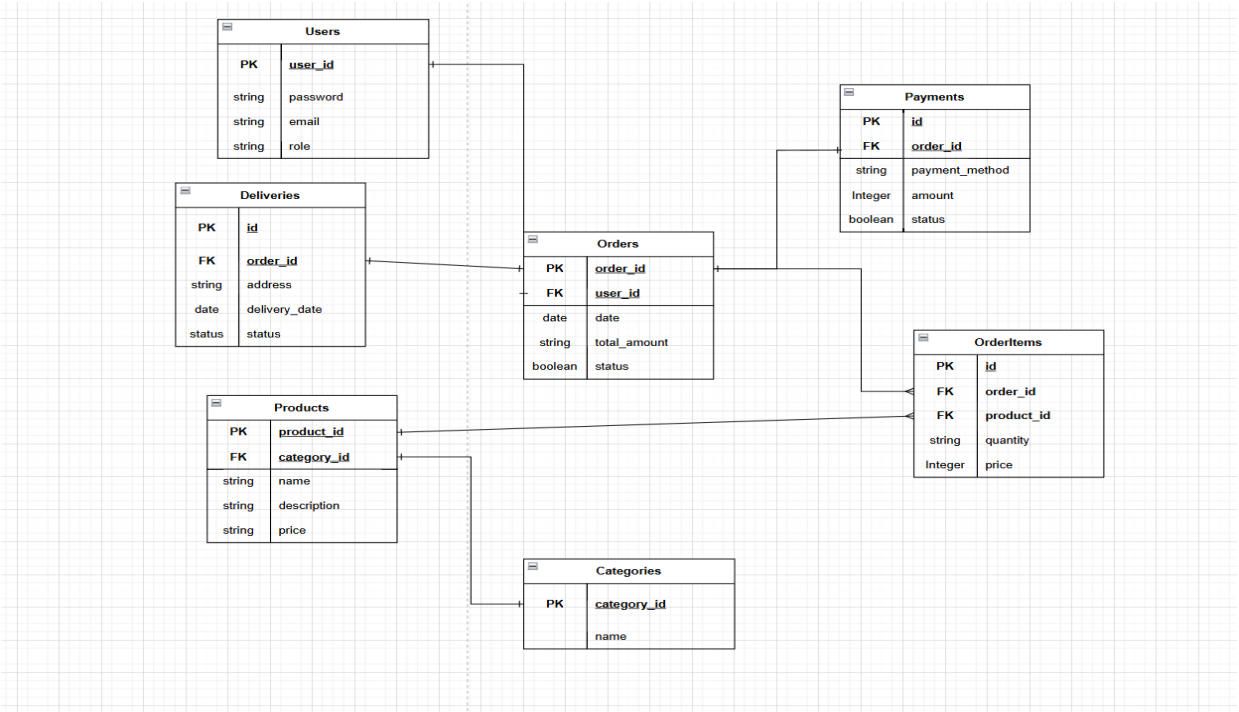

Рисунок 6 – структура базы данных

Каждая из этих таблиц оснащена первичными ключами, обеспечивающими уникальную идентификацию каждой записи. Кроме того, используются внешние ключи для установления и поддержания связей между таблицами, что способствует поддержанию целостности данных и позволяет эффективно выполнять запросы и операции над данными. Например, внешние ключи связывают таблицу заказов с таблицей пользователей и таблицей товаров, что обеспечивает логическую связность информации и облегчает выполнение комплексных аналитических запросов.

Целостность данных является фундаментальным и неотъемлемым требованием для любой системы управления данными, включая систему управления заказами. В контексте данной базы данных целостность данных обеспечивается посредством применения различных механизмов и методик, таких как ограничения целостности, транзакционный контроль и индексация данных [27].

Ограничения целостности**:** Включают в себя несколько уровней обеспечения корректности данных:

– первичные ключи (Primary Keys): Обеспечивают уникальную идентификацию каждой записи в таблице, предотвращая дублирование и гарантируя уникальность данных;

– внешние ключи (Foreign Keys): Поддерживают связи между различными таблицами, гарантируя, что ссылки на записи в других таблицах остаются корректными и актуальными;

– ограничения уникальности (Unique Constraints): Предотвращают дублирование данных в полях, которые должны быть уникальными, например, адрес электронной почты пользователя или номер заказа.

Транзакции обеспечивают атомарность операций над данными, что означает, что все изменения, внесенные в рамках транзакции, либо полностью выполняются, либо полностью откатываются в случае ошибки. Это особенно важно при обработке критически значимых операций, таких как платежи и оформление заказов, где согласованность данных имеет первостепенное

значение. Использование транзакций позволяет избежать частичных обновлений данных и сохраняет их целостность.

Применение индексов существенно ускоряет выполнение запросов, особенно при работе с большими объемами данных. Индексы позволяют быстро находить и извлекать необходимые записи, что способствует повышению общей производительности системы. В данном случае, индексирование может быть применено к полям, часто используемым в поисковых запросах, таким как идентификаторы пользователей, названия товаров и статусы заказов. Дополнительно, для обеспечения устойчивости к потенциальным сбоям и предотвращения потери данных, внедряются механизмы резервного копирования и восстановления данных. Регулярное создание резервных копий позволяет восстанавливать данные в случае аппаратных сбоев, программных ошибок или других непредвиденных обстоятельств, тем самым гарантируя непрерывность бизнес-процессов и защиту ценной информации. Для обеспечения высокой производительности системы и способности эффективно обрабатывать увеличивающиеся объемы данных и растущее число пользователей, необходимо внедрить продуманные стратегии оптимизации и масштабируемости базы данных. Эти стратегии включают в себя следующие ключевые подходы:

– нормализация данных: Процесс нормализации направлен на устранение избыточности данных и повышение эффективности хранения. Это достигается путем разделения данных на логически связанные таблицы и установления правильных связей между ними. Нормализация способствует минимизации дублирования данных, упрощает их обновление и поддерживает целостность информации;

– кэширование: Внедрение кэширования позволяет значительно снизить нагрузку на основную базу данных за счет хранения часто запрашиваемых данных в быстродействующей памяти (например, в оперативной памяти или специализированных кэш-системах). Это ускоряет доступ к данным и снижает

время отклика системы, что особенно важно для пользовательских интерфейсов и приложений с высокими требованиями к скорости обработки запросов;

– шардирование: Шардирование представляет собой процесс разделения базы данных на несколько независимых частей (шардов), каждая из которых хранится на отдельном сервере или кластере серверов. Это позволяет распределить нагрузку и обеспечить горизонтальную масштабируемость системы, что особенно актуально при росте числа пользователей и объема данных. Шардирование способствует повышению отказоустойчивости и улучшению производительности за счет параллельной обработки запросов [28];

– репликация данных: Создание копий базы данных на нескольких серверах повышает отказоустойчивость и доступность данных. Репликация обеспечивает бесперебойную работу системы даже в случае отказа отдельных компонентов инфраструктуры, что критически важно для поддержания непрерывности бизнес-процессов и удовлетворения требований пользователей к доступности сервисов;

– оптимизация запросов: Анализ и оптимизация SQL-запросов позволяют снизить время выполнения операций и повысить общую производительность базы данных. Это включает в себя использование эффективных алгоритмов выполнения запросов, минимизацию количества выполняемых операций, а также применение индексов и других методов ускорения доступа к данным.

Внедрение вышеуказанных стратегий позволяет создать надежную, высокопроизводительную и масштабируемую базу данных, способную удовлетворить как текущие, так и будущие потребности системы управления заказами. Это обеспечивает устойчивость к увеличивающимся нагрузкам, поддерживает высокую скорость обработки данных и гарантирует стабильную работу системы в условиях изменяющихся требований бизнеса.

Выводы к главе 2

В представленной главе проведён детализированный анализ разработки системы управления заказами, которая играет стратегически важную роль в

условиях цифровой трансформации бизнес-процессов и роста электронной коммерции. Важно отметить, что автором обоснованы ключевые задачи, включающие повышение надёжности, гибкости и точности обработки данных, а также интеграцию с корпоративными системами, такими как CRM, что позволяет эффективно управлять ресурсами и взаимодействовать с клиентами. Существенным аспектом исследования является применение нотации BPMN для моделирования бизнес-процессов. Это способствовало выявлению узких мест и оптимизации этапов, таких как выбор товара, оформление и доставка заказа, что улучшает клиентский сервис и повышает операционную эффективность. На основе BPMN-диаграмм выделены ключевые сервисы системы, включая управление каталогом товаров, корзиной покупок, заказами, платежами и доставкой. Это соответствует принципам сервис-ориентированной архитектуры, которая обеспечивает модульность, масштабируемость и надёжность системы. Особое внимание уделено структурному проектированию системы с использованием UML-диаграмм. Описаны классы, их атрибуты и методы, что обеспечивает ясность и согласованность при реализации функциональности системы. Диаграммы потоков данных и последовательностей визуализируют движение информации, уточняя взаимодействие между компонентами. Значимым результатом является проектирование базы данных, отражающее потребности системы. Применение нормализации, индексации и шардирования данных позволило оптимизировать производительность и масштабируемость базы данных. Также внедрены механизмы резервного копирования, что обеспечивает устойчивость к сбоям. В заключение, разработанная система управления заказами представляет собой интегрированное решение, способное адаптироваться к изменяющимся бизнес-требованиям, обеспечивая при этом высокий уровень клиентского сервиса. Применение современных технологий и методик проектирования делает её надёжным инструментом для цифровой трансформации предприятий и укрепления их конкурентоспособности на рынке.

Глава 3 Декомпозиция системы на сервисы и реализация сервис-ориентированной архитектуры

3.1 Принципы декомпозиции на сервисы

В современном мире разработки программного обеспечения и построения корпоративных информационных систем, одним из фундаментальных аспектов архитектурного проектирования является декомпозиция системы на отдельные сервисы. Это позволяет создать гибкую, масштабируемую и легко поддерживаемую архитектуру, которая способна оперативно реагировать на динамично меняющиеся бизнес-требования и технологические инновации. Данный раздел посвящён рассмотрению ключевых принципов и методологий, применяемых при декомпозиции системы на сервисы [34].

Для эффективной реализации сервис-ориентированной архитектуры, которая предполагает разбиение системы на автономные компоненты, называемые сервисами, необходимо учитывать ряд критических критериев. Эти критерии обеспечивают оптимальную структуризацию системы и позволяют достичь высокого уровня производительности, надёжности и адаптивности. Рассмотрим основные из них более подробно:

– когезия. Высокая степень когезии внутри сервиса подразумевает, что все его компоненты тесно связаны и направлены на выполнение общей функциональности или бизнес-цели. Это способствует минимизации внутренней сложности сервиса и облегчает управление внутренними зависимостями. Высокая когезия повышает модульность и упрощает процессы разработки, тестирования и сопровождения, так как изменения внутри сервиса менее вероятно повлияют на другие компоненты системы;

– слабая связанность. Принцип слабой связанности между сервисами подразумевает минимизацию зависимостей между ними, что достигается посредством чётко определённых интерфейсов и протоколов взаимодействия. Каждый сервис должен быть максимально изолирован от остальных, исключая

прямую зависимость от внутренней реализации других сервисов. Это позволяет независимо изменять или заменять сервисы без риска нарушения работы системы в целом, повышая её устойчивость и адаптивность к изменениям;

– бизнес-ориентированность. Декомпозиция системы должна быть тесно связана с бизнес-целями организации. Каждый сервис должен решать конкретную задачу, соответствующую одному или нескольким бизнес-процессам. Такая ориентация обеспечивает согласованность между технической архитектурой и стратегическими целями бизнеса, что повышает эффективность решений и способствует достижению конкурентных преимуществ на рынке;

– автономность. Автономность сервисов подразумевает их независимость в выполнении своих функций. Это позволяет развивать, тестировать, развертывать и масштабировать сервисы отдельно от других компонентов системы. Автономные сервисы снижают сложность управления системой и повышают её устойчивость, так как сбой в одном сервисе не приводит к отказу всей системы;

– масштабируемость. С учётом растущих требований к производительности и объёмам обрабатываемых данных, важно, чтобы сервисы могли масштабироваться независимо. Это может быть достигнуто путём добавления вычислительных ресурсов или оптимизации производительности для высоконагруженных компонентов. Независимая масштабируемость сервисов позволяет эффективно распределять ресурсы и поддерживать высокий уровень обслуживания пользователей;

– возможно повторного применения. Сервисы должны быть спроектированы с учётом возможности их повторного использования в различных бизнес-контекстах. Это достигается путём создания универсальных и гибких интерфейсов, а также отделения бизнес-логики от специфических реализаций. Повторное использование сокращает время и затраты на разработку новых функций, повышая общую эффективность ИТ-инфраструктуры.

Таким образом, применение данных критериев при декомпозиции системы на сервисы позволяет сформировать высокоэффективную архитектуру. Она удовлетворяет требованиям к гибкости, модульности и масштабируемости, а также обеспечивает соответствие стратегическим бизнес-целям организации. Это особенно важно в условиях быстро меняющейся рыночной конъюнктуры и технологического прогресса. В контексте декомпозиции сложных систем на сервисы, применение подхода доменно-ориентированного дизайна зарекомендовало себя как один из наиболее эффективных методов, доменно-ориентированный дизайн фокусируется на глубоком понимании предметной области и моделировании системы таким образом, чтобы она точно отражала бизнес-процессы и правила организации. Рассмотрим основные аспекты этого подхода:

– определение предметной области. На первоначальном этапе необходимо провести всесторонний анализ бизнес-контекста, чтобы выделить основные области деятельности организации. Это включает в себя детальное изучение бизнес-процессов, требований пользователей, правил и ограничений, а также используемой терминологии. Например, в банковской сфере предметными областями могут быть управление счетами, обработка транзакций, управление рисками и обеспечение соответствия нормативным требованиям. Глубокое понимание предметной области позволяет создать модели, отражающие реальные бизнес-сценарии и обеспечивающие релевантность системы для конечных пользователей;

– разделение на под-домены. Каждая предметная область подразделяется на под-домены для упрощения управления сложностью и фокусировки на специфических функциях. Под-домены классифицируются как основные, поддерживающие или общие. Основные под-домены являются критическими для уникальных бизнес-функций и стратегического развития организации. Поддерживающие под-домены обеспечивают необходимые вспомогательные функции, а общие под-домены включают стандартные для отрасли или технологические компоненты.

Такое структурирование позволяет эффективно распределять ресурсы разработки и оптимизировать архитектуру системы;

– моделирование границ контекста.

Для каждого под-домена определяется граница контекста, которая устанавливает чёткие пределы применения конкретной модели и терминологии. Границы контекста служат для изоляции моделей и предотвращения конфликтов между различными частями системы. Это особенно важно в крупных проектах с множественными командами, где согласованность и однозначность терминов критически важны для успешной коммуникации и интеграции компонентов;

– определение агрегатов**.**

Внутри границ контекста выделяются агрегаты — группы связанных объектов, которые рассматриваются как единое целое с точки зрения бизнес-логики и согласованности данных. Агрегаты устанавливают границы транзакций и определяют инварианты, которые должны поддерживаться при любых операциях. Правильное определение агрегатов способствует поддержанию целостности данных и упрощает управление состоянием системы, особенно в распределённых и высоконагруженных средах. Применение доменно-ориентированного дизайна способствует созданию структурированной и эволюционирующей архитектуры, где сервисы естественным образом соответствуют бизнес-процессам и требованиям. Это упрощает управление сложностью системы, повышает качество программного обеспечения и ускоряет адаптацию к изменениям в бизнес-среде. Кроме того, доменно-ориентированный дизайн способствует улучшению взаимодействия между техническими специалистами и бизнес-экспертами за счёт использования единой терминологии и моделей, что повышает эффективность коммуникации и снижает риски недопонимания.

3.2 Реализация сервисов с использованием современных технологий

В современном электронном бизнесе системы управления заказами играют критически важную роль, обеспечивая эффективное взаимодействие между клиентами и компаниями. Исходя из ранее описанных принципов декомпозиции на сервисы, необходимо выделить и детально описать основные сервисы системы управления заказами. Данная декомпозиция позволит оптимизировать процессы, повысить масштабируемость и гибкость системы, а также упростить её дальнейшую разработку и сопровождение.

Сервис каталога товаров
Функциональное назначение:

Прежде всего, сервис каталога товаров служит центральным узлом для хранения, управления и предоставления информации о товарах, доступных для покупки в системе. Он обеспечивает клиентам доступ к обширному ассортименту продукции, позволяя им детально ознакомиться с характеристиками товаров, сравнивать их, а также фильтровать и сортировать по различным критериям для принятия оптимального решения о покупке.

Основные функции:

- управление информацией о товарах**:** включает хранение и обновление таких данных, как название товара, подробное описание, актуальная цена, высококачественные изображения, технические характеристики и отзывы клиентов;
- категоризация товаров**:** обеспечивает структурирование ассортимента по категориям, подкатегориям и тегам, что упрощает навигацию и поиск необходимых товаров для пользователей;
- поиск и фильтрация**:** предоставляет расширенные возможности поиска, по ключевым словам, а также фильтрацию товаров по заданным параметрам, таким как цена, бренд, рейтинг и другие атрибуты;

– обновление информации в реальном времени**:** гарантирует, что все изменения в ассортименте, ценах или наличии товаров моментально отражаются в каталоге, обеспечивая актуальность данных для клиентов.

Технические аспекты:

– оптимизированная база данных**:** использование высокопроизводительной базы данных, оптимизированной для операций чтения, что обеспечивает быстрый доступ к данным каталога;

– кэширование данных**:** внедрение механизмов кэширования для часто запрашиваемой информации с целью повышения производительности системы и уменьшения нагрузки на базу данных;

– restful API**:** предоставление стандартизированного API-интерфейса для доступа к данным каталога, что облегчает интеграцию с другими сервисами и внешними приложениями;

– масштабируемость**:** обеспечение горизонтальной масштабируемости сервиса для поддержания высокой производительности при увеличении количества пользователей и объёма данных.

Сервис корзины покупок

Функциональное назначение:

Сервис корзины покупок играет ключевую роль в процессе формирования заказа, предоставляя клиентам удобный механизм временного хранения выбранных товаров. Он позволяет пользователям накапливать товары, изменять их количество и управлять содержимым корзины перед окончательным оформлением заказа.

Основные функции:

– добавление и удаление товаров**:** предоставляет возможность легко добавлять выбранные товары в корзину и удалять их при необходимости;

– изменение количества товаров**:** позволяет пользователям изменять количество единиц каждого товара в корзине, автоматически обновляя общую стоимость;

– подсчет общей стоимости**:** осуществляет автоматический расчет общей стоимости товаров в корзине с учётом количества и цен, включая возможные скидки и налоги;

– сохранение состояния корзины**:** обеспечивает сохранение содержимого корзины между сессиями пользователя, что особенно важно для незарегистрированных пользователей или при использовании нескольких устройств.

Технические аспекты:

– сессионное хранилище**:** использование сессионных данных для хранения информации о корзине незарегистрированных пользователей, с возможностью переноса данных при регистрации или авторизации;

– интеграция с сервисом управления пользователями**:** для авторизованных пользователей данные корзины могут храниться в привязке к их учетной записи, обеспечивая доступ с различных устройств;

– обеспечение целостности данных**:** обработка конкурентного доступа и предотвращение возможных конфликтов при одновременном обновлении корзины;

– ux-оптимизация**:** обеспечение быстрого отклика и удобного интерфейса для управления корзиной, повышая удовлетворенность пользователей.

Сервис оформления заказов

Функциональное назначение:

Сервис оформления заказов является критическим компонентом, отвечающим за преобразование содержимого корзины в официальный заказ, готовый для дальнейшей обработки. Он обеспечивает сбор и валидацию всей необходимой информации, необходимой для успешного выполнения заказа.

Основные функции:

– проверка корзины**:** проверка наличия товаров на складе, актуальности цен и применения возможных скидок перед оформлением заказа;

– сбор информации об оплате и доставке**:** предоставление пользователю вариантов выбора способа оплаты и доставки, сбор необходимых данных для выполнения транзакции и доставки заказа;

– генерация заказа**:** создание уникального идентификатора заказа, фиксирование всех деталей и подготовка данных для дальнейшей обработки в системе;

– уведомления**:** отправка подтверждения заказа клиенту через электронную почту, SMS или другие каналы, информирование о статусе заказа.

Технические аспекты:

– транзакционная обработка**:** обеспечение атомарности операций при оформлении заказа, использование транзакций для предотвращения несогласованности данных;

– интеграция с внешними сервисами**:** взаимодействие с сервисами платежей для обработки оплаты и с сервисом доставки для расчета и планирования доставки;

– логирование и мониторинг**:** подробное ведение журналов операций для отслеживания и анализа процесса оформления заказов, что важно для выявления и устранения возможных проблем;

– обеспечение безопасности**:** защита данных клиента и обеспечение соответствия нормативным требованиям в области обработки персональных и финансовых данных.

Сервис платежей

Функциональное назначение:

Сервис платежей отвечает за безопасную и надежную обработку финансовых транзакций, связанных с оплатой заказов. Он обеспечивает интеграцию с различными платежными системами и гарантирует защиту конфиденциальных данных клиентов.

Основные функции:

– интеграция с платежными шлюзами**:** взаимодействие с различными платежными системами и банками, поддержка множества способов оплаты для удобства клиентов;

– обработка платежей**:** верификация и авторизация платежных данных, проведение транзакций, обеспечение корректности и своевременности платежей;

– управление транзакциями**:** обработка возвратов, отмен транзакций, управление удержаниями и другими финансовыми операциями;

– обеспечение безопасности**:** применение современных методов шифрования и защиты данных для предотвращения мошенничества и несанкционированного доступа;

Технические аспекты:

– соответствие стандартам безопасности**:** полное соответствие требованиям стандарта PCI DSS для обработки платежных карт и другим регуляторным нормам;

– шифрование данных**:** использование SSL/TLS для защиты данных при передаче и хранения конфиденциальной информации в зашифрованном виде;

– обработка ошибок**:** разработка механизмов обработки ошибок и отказоустойчивости, обеспечение бесперебойной работы сервиса даже при сбоях в связанных системах;

– мониторинг транзакций**:** постоянный мониторинг и анализ транзакционной активности для обнаружения подозрительных операций и предотвращения мошенничества.

Сервис доставки

Функциональное назначение:

Сервис доставки управляет всеми аспектами, связанными с доставкой заказов клиентам. Он обеспечивает расчет стоимости доставки, выбор оптимальных вариантов, взаимодействие с логистическими компаниями и информирование клиентов о статусе доставки.

Основные функции:

– расчет стоимости доставки**:** вычисление стоимости на основе веса, габаритов заказа, расстояния до пункта назначения и выбранного способа доставки;

– выбор вариантов доставки**:** предоставление клиентам различных опций, таких как стандартная доставка, экспресс-доставка, самовывоз или пункты выдачи заказов;

– интеграция с логистическими партнерами**:** автоматизация передачи заказов в системы логистических компаний, отслеживание статуса отправлений в реальном времени;

– уведомления клиентов**:** информирование клиентов о статусе доставки через электронную почту, SMS или мобильные приложения, предоставление трекинг-номеров и обновлений.

Технические аспекты:

– api-интеграция**:** использование API для взаимодействия с системами логистических партнеров, обеспечение автоматизированного обмена данными;

– обработка данных геолокации**:** использование геоинформационных сервисов для определения оптимальных маршрутов и расчета сроков доставки;

– масштабируемость и производительность**:** способность обрабатывать большой объем запросов, особенно в пиковые периоды, такие как праздничные сезоны;

– отчетность и аналитика**:** сбор данных о доставке для анализа эффективности логистики, выявления узких мест и оптимизации процессов.

Сервис управления пользователями

Функциональное назначение:

Сервис управления пользователями является фундаментальным компонентом системы, отвечающим за аутентификацию, авторизацию и управление учетными записями пользователей. Он обеспечивает безопасность доступа и персонализацию опыта взаимодействия с системой.

Основные функции:

– регистрация и аутентификация: позволяет новым пользователям создавать учетные записи, а существующим — безопасно входить в систему;

– управление профилями: предоставляет пользователям возможность обновлять личные данные, сохранять адреса доставки, управлять платежной информацией и настройками предпочтений;

– безопасность учетных записей: включает функции восстановления пароля, настройку двухфакторной аутентификации и оповещения о подозрительной активности;

– управление доступом: определение ролей и прав доступа для различных категорий пользователей, включая администраторов, менеджеров и клиентов.

Технические аспекты:

– безопасное хранение данных: использование алгоритмов хеширования и соления для хранения паролей, защита персональных данных в соответствии с лучшими практиками;

– токены аутентификации: применение JWT (JSON Web Tokens) для безопасной аутентификации запросов и поддержки без серверной архитектуры;

– соответствие нормативным требованиям: обеспечение соответствия требованиям GDPR и других регуляторных актов в части сбора, хранения и обработки персональных данных;

– скалируемость и отказоустойчивость: разработка архитектуры, способной выдерживать высокий уровень нагрузки и обеспечивать бесперебойный доступ пользователей к своим учетным записям.

В процессе разработки современной системы управления заказами необходимо тщательно подойти к выбору технологий и инструментов, которые будут использоваться для реализации сервисов. Это решение оказывает существенное влияние на эффективность разработки, качество конечного продукта и его способность удовлетворять бизнес-требования.

Критерии выбора технологий:

– соответствие функциональным требованиям: Технологии должны полностью поддерживать необходимый функционал, обеспечивая возможность реализации всех предусмотренных бизнес-процессов. Это включает в себя поддержку необходимых протоколов, интеграцию с внешними сервисами и соответствие отраслевым стандартам;

– обеспечение производительности и масштабируемости: Выбранные инструменты должны обеспечивать высокую производительность системы при обработке больших объемов данных и поддерживать горизонтальное и вертикальное масштабирование. Это критично для обеспечения бесперебойной работы системы при росте числа пользователей и объема транзакций;

– поддержка сообщества и наличие ресурсов для разработки: Широкое сообщество разработчиков и обширная база знаний облегчают процесс разработки и решения возникающих проблем. Это также гарантирует долгосрочную поддержку и обновления выбранных технологий;

Рекомендуемые технологии и инструменты:

– часть ссерверной разработки: для использования был выбран фреймворк Spring Boot. Это комплексный набор инструментов для разработки производительных и надежных веб-сервисов на языке Java. Обеспечивает простоту конфигурации, интеграцию с различными модулями и поддержку микросервисной архитектуры;

– базы данных: рекомендуется использовать PostgreSQL как основную реляционную базу данных для хранения структурированных данных. Она отличается высокой производительностью, надежностью и поддержкой сложных транзакций и запросов;

– контейнеризация: для контейнеризации был выбран Docker. Он используется для контейнеризации приложений, что обеспечивает их изоляцию, переносимость и упрощает процесс развертывания;

А также был использован Kubernetes. Он предоставляет средства для оркестрации контейнеров, автоматизируя развертывание, масштабирование и управление приложениями, находящиеся в контейнере, в кластере.

Разработка и развертывание сервисов:

– сервис-ориентированная архитектура: применение сервис-ориентированной архитектуры позволяет разделить систему на независимые сервисы, каждый из которых отвечает за определенную бизнес-функцию. Это повышает гибкость разработки, облегчает масштабирование отдельных компонентов и упрощает поддержку и обновление системы в целом;

– непрерывная интеграция и развертывание: использование подходов непрерывной интеграции и развертывания обеспечивает автоматизацию процессов сборки, тестирования и развертывания приложений. Инструменты такие как GitLab CI/CD позволяют настроить конвейеры, которые ускоряют выпуск новых версий, повышают качество кода и снижают риск ошибок при развертывании;

Безопасность:

– аутентификация и авторизация: реализация протоколов OAuth 2.0 обеспечивает надежную аутентификацию и авторизацию пользователей, позволяя использовать единые стандарты безопасности и интегрироваться с различными провайдерами идентификации;

– шифрование данных: для защиты данных в транзите используется SSL/TLS, обеспечивая безопасное соединение между клиентом и сервером. Данные в покое должны быть зашифрованы с использованием встроенных механизмов PostgreSQL или сторонних решений для шифрования на уровне диска.

Производительность:

– кэширование: внедрение Redis позволяет кэшировать часто запрашиваемые данные, снижая нагрузку на базу данных и ускоряя время отклика системы;

– оптимизация баз данных: использование индексов, анализ и оптимизация SQL-запросов повышают эффективность работы с базой данных, уменьшая время выполнения операций чтения и записи.

Отказоустойчивость:

– автоматическое масштабирование: настройка автоматического масштабирования сервисов позволяет системе динамически адаптироваться к изменениям нагрузки, обеспечивая стабильную работу при пиковых нагрузках.

– репликация и балансировка нагрузки: репликация данных в PostgreSQL и использование балансировщиков нагрузки повышают доступность системы и предотвращают точку отказа, обеспечивая непрерывность бизнес-процессов.

3.5 Практическая реализация сервисов системы управления заказами

Разработка сервиса каталога товаров, сервиса корзины покупок, сервиса оформления заказов, сервиса платежей, сервиса доставки, сервиса управления пользователями.

а) модель товара (Product):

1) id: Уникальный идентификатор товара (тип данных: UUID),

2) название (name): Название товара (тип данных: String),

3) описание (description): Подробное описание товара (тип данных: Text),

4) цена (price): Стоимость товара (тип данных: BigDecimal),

5) категория (category): Ссылка на категорию товара (тип данных: Foreign Key),

6) наличие на складе (stock): Количество доступных единиц на складе (тип данных: Integer),

7) тзображения (images): Список URL-адресов изображений товара (тип данных: List<String>);

б) модель категории (Category):

1) id: Уникальный идентификатор категории (тип данных: UUID),

2) название (name): Название категории (тип данных: String),

3) родительская категория (parentCategory): Ссылка на родительскую категорию для создания иерархии (тип данных: Foreign Key);

в) api-эндпойнты:

1) get /products: Получение списка товаров с возможностью фильтрации по категориям, цене и другим параметрам,

2) get /products/{id}: Получение подробной информации о конкретном товаре по его идентификатору,

3) post /products: Добавление нового товара в каталог (доступно только администраторам),

4) put /products/{id}: Обновление информации о товаре (доступно только администраторам),

5) delete /products/{id}: Удаление товара из каталога (доступно только администраторам);

г) технологии:

1) язык программирования: Java с использованием Spring Boot для построения RESTful API,

2) база данных: PostgreSQL для надежного хранения структурированных данных о товарах и категориях,

3) кэширование: Внедрение Redis для кэширования результатов запросов и улучшения производительности при доступе к часто запрашиваемым данным,

4) взаимодействие с базой данных: Использование Spring Data JPA для упрощения работы с базой данных через объектно-реляционное отображение (ORM);

д) дополнительные аспекты:

1) валидация данных: Применение аннотаций валидации для обеспечения корректности вводимых данных,

2) обработка исключений: Глобальный обработчик исключений для предоставления информативных сообщений об ошибках,

3) документация API: Автоматическая генерация документации с помощью Swagger/OpenAPI для облегчения интеграции с клиентскими приложениями;

е) модель корзины (Cart):

1) id пользователя (userId): Идентификатор пользователя, владеющего корзиной (тип данных: UUID),

2) список товаров (items): Коллекция объектов товара в корзине,

ж) модель товара в корзине (CartItem):

1) товар (product): Ссылка на объект товара из каталога (тип данных: Product),

2) количество (quantity): Количество выбранного товара (тип данных: Integer);

и) api-эндпойнты:

1) get /cart: Получение текущей корзины пользователя,

2) post /cart/items: Добавление товара в корзину,

3) put /cart/items/{productId}: Изменение количества товара в корзине,

4) delete /cart/items/{productId}: Удаление товара из корзины;

к) технологии:

1) язык программирования: Java с использованием Spring Boot для создания RESTful сервисов,

2) хранилище данных: Использование Redis для хранения состояния корзины в памяти, обеспечивая быстрый доступ и обновление данных,

3) сессии пользователя: Реализация механизма сессий или токенов для идентификации пользователя и его корзины;

л) дополнительные аспекты:

1) согласованность данных: Проверка наличия товара и его доступного количества при добавлении в корзину,

2) обработка конкуренции: Механизмы блокировок или атомарных операций для предотвращения конфликтов при одновременном доступе;

м) модель заказа (Order):

1) id: Уникальный идентификатор заказа (тип данных: UUID),

2) пользователь (user): Ссылка на пользователя, сделавшего заказ (тип данных: Foreign Key),

3) список товаров (orderItems): Коллекция заказанных товаров с количеством и ценой,

4) общая стоимость (totalAmount): Итоговая сумма заказа (тип данных: BigDecimal),

5) статус (status): Текущий статус заказа (тип данных: Enum),

6) дата создания (createdAt): Дата и время создания заказа (тип данных: Timestamp);

н) модель статуса заказа (OrderStatus):

1) значения: CREATED, PAID, PROCESSING, SHIPPED, DELIVERED, CANCELLED;

2) п) api-эндпойнты:

3) post/orders: Создание нового заказа на основе содержимого корзины пользователя,

4) get/orders/{id}: Получение информации о конкретном заказе по его идентификатору,

5) get/orders: Получение списка заказов текущего пользователя с возможностью фильтрации по статусу и дате;

р) технологии:

1) язык программирования: Java с использованием Spring Boot,

2) база данных: PostgreSQL для хранения данных о заказах с поддержкой транзакций и сложных запросов,

3) взаимодействие с другими сервисами: Интеграция с сервисами платежей и доставки посредством REST API или очередей сообщений;

с) дополнительные аспекты:

1) транзакционность: Обеспечение атомарности операций при создании заказа и списании товаров со склада,

2) уведомления: Отправка уведомлений пользователю о смене статуса заказа через email или уведомления через телефон;

т) функциональность:

1) интеграция с платежными шлюзами: Поддержка популярных платежных систем, для обработки платежей,

2) обработка транзакций: Инициирование платежей, обработка ответов от платежных систем и обновление статуса заказа в зависимости от результатов транзакции;

у) api-эндпойнты:

1) post/payments: Инициирование платежа для конкретного заказа,

2) get/payments/{paymentId}/status: Получение текущего статуса платежа по его идентификатору;

ф) технологии:

1) язык программирования: Java с использованием Spring Boot для обеспечения надежности и безопасности при обработке финансовых данных;

х) безопасность:

1) SSL/TLS: Шифрование всех соединений для защиты данных,

2) соблюдение стандартов PCI DSS: Соответствие требованиям стандарта безопасности данных индустрии платежных карт для обработки, хранения и передачи данных держателей карт;

ц) дополнительные аспекты:

1) обработка ошибок: Детальная обработка ошибок и непредвиденных ситуаций при взаимодействии с платежными системами,

2) логирование и мониторинг: Ведение журналов транзакций и настройка мониторинга для своевременного обнаружения и решения проблем;

ш) функциональность:

1) расчет стоимости и сроков доставки: Предоставление пользователю информации о возможных вариантах доставки на основе его адреса и доступных логистических партнеров,

2) создание заявок на доставку: Автоматическое создание и отправка заказов в системы логистических компаний;

щ) api-эндпойнты:

1) post/delivery/options: Запрос вариантов доставки для указанного адреса и заказа,

2) post/delivery/orders: Создание заявки на доставку в выбранной логистической компании,

3) get/delivery/{orderId}/status: Отслеживание статуса доставки по идентификатору заказа;

э) технологии:

1) язык программирования: Java с использованием Spring Boot,

2) интеграция с внешними API: Взаимодействие с API логистических партнеров для обмена информацией о доставке;

ю)идополнительные аспекты:

1) кэширование географических данных: Использование кэша для ускорения доступа к информации о зонах доставки и тарифах,

2) обработка асинхронных событий: Использование очередей сообщений для обработки уведомлений о статусе доставки;

я) модель пользователя (User):

1) id: Уникальный идентификатор пользователя (тип данных: UUID),

2) имя (firstName, lastName): Личная информация пользователя (тип данных: String),

3) email: Адрес электронной почты (тип данных: String, уникальный),

4) пароль (password): Хэшированный пароль пользователя (тип данных: String),

5) holи (roles): Список ролей пользователя (тип данных: List<String>),

6) адреса (addresses): Коллекция адресов доставки пользователя;

a) api-эндпойнты:

1) post/users/register: Регистрация нового пользователя,

2) post/users/login: Аутентификация пользователя и выдача токена доступа,

3) get/users/profile: Получение информации о текущем пользователе,

4) put/users/profile: Обновление информации профиля пользователя;

b) технологии:

1) язык программирования: Java с использованием Spring Boot и модуля Spring Security для реализации механизмов безопасности;

c) безопасность:

2) JWT (JSON Web Tokens): Использование JWT для аутентификации и авторизации, позволяя безопасно передавать информацию о пользователе между клиентом и сервером,

3) Хэширование паролей: Применение алгоритмов хэширования, таких как bcrypt или Argon2, для безопасного хранения паролей;

4) дополнительные аспекты:

5) восстановление пароля: Реализация механизма сброса пароля через email,

6) верификация email: Отправка подтверждающих писем при регистрации для проверки действительности адреса электронной почты,

7) защита от атак: Внедрение ограничений на количество попыток входа для предотвращения атак перебора паролей.

3.3 Пример практической реализации: система управления заказами

Для иллюстрации рассмотрим реальный пример внедрения одного из сервисов сервис-ориентированной архитектуры в компании, специализирующейся на продаже электроники через интернет.

Бизнес-требования:

- обеспечить масштабируемость системы для обработки растущего числа заказов;

– улучшить время отклика и опыт взаимодействия пользователей с сайтом;

– упростить интеграцию с внешними сервисами платежей и доставки.

Декомпозиция на сервисы:

– выделены сервисы в соответствии с ранее описанными: каталог товаров, корзина покупок, оформление заказов, платежи, доставка, управление пользователями;

– определение границ контекста;

– каждый сервис имеет четко определенную область ответственности и взаимодействует с другими через стандартизированные интерфейсы;

– сервис оформления заказов.

Сервис оформления заказов отвечает за создание заказов на основе данных из корзины, управление статусами заказов и взаимодействие с сервисами платежей, и доставки.

Структура проекта:

– Сущности: Order, OrderItem;

– Репозитории: OrderRepository, OrderItemRepository;

– Сервисы: OrderService;

– Контроллеры: OrderController.

Сущность «Order» описывает сам заказ, номер заказа, сумму заказа и статус заказа. На рисунке 7 представлен код сущности.

```java
@Entity
@Table(name = "orders")
@Data
public class Order {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private Long userId;
  private BigDecimal totalAmount;
  private String status; // Например, NEW, PAID, SHIPPED, COMPLETED

  @OneToMany(mappedBy = "order", cascade = CascadeType.ALL)
  private List<OrderItem> items;
```

Рисунок 7 – код сущности Order

Сущность «Orderitem» описывает позицию заказа и связь с сущностью Order. На рисунке 8 представлен код сущности.

```java
@Entity
@Table(name = "order_items")
@Data
public class OrderItem {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private Long productId;
  private Integer quantity;
  private BigDecimal price;

  @ManyToOne
  @JoinColumn(name = "order_id")
  private Order order;
}
```

Рисунок 8 – код сущности Orderitem

Сервис «OrderService» описывает бизнес-логику управлений заказов. На рисунке 9 представлен код бизнес-логики.

```java
@Service
public class OrderService {

    5 usages
    @Autowired
    private OrderRepository orderRepository;

    // Создание нового заказа
    1 usage
    public Order createOrder(Order order) {
        order.setStatus("NEW");
        return orderRepository.save(order);
    }

    // Получение заказа по ID
    1 usage
    public Optional<Order> getOrderById(Long id) {
        return orderRepository.findById(id);
    }

    // Обновление статуса заказа
    1 usage
    public Order updateOrderStatus(Long id, String status) {
        Order order = orderRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Order not found for this id :: " + id));
        order.setStatus(status);
        return orderRepository.save(order);
    }

    // Получение всех заказов пользователя
    1 usage
    public List<Order> getOrdersByUserId(Long userId) {
        return orderRepository.findByUserId(userId);
    }
}
```

Рисунок 9 – код бизнес-логики управления заказами

Контроллер «OrderController» обрабатывает HTTP запросы которые связаны с заказами. На рисунке 10 представлен код контроллера.

```java
@RestController
@RequestMapping("/api/v1")
public class OrderController {

    4 usages
    @Autowired
    private OrderService orderService;

    // Создание нового заказа
    @PostMapping("/orders")
    public Order createOrder( @RequestBody Order order) {
        return orderService.createOrder(order);
    }

    // Получение заказа по ID
    @GetMapping("/orders/{id}")
    public ResponseEntity<Order> getOrderById(@PathVariable(value = "id") Long orderId) {
        Order order = orderService.getOrderById(orderId)
                .orElseThrow(() -> new ResourceNotFoundException("Order not found for this id :: " + orderId));
        return ResponseEntity.ok().body(order);
    }

    // Обновление статуса заказа
    @PutMapping("/orders/{id}/status")
    public ResponseEntity<Order> updateOrderStatus(@PathVariable(value = "id") Long orderId,
                                                   @RequestParam String status) {
        Order updatedOrder = orderService.updateOrderStatus(orderId, status);
        return ResponseEntity.ok(updatedOrder);
    }

    // Получение заказов пользователя
    @GetMapping("/users/{userId}/orders")
    public List<Order> getOrdersByUserId(@PathVariable(value = "userId") Long userId) {
        return orderService.getOrdersByUserId(userId);
    }
}
```

Рисунок 10 – код контроллера

Репозиторий «OrderRepository» - описывает связь кода с базой данных. На рисунке 11 представлен код репозитория.

```
1 usage
public interface OrderRepository extends JpaRepository<Order, Long> {
    1 usage
    List<Order> findByUserId(Long userId);
}
```

Рисунок 11 – код репозитория

Для развёртывания сервисов было решено воспользоваться сервисом контейнеризации «Docker». На рисунке 12 представлено изображение контейнера сервис-ориентированного приложения управления заказов.

Рисунок 12 – Контейнер сервис-ориентированного приложения

Бизнес-требования, такие как масштабируемость, улучшение времени отклика и упрощение интеграции с внешними сервисами, были эффективно реализованы через декомпозицию системы на специализированные сервисы: каталог товаров, корзина, оформление заказов, платежи, доставка и управление пользователями. Каждый сервис обладает четко определенной областью ответственности и взаимодействует с другими через стандартизированные интерфейсы, что обеспечивает гибкость и надежность системы. Особое

внимание уделено сервису оформления заказов, который управляет созданием и статусами заказов, а также взаимодействует с платежными и доставочными сервисами. Применение принципов многоразовости, абстрагирования, формальных спецификаций и соответствия функциональности потребностям пользователей способствует созданию устойчивой и легко масштабируемой архитектуры, способной эффективно поддерживать рост бизнеса и улучшать пользовательский опыт.

Выводы к главе 3

В данной главе были рассмотрены принципы декомпозиции систем на сервисы и практической реализации сервис-ориентированной архитектуры. Подчёркивается важность высоких когезии и автономности сервисов, слабой связанности между ними, их бизнес-ориентированности и масштабируемости. Представлены ключевые критерии, такие как возможность повторного использования, оптимизация данных и интеграция с внешними сервисами. Также описаны этапы разработки ключевых сервисов, включая каталог товаров, корзину покупок, оформление заказов, платежи, доставку и управление пользователями. Рассматриваются аспекты безопасности, масштабируемости, производительности и отказоустойчивости. Для реализации используются современные технологии, такие как Spring Boot, PostgreSQ, Docker и Kubernetes. Практическая часть иллюстрирует внедрение сервисов с соблюдением принципов декомпозиции. Примером служит система управления заказами, где каждая функциональная область автономна и интегрируется через стандартизированные API. Подход демонстрирует гибкость, устойчивость к изменениям и соответствие бизнес-требованиям.

Глава 4 Оценка эффективности предложенного подхода и перспективы развития

4.1 Методология оценки эффективности

В целях проведения детального и всестороннего анализа эффективности предложенной методологии был применён количественный подход, основанный на тщательном сравнении ключевых показателей до и после внедрения соответствующих решений. Для обеспечения объективности и точности результатов были собраны и проанализированы статистические данные, представленные в формате таблицы, где наглядно отражены изменения по основным параметрам производительности и качественным характеристикам работы системы. Такой подход позволяет не только оценить текущую эффективность, но и выявить потенциал для дальнейшего совершенствования бизнес-процессов. В процессе комплексной оценки эффективности системы были учтены следующие критически важные критерии, каждый из которых играет существенную роль в обеспечении высококачественного функционирования и удовлетворённости клиентов. Все важные критерии оценивания представлена в таблице 2:

Таблица 2 – Результаты оценивания эффективности внедрения сервис-ориентированной архитектуры

Параметр	До внедрения методологии	После внедрения методологии	Предполагаемое улучшение
Время обработки заказа	15 минут	12 минут	-20%
Время подтверждения наличия товара	5 минут	2 минуты	-60%
Число одновременно обрабатываемых заказов	100 заказов в час	250 заказов в час	+150%
Процент успешных транзакций	95%	98%	+3%
Среднее время ответа на запрос клиента	10 секунд	4 секунды	-60%
Скорость интеграции с внешними системами (например, платежными шлюзами)	2 недели	3 дня	-85%
Качество обслуживания (по отзывам пользователей)	80% положительных отзывов	90% положительных отзывов	+12,5%
Время на масштабирование системы (добавление нового сервиса)	3 месяца	1 месяц	-66%

– время обработки заказа: Оценивалось общее время, затраченное на выполнение полного цикла операций, необходимых для обработки одного заказа. Это включает в себя все этапы от момента размещения заказа клиентом до его окончательного выполнения, включая подтверждение, обработку платежей, сборку и подготовку к отгрузке. Сокращение этого времени напрямую влияет на оперативность обслуживания и конкурентоспособность компании на рынке;

– скорость подтверждения наличия товара: Измерялось время между поступлением запроса на наличие определённого товара и моментом предоставления клиенту подтверждения о его доступности. Высокая скорость в этом процессе важна для поддержания удовлетворённости клиентов и предотвращения потенциальных потерь продаж из-за задержек или неточностей в информации о наличии товаров;

– пропускная способность: Анализировалось максимальное количество заказов, которое система способна обработать одновременно в течение одного часа без снижения качества обслуживания. Этот показатель является ключевым для оценки производительности системы, особенно в условиях пиковых нагрузок или сезонных всплесков спроса;

– процент успешных транзакций: Рассчитывалась доля транзакций, завершённых без технических ошибок, сбоев или отмен. Высокий процент успешных транзакций свидетельствует о надёжности системы и эффективности интегрированных процессов, что положительно влияет на доверие клиентов и репутацию бренда;

– среднее время ответа на запрос клиента: Измерялась скорость обработки различных пользовательских запросов, включая обращения в службу поддержки, запросы информации о продуктах или услугах. Быстрое и эффективное реагирование на запросы клиентов является критически важным для поддержания высокого уровня обслуживания и укрепления лояльности клиентов;

– скорость интеграции с внешними системами: Оценивалось время, необходимое для подключения и настройки новых платёжных шлюзов, логистических сервисов или других внешних систем и сервисов. Эффективная интеграция позволяет расширять функциональность системы, улучшать клиентский опыт и быстро адаптироваться к изменениям рыночных условий;

– качество обслуживания: Уровень удовлетворённости пользователей измерялся на основе анализа доли положительных отзывов, рейтингов и показателей повторных обращений. Высокое качество обслуживания способствует повышению доверия клиентов, увеличению их лояльности и стимулированию рекомендаций, что в целом усиливает позиции компании на рынке;

– время на масштабирование системы: Оценивалась скорость внедрения новых сервисов, модулей или функциональных возможностей в существующую систему. Способность быстро и безболезненно масштабировать систему является важным фактором для поддержания конкурентоспособности, позволяя компании оперативно реагировать на рост спроса и расширять спектр предлагаемых услуг без существенных задержек или затрат.

4.2 Результаты Тестирования и Анализ

Улучшение производительности В результате проведённого сравнительного анализа показателей системы до и после внедрения разработанной методологии, были зафиксированы значительные улучшения в области производительности. В частности, время обработки одного заказа сократилось с первоначальных 15 минут до 12 минут, что представляет собой снижение на 20%. Этот эффект обусловлен комплексной оптимизацией внутренних процессов обработки данных и внедрением автоматизации ключевых операций, что позволило устранить избыточные шаги и минимизировать человеческий фактор. Более того, время подтверждения наличия товара на складе уменьшилось на 60%, сократившись с 5 минут до 2 минут. Данное улучшение стало возможным благодаря интеграции механизма

обновления данных о товарных запасах в режиме реального времени. Это обеспечило своевременное и точное отражение информации о доступности товаров, что, в свою очередь, ускорило процесс принятия решений и повысило удовлетворённость клиентов. На рисунке 13 изображена диаграмма статистика показателей до и после внедрения сервис-ориентированной архитектуры.

Рисунок 13 – показатели до и после внедрения сервис-ориентированной архитектуры

Повышение пропускной способности, анализ производительности системы также выявил существенное увеличение её пропускной способности. Количество заказов, которые система способна обработать в течение одного часа, увеличилось с 100 до 250, демонстрируя рост на впечатляющие 150%. Такой результат был достигнут благодаря переходу на сервис-ориентированную архитектуру (SOA), которая обеспечивает возможность параллельной обработки большого числа запросов. Это архитектурное изменение позволило масштабировать систему более эффективно и обеспечило гибкость при добавлении новых функциональных модулей.

Увеличение стабильности транзакций С точки зрения надежности операций, процент успешных транзакций увеличился с 95% до 98%. Достигнутый рост в 3% свидетельствует о повышении стабильности и надежности системы. Это улучшение стало возможным благодаря интеграции усовершенствованных механизмов проверки данных на этапе ввода и реализации надежных протоколов обработки платежей. В результате снизилось количество ошибок и сбоев в процессе проведения транзакций, что положительно отразилось на доверии клиентов и репутации компании.

Снижение времени отклика системы, Среднее время отклика системы на клиентский запрос сократилось с 10 секунд до 4 секунд, что представляет собой снижение на 60%. Такой существенный прогресс был достигнут за счет внедрения оптимизированных интерфейсов программирования приложений (API) и реализации кэширования часто используемых данных. Оптимизация API включала в себя рефакторинг кода для повышения эффективности и уменьшения избыточности, а кэширование позволило ускорить доступ к данным, которые запрашиваются наиболее часто, тем самым снизив нагрузку на базу данных.

Улучшение интеграции с внешними системами также претерпела значительные изменения. В то время как ранее на этот процесс требовалось около двух недель, теперь интеграция занимает всего лишь 3 дня, что означает сокращение времени на 85%. Такой впечатляющий результат стал возможен благодаря разработке и внедрению стандартизированных интерфейсов взаимодействия, использующих общепринятые протоколы и форматы данных. Это не только ускорило процесс интеграции, но и снизило вероятность ошибок, связанных с несовместимостью систем.

Повышение качества обслуживания клиентов, анализ клиентской удовлетворённости показал, что доля положительных отзывов увеличилась с 80% до 90%. Этот рост на 10% свидетельствует о существенном повышении уровня качества обслуживания. Улучшение обусловлено более быстрым выполнением заказов, повышенной стабильностью системы и улучшенным

взаимодействием с клиентами через персонализированные коммуникации и оперативную поддержку. Это, в свою очередь, способствует укреплению лояльности клиентов и повышению конкурентоспособности компании на рынке.

Ускорение процесса масштабирования системы необходимое для добавления новых сервисов и функциональных модулей в систему, сократилось с 3 месяцев до 1 месяца, что эквивалентно снижению на 66%. Такой ускоренный темп масштабирования был достигнут благодаря внедрению модульной архитектуры системы и автоматизации процессов развёртывания. Модульная архитектура позволяет независимую разработку и интеграцию отдельных компонентов без необходимости существенных изменений в существующей инфраструктуре. Автоматизация развёртывания с использованием современных инструментов сократила время на тестирование и ввод в эксплуатацию новых сервисов, что повысило общую гибкость и адаптивность системы к требованиям рынка.

4.3 Перспективы развития

На основании детального анализа собранных эмпирических данных и учитывая современные тенденции в области информационных технологий и управления бизнес-процессами, перспективным направлением развития системы представляется интеграция дополнительных продвинутых аналитических инструментов. В частности, внедрение систем прогнозирования спроса, основанных на методологиях больших данных (Big Data) и машинного обучения, позволит существенно повысить точность предсказаний будущих потребностей рынка. Это, в свою очередь, обеспечит возможность автоматической оптимизации запасов, что приведёт к снижению издержек, связанных с хранением и управлением товарными ресурсами. Более того, применение таких аналитических инструментов способствует повышению эффективности цепочки поставок и ускорению оборота капитала, что является

критически важным для повышения конкурентоспособности организации в динамично развивающейся бизнес-среде. С учётом растущей сложности пользовательских запросов и стремления к повышению эффективности взаимодействия с клиентами, внедрение передовых технологий машинного обучения и искусственного интеллекта становится неотъемлемым элементом стратегического развития. Автоматизация обработки пользовательских запросов посредством интеллектуальных алгоритмов позволит не только сократить время отклика и снизить нагрузку на человеческие ресурсы, но и значительно повысить уровень персонализации предоставляемых услуг. Применение нейронных сетей и алгоритмов глубокого обучения обеспечит адаптивность системы к индивидуальным предпочтениям каждого пользователя, что повысит степень удовлетворённости клиентов и укрепит их лояльность к бренду. Кроме того, данные технологии откроют возможности для прогнозной аналитики поведения клиентов, что станет основой для разработки эффективных маркетинговых стратегий и увеличения доходности бизнеса. В условиях экспоненциального роста числа пользователей и увеличения объёмов, обрабатываемых данных, обеспечение масштабируемости и отказоустойчивости системы выходит на первый план. Использование облачных технологий и инфраструктур позволяет реализовать горизонтальное масштабирование, что обеспечивает гибкое и эффективное увеличение вычислительных ресурсов в соответствии с текущими нагрузками. Облачные платформы предоставляют встроенные механизмы балансировки нагрузки, резервирования и автоматического восстановления после сбоев, что гарантирует высокую надёжность и доступность системы. Интеграция облачных решений способствует снижению капитальных и операционных затрат, связанных с поддержанием собственной серверной инфраструктуры, и позволяет сконцентрироваться на ключевых бизнес-задачах. Таким образом, переход на облачные технологии не только повышает устойчивость и производительность системы, но и обеспечивает стратегические преимущества в условиях быстро меняющейся конкурентной среды.

Выводы к главе 4

В результате анализа внедрения сервис-ориентированной архитектуры (SOA) была зафиксирована значительная оптимизация производительности системы. Основные достижения включают сокращение времени обработки заказов на 20%, увеличение пропускной способности на 150% и повышение стабильности транзакций до 98%. Среднее время отклика системы на запросы клиентов уменьшилось на 60%, а скорость интеграции с внешними системами увеличилась на 85%. Эти изменения стали возможны благодаря внедрению стандартизированных интерфейсов, оптимизации API, кэшированию данных и использованию инструментов CI/CD. Кроме того, улучшение качества обслуживания клиентов выразилось в росте положительных отзывов с 80% до 90%, а время на масштабирование системы сократилось на 66%. Применение модульной архитектуры и современных технологий позволяет компании более гибко адаптироваться к изменениям рынка и запросам пользователей. Перспективы развития системы связаны с интеграцией технологий искусственного интеллекта и машинного обучения, а также использованием облачных решений для повышения масштабируемости и отказоустойчивости. Это обеспечит стратегическое преимущество в конкурентной среде и дальнейшее улучшение качества предоставляемых услуг.

Заключение

На основе проведённого исследования в рамках диссертационной работы на тему: «Моделирование и разработка сервис-ориентированных приложений» были получены следующие основные научные результаты и практические выводы:

Разработана методология проектирования и реализации сервис-ориентированных архитектур (SOA) для систем управления заказами, которая обеспечивает повышение гибкости, масштабируемости и надёжности систем.

Проанализированы и применены современные подходы к декомпозиции систем на сервисы с использованием принципов доменно-ориентированного дизайна (DDD). Установлено, что автономность и слабая связанность сервисов являются ключевыми факторами для достижения оптимальной производительности и адаптации.

Создана модель бизнес-процессов на основе BPMN, что позволило выявить узкие места и оптимизировать процессы оформления заказов. На основе модели выделены ключевые сервисы, включая управление каталогом товаров, корзиной, заказами, платежами и доставкой.

Проведена детальная разработка структурного проектирования системы, включая UML-диаграммы классов, последовательностей и потоков данных, что обеспечило ясность и согласованность при реализации системы.

Продемонстрированы преимущества применения современных технологий, таких как Spring Boot, PostgreSQL, Docker и Kubernetes, для реализации SOA. Это способствовало значительному улучшению производительности системы. Внедрение предложенной методологии в предприятиях электронной коммерции позволит повысить операционную эффективность за счёт сокращения времени обработки заказов, увеличения пропускной способности и улучшения качества клиентского обслуживания.

Использование предложенного подхода может быть расширено на другие домены, требующие высокой адаптивности и надёжности, такие как логистика, банковская сфера и управление цепочками поставок.

Для дальнейшего повышения конкурентоспособности системы рекомендуется интеграция технологий искусственного интеллекта и машинного обучения для предиктивной аналитики и автоматизации процессов.

Применение облачных технологий позволит обеспечить ещё большую масштабируемость и отказоустойчивость, а также снизить затраты на инфраструктуру.

Получение исходных данных осуществлялось посредством анализа существующих систем управления заказами, изучения литературы и актуальных исследований в области SOA.

Проведённые эксперименты показали, что внедрение SOA позволило улучшить показатели производительности на 20-150%, а интеграция с внешними системами стала быстрее на 85%.

Таким образом, результаты работы подтверждают гипотезу о том, что применение сервис-ориентированной архитектуры способствует значительному улучшению показателей производительности и качества работы систем управления заказами. Разработанная методология имеет высокий потенциал для применения в реальных условиях, обеспечивая стратегическое преимущество в конкурентной среде.

Список используемой литературы

1. Аунг Аунг Хейн. Моделирование и разработка сервис-ориентированных приложений: специальность 05.13.11 "Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»: диссертация на соискание ученой степени кандидата технических наук / Аунг Аунг Хейн. – Москва, 2013. – 152 с. – EDN SUPOEN.

2. Долженко, А. И. Задача выбора эффективной сервис-ориентированной архитектуры экономической информационной системы / А. И. Долженко // Экономический вестник Ростовского государственного университета. – 2008. – Т. 6, № 3-3. – С. 124-127. – EDN PTVXUB.

3. Караханова, А. А. Анализ микросервисной архитектуры, монолитных приложений, архитектуры SOA / А. А. Караханова // Синергия Наук. – 2020. – № 46. – С. 255-262. – EDN BVOVEU.

4. Ключников, Е. А. SOA система хранения и поиска информации (платформа Magnet) / Е. А. Ключников // Вестник Сыктывкарского университета. Серия 1: Математика. Механика. Информатика. – 2008. – № 8. – С. 91-108. – EDN RUIFJP.

5. Козин, Е. Г. Реинжиниринг ИТ-архитектуры предприятия на базе сервис-ориентированного анализа архитектуры предприятия / Е. Г. Козин, И. В. Ильин, А. И. Левина // Перспективы науки. – 2016. – № 9(84). – С. 48-56. – EDN WYJVPX.

6. Коробков, К. Н. Применение сервис-ориентированной архитектуры в корпоративных информационных системах / К. Н. Коробков // Техника и технология. – 2008. – № 1. – С. 54-57. – EDN JUDIUJ.

7. Кравцов, К. И. Сервис-ориентированные архитектуры как средство автоматизации для управления информацией в электронной коммерции / К. И. Кравцов, Д. В. Тихоненко, А. В. Низамеева // Перспективы науки. – 2023. – № 8(167). – С. 69-72. – EDN XEZLAI.

8. Лучшая архитектура для MVP: монолит, SOA, микросервисы или бессерверная?.. Часть 1 [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/otus/blog/476024/ (Дата обращения: 08.11.2024).

9. Лучшая архитектура для MVP: монолит, SOA, микросервисы или бессерверная?.. Часть 2 [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/otus/blog/477930/ (Дата обращения: 08.11.2024).

10. Микросервисная и сервис-ориентированная архитектуры в задачах оптимизации управления информационной инфраструктурой ИТ-подразделения / С. А. Петрухин, П. Е. Черников, Г. Е. Глухов, А. Г. Карапетян // Научный вестник ГосНИИ ГА. – 2023. – № 43. – С. 187-196. – EDN WGSHOE.

11. Михеева, Е. Д. Микросервисная архитектура как разновидность сервис-ориентированной архитектуры / Е. Д. Михеева, М. С. Гребенюк // Наука молодых - будущее России : сборник научных статей 7-й Международной научной конференции перспективных разработок молодых ученых, Курск, 12–13 декабря 2022 года. Том 4. – Курск: Юго-Западный государственный университет, 2022. – С. 433-436. – EDN MIBUKM.

12. Михеева, Е. Д. Сравнение сервис-ориентированной архитектуры и микросервисной архитектуры / Е. Д. Михеева, М. С. Гребенюк // Наука молодых - будущее России : сборник научных статей 7-й Международной научной конференции перспективных разработок молодых ученых, Курск, 12–13 декабря 2022 года. Том 4. – Курск: Юго-Западный государственный университет, 2022. – С. 437-439. – EDN TJOKNE.

13. Пимкин, Д. Ю. Технология интеграции информационных систем на основе сервис-ориентированной архитектуры / Д. Ю. Пимкин // Прикладные исследования и технологии ART2015 : сборник трудов Второй международной конференции, Москва, 27–29 мая 2015 года. – Москва: Негосударственное образовательное учреждение высшего образования Московский технологический институт, 2015. – С. 207-211. – EDN UBFFPX.

14. Романов, А. SOA в России: особенности первых проектов / А. Романов // Банковские технологии. – 2008. – № 9. – С. 24-27. – EDN JUOQMD.

15. Самарин, А. В. О подходе к архитектуре, построению и развитию информационных бизнес-систем на основе BPM и SOA / А. В. Самарин // Бизнес-информатика. – 2009. – № 3(9). – С. 11-15. – EDN KVWYED.

16. Сергеева, И. И. Сервисно-ориентированная архитектура (SOA): опыт внедрения / И. И. Сергеева, А. А. Белильщикова // Научные Записки ОрелГИЭТ. – 2012. – № 1(5). – С. 440-444. – EDN QJDYTD.

17. Сервис-ориентированная архитектура (SOA). – Электронный ресурс – Режим доступа: https://habr.com/ru/company/mailru/blog/342526/ (дата обращения 02.11.2024).

18. Сервис-ориентированная архитектура. – Электронный ресурс – Режим доступа: http://citforum.ru/internet/webservice/soa/ (дата обращения 07.11.2024).

19. Сервис-ориентированная архитектура. – Электронный ресурс – Режим доступа: http://iso.ru/ru/press-center/journal/2046.phtml (дата обращения 02.11.2024).

20. Сервисно-ориентированная архитектура ИС ВШМ СПбГУ и проблемы стохастической оптимизации системы – Электронный ресурс – Режим доступа: https://www.math.spbu.ru/user/gran/sb3/gran_sheron.pdf (дата обращения 01.11.2024).

21. Сидорова, Н. П. Принципы построения корпоративных информационных систем на основе сервис-ориентированной архитектуры / Н. П. Сидорова // Информационно-технологический вестник. – 2016. – № 3(9). – С. 74-82. – EDN XAMNMT.

22. СОА (сервис-ориентированная архитектура) [Электронный ресурс]. – Режим доступа: https://www.ibm.com/ru-ru/cloud/learn/soa (Дата обращения: 10.11.2024).

23. Черноусов, А. В. Преимущества использования SOA при построении мультиагентных систем / А. В. Черноусов, Е. С. Черноусова // Информационные и математические технологии в науке и управлении : Труды XII Байкальской Всероссийской конференции, Иркутск-Байкал, 02–11 июля

2007 года / Ответственный редактор Л.В. Массель. Том Часть II. – Иркутск-Байкал: ИСЭМ СО РАН, 2007. – С. 59-65. – EDN UXDMLZ.

24. Щекочихин, О. В. Сервис-ориентированная архитектура как обеспечение расширения функций управления корпоративной информационной системы / О. В. Щекочихин // Тенденции развития науки и образования. – 2017. – № 29-3. – С. 15-17. – DOI 10.18411/lj-31-08-2017-39. – EDN ZIPEUH.

25. Что такое SOA? системы – Электронный ресурс – Режим доступа: http://www.finecosoft.ru/SOA (дата обращения 02.11.2024).

26. SOA бессмертна // Открытые системы. СУБД. – 2011. – № 1. – С. 4-13i. – EDN SEQDIZ.

27. SOA по-прежнему популярна // Открытые системы. СУБД. – 2011. – № 3. – С. 4-11a. – EDN SEQEON.

28. Bloomberg J. Process-Driven SOA: Leveraging Service-oriented Architecture for Business Process Innovation// ZapThink White Paper. 2006. June.

29. Castillo P. A. et al. SOAP vs REST: Comparing a master-slave GA implementation //arXiv preprint arXiv:1105.4978. – 2011.

30. Design and Implementation of a Service-oriented Information System Architecture Based on a Case Study – Электронный ресурс – Режим доступа: https://www.grin.com/document/71427 (дата обращения 05.03.2024).

31. Hill J. Achieving Agility: SOA Will Build Organizational Agility, but Watch the Hype//Gartner Research. 2006. April.

32. Kumar P. et al. Comparing Performance of Web Service Interaction Styles: SOAP vs. REST //Journal of Information Systems Applied Research. – 2013. – Т. 6. – №. 1. – С. 4.

33. Service Oriented Architecture (SOA) – Электронный ресурс – Режим доступа: http://www.institutio.ru/soa (дата обращения 01.04.2024).

34. Ultimus. Ultimus and the Business Process Ecosystem: Complementing SAP and Enterprise Applications//Ultimus White Paper. 2006. June.

35. Understanding Service-Oriented Architecture – Электронный ресурс – Режим доступа: https://learn.microsoft.com/en-us/previous-

versions/aa480021(v=msdn.10)?redirectedfrom=MSDN#soa-basics (дата обращения 01.11.2024).