МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

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

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Бизнес-информатика
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка информационной системы учета заказов для производственной фирмы»

Обучающийся О.В. Евзрезов
(Инициалы Фамилия) (личная подпись)

Руководитель В.Ф. Глазова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2022

Аннотация

Объектом исследования выпускной квалификационной работы является бизнес-процесс учета заказов в производственной фирме.

Предметом исследования является процесс автоматизации учета заказов в производственной фирме.

Основной целью работы является разработка информационной системы учета заказов для производственной фирмы.

Актуальность работы обусловлена необходимостью повышения эффективности работы фирмы, что позволяет более эффективно конкурировать с другими компаниями из того же сегмента рынка. Внедрение в компанию автоматизированной системы учета заказов позволяет формировать разносторонние сведения о текущей деятельности компании и перспективах её развития.

В первой главе даются характеристики предприятию, для которого будет разработано решение. Проводится анализ бизнес-процессов, соотносящихся с темой работы, приводится список требований, предъявляемых к информационной системе со стороны предприятия. Описана модель бизнес-процессов «Как должно быть». А также даётся обзор существующих решений на рынке.

Вторая глава посвящена проектированию системы учета заказов. Приводится описание связей между различными сущностями и сами эти сущности как таблицы в БД или как классы программы.

В третьей главе описана разработка системы учета заказов. Представлено тестирование разработанной системы, и проведен расчет экономической эффективности разработки.

Выпускная квалификационная работа на тему «Разработка информационной системы учета заказов для производственной фирмы» состоит из введения, 3 глав, заключения, списка используемой литературы и источников, который состоит из 21 источника, а также – двух приложений.

Оглавление

Введение

Современные информационные системы являются не просто средством автоматизации и повышения эффективности, но важным элементом архитектуры компании. На основе этих решение компания может собирать дополнительную информация о своей деятельности, оценивать работу сотрудников и отдельных подразделений. Организации все чаще вкладывают значительные средства в системы, способные помочь компании выжить в стремительно изменяющееся внешней среде и условиях конкуренции.

Целью работы является анализ и сбор технических и экономических исходный данных, связанных с исследуемыми процессами фирмы, анализ бизнес-процессов предприятия, создание логической концептуальной модели информационной системы и ее дальнейшее реализация в рамках сформированных требований.

Задачами выпускной квалификационной работы являются:

- исследовать организацию учета заказов продукции в компании;
- проанализировать существующие программные продукты для расчета заказов;
- разработать информационное обеспечение для хранения и обработки информации по заказам;
- создать программное обеспечение для расчета заказа;
- рассчитать экономический эффект от внедрения информационной системы.

Разработка информационной системы – от начальной фазы до развертывания – состоит из трех последовательных и поступательных этапов: анализа, проектирования и реализации. Задача анализа включает в себя определение основных бизнес-процессов организации и ее задач. Бизнес-процессы описываются при помощи определенных стандартов таких как IDEF3, DFD, и DEF0. На этапе проектирования строится скелет системы, используются диаграммы вариантов использования. Это дает возможность

описать систему на концептуальном уровне. Затем проектируется база данных определяются необходимые таблицы их поля, если нужно логика программы разбиваются на отдельные модули или независимые сервисы, которые представляют из себя полностью независимый от основного приложения функционал. Сервисы имеют определенные точки входа и протокол взаимодействия. Последним этапом являться реализация. На этом этапе создаётся программное обеспечение.

Объектом исследования является бизнес-процесс учета заказов в производственной фирме.

Предметом исследования является процесс автоматизации учета заказов в производственной фирме.

При выполнении работы применялись следующие методы исследования:

- объектно-ориентированное программирование;
- метод проектирования диаграммы сущность-связь;
- метод структурно-функционального анализа;
- метод классификации и кодирования;
- методы расчета экономической эффективности.

Глава 1 Функциональное моделирование предметной области

1.1 Характеристика предприятия и описание его деятельности

В данной работе рассматривается фирма, которая занимается внедрением решений связанными с информационными технологиями, а также разрабатывает автоматизированные системы управления. Компания хорошо зарекомендовала себя в сфере защиты информации, телекоммуникации.

Также фирма в ближайшее время планирует расширить линейку продукции добавив сборку и производство инфокиосков. Для этого будет создан новый дивизион, который будет ответственен за следующие процессы: приема заказов от клиента, закупки, складирование, производство. Решение отдать все процессы, связанные с производством, в отдельный дивизион, было принято в связи с тем, что это позволит использовать лучшие современные практики автоматизации процессов и сосредоточиться на разработке конкретного продукта.

Основными направлениями деятельности предприятия на текущий момент являются:

- создание полностью готовых к эксплуатации систем информатизации и телекоммуникаций;
- разработка решений для защиты информации, включая средства криптографической защиты информации;
- проектирование, разработка и поставка, строительство и монтаж локально-вычислительных сетей и связи на основе волоконно-оптических линий, в том числе с мониторингом оптической сети и защитой от несанкционированного доступа;
- аттестация объектов информатизации по требованиям безопасности информации, включая спец исследования и спец проверки.

Во главе предприятия стоит генеральный директор. В его функции

входит контроль деятельности предприятия, анализ деятельности подразделений и принятие решений, которые будут способствовать увеличению эффективности или устранению проблем, а также участие в обсуждение и принятия решений по вопросам, определяющим работу предприятия, подписание договоров.

Определенная часть полномочий директора делегируется его заместителям или руководителям подразделений.

На предприятии есть общие административные подразделения такие как отдел кадров, финансово экономический отдел, отдел маркетинга, юридический отдел, отдел перспективного планирования.

На предприятии действуют отдел технического контроля (ОТК), подразделения материально-технического обеспечения, сопровождения и сборочно-монтажные участки, налажено серийное производство. Имеется архив конструкторской документации и программных изделий, соответствующий нормативным требованиям.

Производственная база предприятия позволяет выполнять все виды работ по разработке и производству сложной высококачественной продукции.

Предприятие имеет библиотеку научно-технической документации, насчитывающую более 1500 наименований действующих российских и международных стандартов и другой нормативной документации.

1.2 Характеристика комплекса задач

1.2.1 Инструменты для построения диаграмм и схем

В настоящий момент есть множество решений для построения диаграмм, схем потока данных, связей таблиц базы данных и т. д.

Одним из решений, котором активно пользовались в прошлом, является BPwin продукт компании ltd. Logic Works. Этот инструмент используют для поддержки процесса создания информационных систем. Он

принадлежит к группе CASE средств верхнего уровня. Впервые BPwin была выпущен в 1995 г. вместе с другим CASE решением – ERwin, инструмент для моделирования данных.

ERwin process modeler предоставляет возможность реализовывать функциональное моделирование. Данный модуль поддерживает работу со следующими стандартами DFD, IDEF3 и DEF0. ERwin process modeler может быть использован для создания диаграммы потока работ, диаграммы функций и диаграммы потока данных.

Правда нужно отметить, что на 2022 год последний релиз ERwin был в январе 2017, то есть 5 лет назад. Также эта программа поддерживает только операционную систему Windows. В то же время есть множество «облачных» платформ, которые можно использовать для построение дигамм и схем. В основном, они предоставляют общие инструменты для решения данных задач.

Одним из примеров подобных платформ являются Draw.io и Lucid.app так как это «облачные» платформы, то они не зависят от операционной системе, подключиться к этим сервисам можно с любого устройства, у которого есть доступ в интернет и установлен браузер. Экспортировать данные из платформ можно в большинстве современных форматов (svg, png, jpeg, pdf, xml, а также в собственных форматах платформы).

1.2.2 Выбор комплекса задач автоматизации

Основной целью моделирования является документирование и последующие осуществление функционального анализа на предмет поиска «узких» мест процессов и возможностей для их совершенствования. Важно, что эта деятельность ни в коем случае не является «моделированием ради моделирования». В дальнейшем полученные результаты будут использоваться при формировании функциональных требований к системам.

Для выполнения структурно-функционального анализа процесса работы подразделения было выполнено проектирование на основе методологии IDEF0. На рисунке 1 приведем контекстную диаграмму

деятельности подразделения предприятия [6], [18].

Стратегии и процедуры, которыми руководствуется процесс (управление) – это организация документооборота предприятия.

Входом для системы являются заказ клиента, а также товары от поставщика и оплата по выставленному клиенту счету.

Выходом для системы являются счет клиенту, отгруженный товар и накладная.

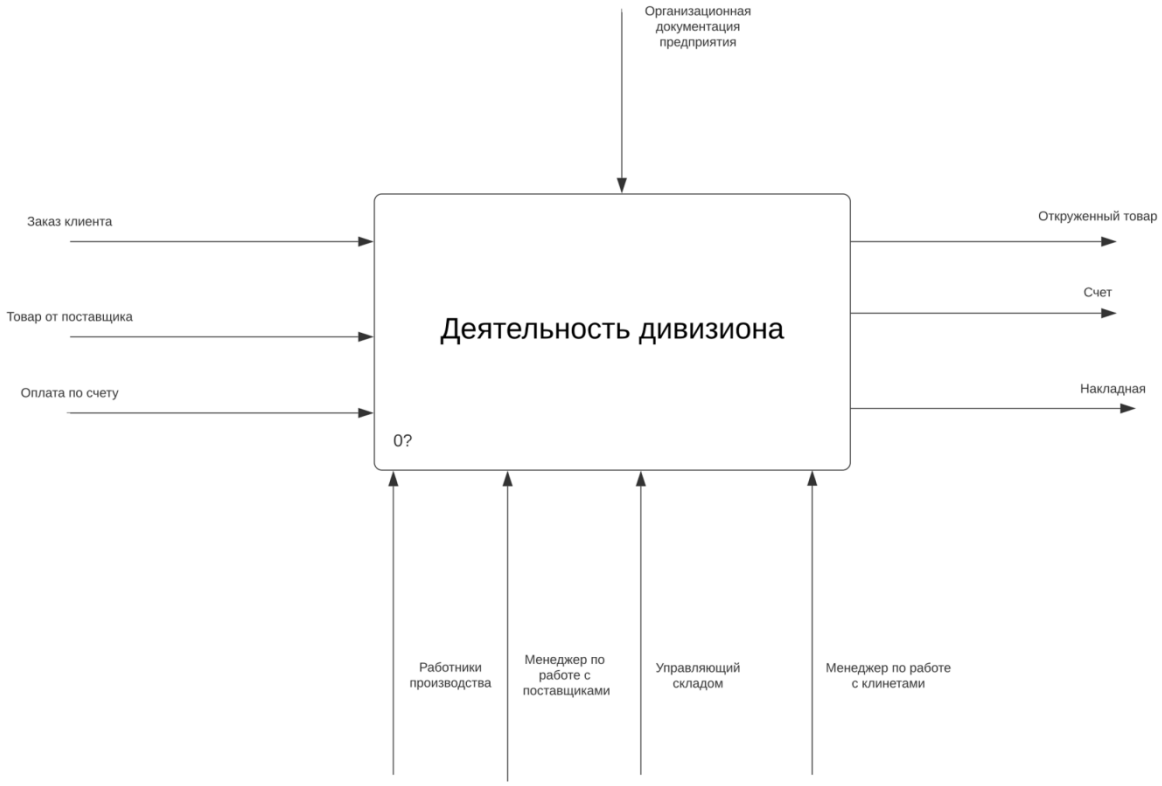

Рисунок 1 – Диаграмма IDEF0 деятельности подразделения

Функциональная декомпозиция работы подразделения, приведенная на рисунке 2, также выполнена на основе методологии IDEF0.

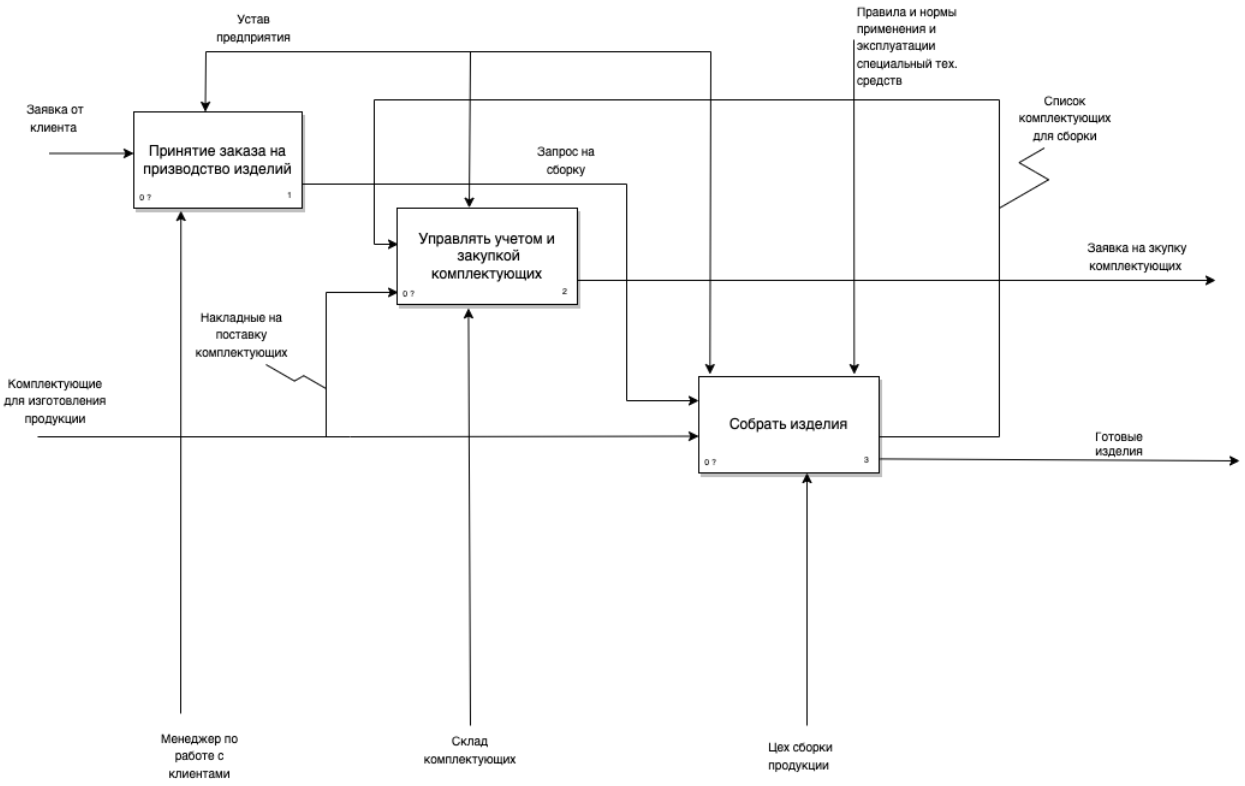

Рисунок 2 – Диаграмма декомпозиции деятельности подразделения «Как есть»

После получения заявки на производство изделий от заказчика идет процесс анализа заказа.

На этом этапе идет анализ того сколько трудовых ресурсов понадобится для осуществления заказа, какой сейчас уровень загрузки производства и может ли цех сборки взять дополнительный заказ, если нет, то через какой промежуток времени это будет возможно. Несмотря на то, что срок изготовления единицы продукции заранее известен, в процессе выполнения работы могут возникнуть форс-мажоры, которые могу повлечь за собой серьезный пересмотр сроков выполнения заказа. Если нет никаких серьезных проблем и цех сборке может принять новый заказ или поставить его в очередь, создается запрос на создание продукции, который поступает в цех сборки. Цех сборки создает список на закупку комплектующих, после получения которых преступает к разработки продукции.

На данный момент компания не имеет автоматизации данных бизнес-процессов. Отдел сборки использует Microsoft Excel для формирования списка комплектующих. Этот список получает склад, который отправляет заявку на комплектующие, после получения комплектующих идет сборка деталей. Сами заказы согласуются с цехом сборки на специальных встречах, на которых присутствует менеджер отдела продаж и представитель цеха сборки. На этих встречах обсуждаются требования к заказу и сроки его выполнения.

Данные подход не являться оптимальным, так как не поддаётся масштабированию, и он не сохраняет историю, к которой можно легко обратиться в случае возникновения спорных вопрос, также он не дает четкого понимания, на какой конкретно стадии находится выполнение заказа. Удобство в создании необходимых документов и отчетности тоже вызывает вопросы.

1.2.3 Обоснование необходимости использования вычислительной техники для решения задачи

Сегодня наблюдается становление и окончательное формирование цифровой трансформации существующих организаций. На данном этапе информационные технологии становятся инноватором, обеспечивающим конкурентное развитие предприятия и получение добавленной стоимости. Предприятия, которые изначально не были цифровыми и применяли ИТ в минимальном объеме, производят цифровизацию собственной деятельности для максимально эффективного использования информационных технологий.

Вместе с этим формируются цифровые платформы, позволяющие сформировать экосистему взаимодействия многочисленных участников цепочки создания ценности. В рамках единой цифровой платформы взаимодействуют поставщики, разработчики, производители, консультанты, аналитики, конечные потребители и иные участники рынка. Компания, владеющая цифровой платформой, становится модератором экосистемы –

своеобразным центром виртуального предприятия, охватывающего полную цепочку создания ценности, начиная с поставок и заканчивая производством, маркетингом и обслуживанием.

Внедрение информационной системы для автоматизации задач на сегодняшний день нужно не только для того, чтобы увеличить эффективность процессов предприятия, но и для сбора цифровых данных. На основе этих данных в будущем можно построить различные предсказательные модели о поведении клиентов, динамики спроса, цен на продукцию и т д.

1.2.4 Функциональные требования к системе

Требование – это набор определенных параметров, которым должна соответствовать информационная система. Есть множество источников, из которых можно достать эти данные (ведь в итоге необходимо создать систему для большого числа пользователей с различными задачами), в связи с чем исходная информация является достаточно противоречивой, неструктурированной и изменяющейся во времени. Но без этого этапа начинать работу не имеет смысла, так как именно согласие заказчика и исполнителя по вопросам объема и содержания работ, временных и финансовых ограничений является критическим для выполнения проекта автоматизации.

FURPS+**.** Для рассмотрения всех аспектов проектируемой ИС используется специальная классификация требований. Сокращение FURPS является аббревиатурой и в переводе на русский включает пять следующих аспектов: Функциональность, Удобство использования, Надежность, Производительность и Поддержку.

Ниже описаны требования к информационной системе учета заказов. В функциональных требованиях описаны лишь основные функции.

Functionality (функциональность):

– должна быть возможность создания заказа в системе;

– над заказом можно совершить любую из CRUD (Create, read, update,

delete) операций, также у заказа можно поменять статус готовности;

- должна быть возможность создания конкретного комплектующего в системе;
- должна быть возможность создавать и связывать с заказом необходимые для него списки комплектующих.

Usability (удобство использования — характеристики пользовательского интерфейса):

- интуитивно понятный интерфейс;
- защита от ошибок.

Reliability (надежность):

- отказоустойчивой;
- восстанавливаемой (резервное копирование с определенной частотой).

Performance (производительность):

- поддержка 10 и более одновременно работающих сотрудников;
- время отклика системы максимум 2 секунды.

Supportability (поддерживаемость):

- легко установить.

1.3 Анализ существующих решений и выбор стратегии автоматизации

1.3.1 Анализ существующих решений для автоматизации задачи

На вопрос о том, какая программа или сервис является лучшим, ответить достаточно трудно. Все зависит от того, что нужно конкретному предприятию, учитывая его специфику. Большинство подобных сервисов существует в интернете и выступают они в качестве CRM систем. Подобрать нужные функции очень сложно и в силу этого проще приобрести программу, сделанную на заказ.

Мною были рассмотрены некоторые программные продукты, которые

подходят для автоматизации системы учета заказов:

– онлайн сервис CASO;

– система учета заказов и клиентов SBIClients;

– CRM программа Supasoft;

– Мегаплан CRM;

– «1С: Торговля и Склад - проф.».

Сравнительная характеристика программ учета заказов представлена в таблице 1.

Таблица 1 – Сравнительная характеристика программ учета заказов

	Название программы	**Приблизительная стоимость**	**Открытость архитектуры**	**Справочная система**	**Простота интерфейса**	**Создание архивных копий**
1.	Онлайн сервис CASO	От 290 р. в мес. + 300р. за каждые 5 пользователей	-	+	+	-
2.	Система учета заказов и клиентов SBIClients	Условно бесплатная	+	+	+	-
3.	CRM программа Supasoft	бесплатная	+	-	-	-
4.	Мегаплан CRM	От 7500р. в месяц	-	+	-	-
5.	«1С: Торговля и Склад - проф»	8400р.	+	+	-	+

Обзор показал, что основными недостатками проанализированных программ являются:

– оплата абонентской платы за веб-сервис;

– множество функций неподходящих предприятию;

– отсутствие возможности настроить интерфейс веб-сервиса под свои потребности;

– недостаточное количество функций бесплатных программ и

сервисов;

– отсутствие создания архивных копий.

Поэтому было принято решение сформулировать все необходимые задачи и функции для организации системы учета заказов и на основе этих данных создать программу, которая удовлетворяла бы всем требованиям предприятия.

1.3.2 Выбор и обоснование способа приобретения ИС для автоматизации задачи

Существует несколько способов приобретения информационной системы:

– покупка готового решения;

– покупка и последующая доработка информационной системы;

– аутсорсинг, аренда;

– разработка собственной информационной системы.

Первый вариант не учитывается, т.к. при сравнении готовых информационных систем учета заказов в пункте 1.3.1 было наглядно продемонстрировано, что из пяти систем, наиболее подходящих под требования, ни одна не удовлетворяет им на сто процентов.

Второй вариант кажется более привлекательным, нежели первый, однако покупка, формирование технического задания, доработка, внедрение, тестирование информационной системы плюс, возможно, приобретение дополнительного коммерческого программного обеспечения является не самым оптимальным решением и в плане временных, и в плате стоимостных затрат.

Аутсорсинг и аренда большого интереса не представляют, т.к. на первых порах будет необходима постоянная доработка программы, и, кроме того, её жизненный цикл будет составлять не один год.

Разработка собственной информационной системы – самый оптимальный вариант. Как уже было рассмотрено выше, разрабатывать информационную систему собственными силами гораздо удобнее. Можно

выделить следующие плюсы:

- возможность самостоятельно выбрать и язык программирования, и сервер баз данных, и библиотеку графического интерфейса, и библиотеку доступа к базе данных, опираясь как на стоимостные характеристики, так и на временные при использовании того или иного компонента, и учитывая наличие опыта при работе с ними, документации, примеров, и пр.;
- реализация только того минимального функционала, который сейчас нужен для автоматизации системы учета заказов;
- изначальное ведение проекта с хорошим сопровождением, чтобы в будущем, даже при уходе разработчика проект можно было бы развивать и дальше;
- разработка с одновременным тестированием и частичным внедрением, позволят таким образом уменьшить время и стоимость разработки;
- на этапе тестирования и внедрения можно учитывать пожелания пользователей системы, например, в части графического интерфейса, и изначально создавать систему максимально удобной конечному пользователю.

1.3.3 Разработка модели бизнес-процесса «Как должно быть»

Модель бизнес-процессов «Как должно быть» приведена на рисунке 3. За основу были взяты бизнес-процессы из раздела 1.2.2 и изменены с учетом внедренной информационной системы учета заказов, сокращенно ИС «УЗ» [12].

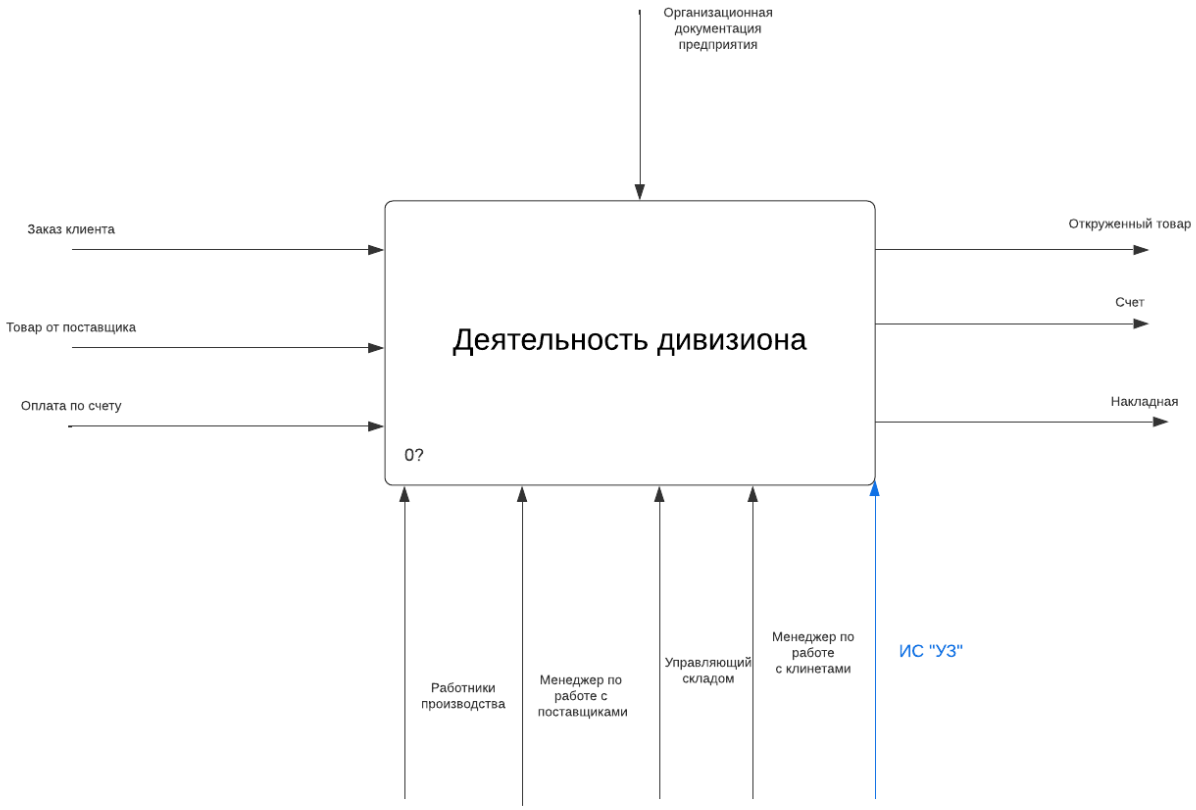

Рисунок 3 – Диаграмма деятельности подразделения «Как должно быть»

На рисунке 4 приведена декомпозиция деятельности подразделения «Как должно быть».

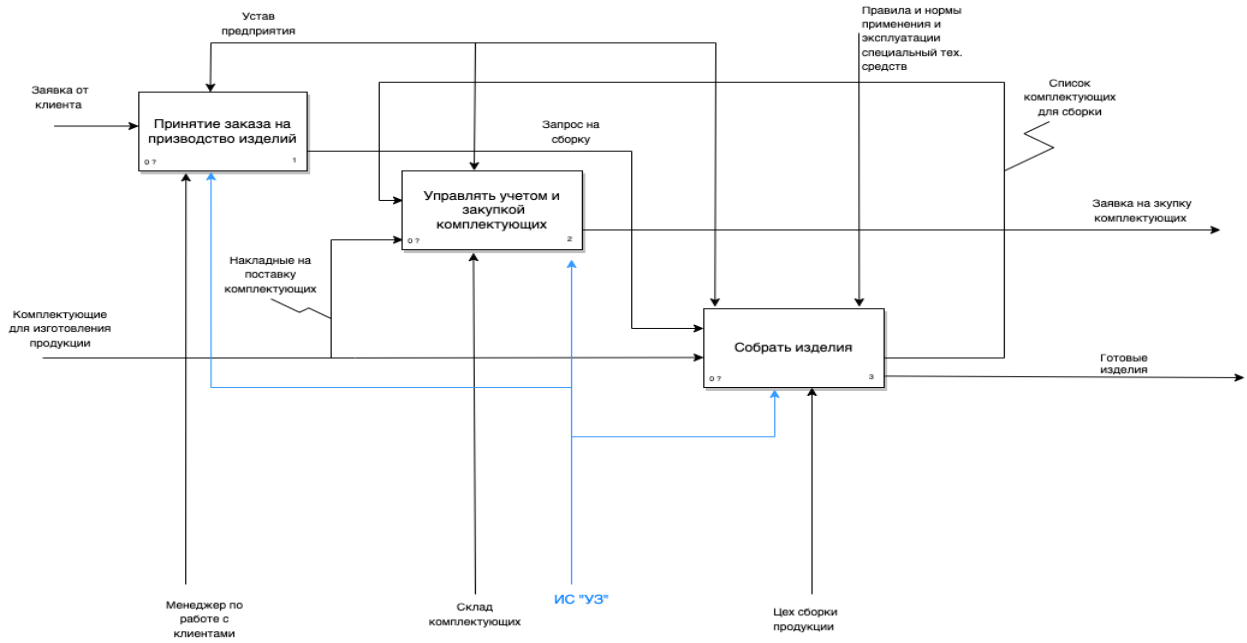

Рисунок 4 – Диаграмма декомпозиции деятельности подразделения «Как должно быть»

Автоматизированная система учета заказав позволяет контролировать процесс изготовления продукции подразделением, управлять заказами и закупкой комплектующих.

Выводы по главе 1

В результате проведенного анализа бизнес-процессов на предприятии была сформирован список требований для информационной системы. Приведен краткий обзор программных продуктов, так или иначе способствующих реализации задач, и приведены аргументы в пользу разработки собственного проектного решения.

Глава 2 Проектирование информационной системы учета заказов

2.1 Варианты использования приложения

Одной из основных задач создания информационной системы является разработка программного продукта, которые позволяет автоматизировать и упростить выполнения повседневных задач сотрудником. Для создание такого программного продукта необходимо собрать требования. В подразделе 1.2.4 уже были описаны функциональные требования к системе, но для того, чтобы более точно понимать как должна работать программа, следует описать функциональность системы через варианты использования, еще это называют Use Case (прецеденты) [1].

На рисунки 5 приведена модель функциональный требований. На диаграмме отображены основные роли в системе: менеджер по работе с клиентами, сотрудник склада, сотрудник производственного отдела (сборщик).

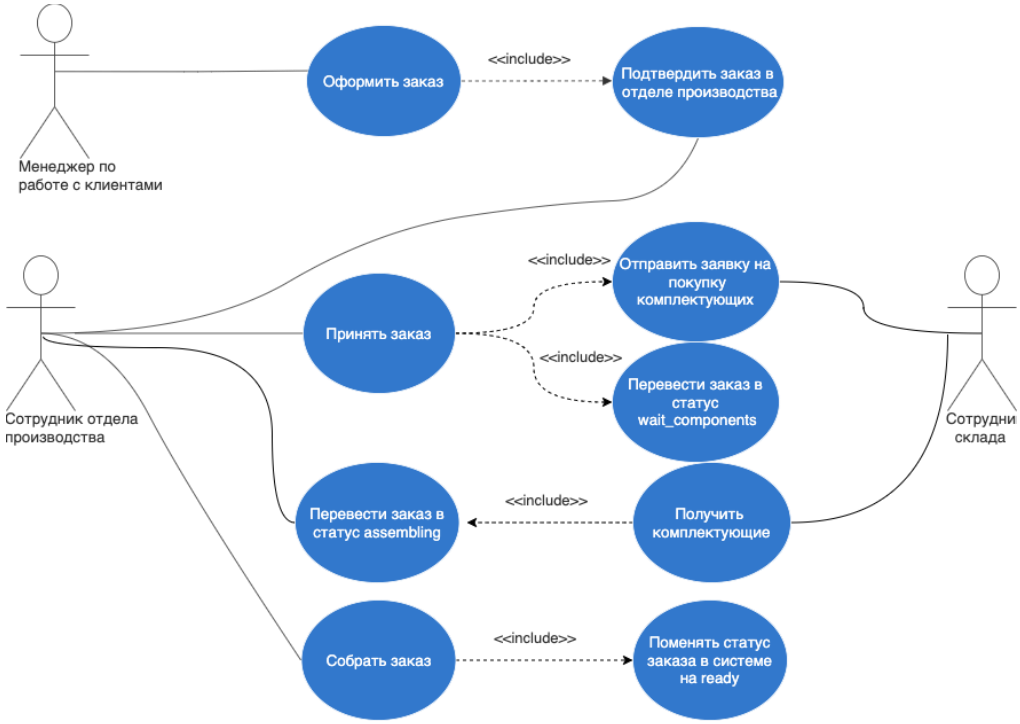

Рисунок 5 – Диаграмма вариантов использования

Заказ имеет следующие статусы:

– init – статус присваивается только что созданному заказу;
– wait_components – статус присваиваться заказу после того, как был отправлена заявка на компоненты;
– assembling – в этот статус переходит заказ, когда получены компоненты;
– ready – статус «Заказ готов».

Этап «Подтвердить заказ в производственном отделе» требует обсуждения заказа с менеджером из отдела продаж и представителем производственного отдела. Все заказы на продукции имеют определенный минимум, от которого заказ может быть принят в работу. Каждый из этих заказов обсуждается отдельно. Для сокращения времени ответа менеджер связывается с производством по телефону или через электронную почту, если заказ принимается, он переводится на сотрудника с ролью сборщик, который дальше ведет работу с заказом в программе. Если в связи с загруженностью на заказ нужно потратить дополнительное время, сдвиг сроков согласуется отдельно с заказчиком и сроки меняются в заказе.

2.2 Выбор архитектуры приложения

Перед тем как приступить к проектированию базы данных, нужно определиться с архитектурой приложения, будет ли это монолитное приложение или оно будет построено по принципу микросервисов.

Монолитная архитектура отличается тесной связью между различными частями программы, наличием единого репозитория для хранения кода. Программа, основанная на данной архитектуре, представляет собой единое целое, и внутри себя программа может проводить манипуляции с совершенно логически несвязанными сущностями. Например, сущность «Сотрудник» отличается от сущности «Заказ», но в рамках программы они никак не разделены. В случаи с монолитной архитектурой даже мелкая правка

требует развертывания всей программы на сервере. Микросервисная архитектура напротив представляет из себя множество независимых программ (сервисов), которые могу разрабатываться и тестироваться независимо друг от друга. Каждый из сервисов предоставляет определенный API (Application Programming Interface). На рисунки 6 схематична описана разница между монолитной и микросервисной архитектурой.

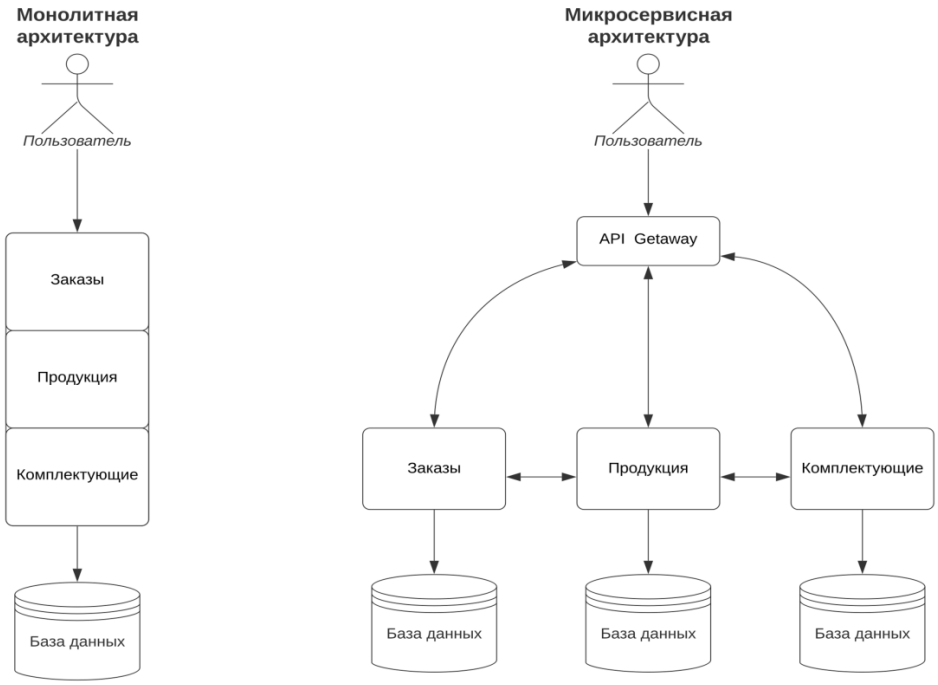

Рисунок 6 – Разница между микросервисной и монолитной архитектурой

Под пользователем на рисунки 6 подразумевается клиент-программа, отправляющая и получающая данные, например по протоколу HTTPS.

У микросервисной архитектуры есть множество плюсов, но при этом есть и значительные минусы. Например, она не позволяет проводить join-операции с таблицами в БД. Также множество отдельных сервисов тяжелее обслуживать чем один большой, так как каждый сервис нужно настраивать отдельно, и любая доработка, которая затрагивает несколько сервисов, требует согласованного релиза всех этих сервисов [13].

Микросервисная архитектура больше подходит сложным развивающимся проектам в котором участвует множество команд, каждая из которых поддерживает свой конкретный сервис. Поэтому для системы автоматизации заказов лучше использовать монолитную архитектуру.

2.3 Описание взаимодействия с системой

Сотрудник фирмы может взаимодействовать с системой через рабочий ПК. Требования к ПК сотрудника: Процессор – Intel Core i5 650 (Clarkdale) и выше, HDD (жесткий диск) – не менее 500 Мб свободного места, оперативной память – не менее 4 ГБ, разрешение экрана – 1280 x 720 (широкоформатный), операционная система: Windows 7 (32/64), macOS Catalina и более поздние версии, один из браузеров Safari, Google Chrome, Mozilla Firefox.

На рисунке 7 описано взаимодействие с сотрудника с сервером на котором развернуто приложение.

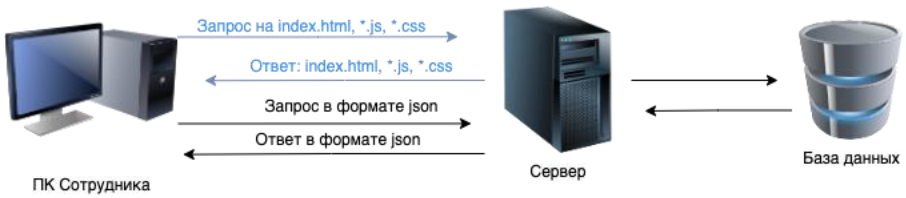

Рисунок 7 – Взаимодействие с системой

Для взаимодействия с приложением сотрудник фирмы должен открыть браузер и ввести в адресную строку хост сервера, на котором развернуто приложение. После этого в окне браузера должно появиться страница авторизации с просьбой ввести свой логин и пароль. Нужно отметить, что у фирмы нет требования на ограничение доступа к серверу только из локальной сети.

Для того чтобы реализовать маршрутизацию запросов на сервере можно использовать веб-сервер Nginx. И настроить конфигурации таким образом, чтобы по умолчанию возвращалась страница index.html в коде которой присутствуют ссылки на файлы CSS стилей и JS скриптов [2], [17].

Взаимодействие с сервером для CRUD (Create, Read, Update, Delete) операций с данными лучше построить на основе REST-архитектуры [4]. В этом случае взаимодействие с сервером осуществляется по протоколу HTTP или HTTPS и особое значение имеют методы протокола (тип запроса). Метод Get на получение данных, POST на их создание, PUT на изменение, DELETE на удаление. Данная архитектура позволяет стандартизировать запросы, пользователю API в этом случае сразу понятно, что все запросы с методом GET отвечают исключительно за получение данных.

2.4 Проектирование базы данных

2.4.1 Описание таблиц и их связей

На рисунке 8 отображена схема таблиц в базе данных.

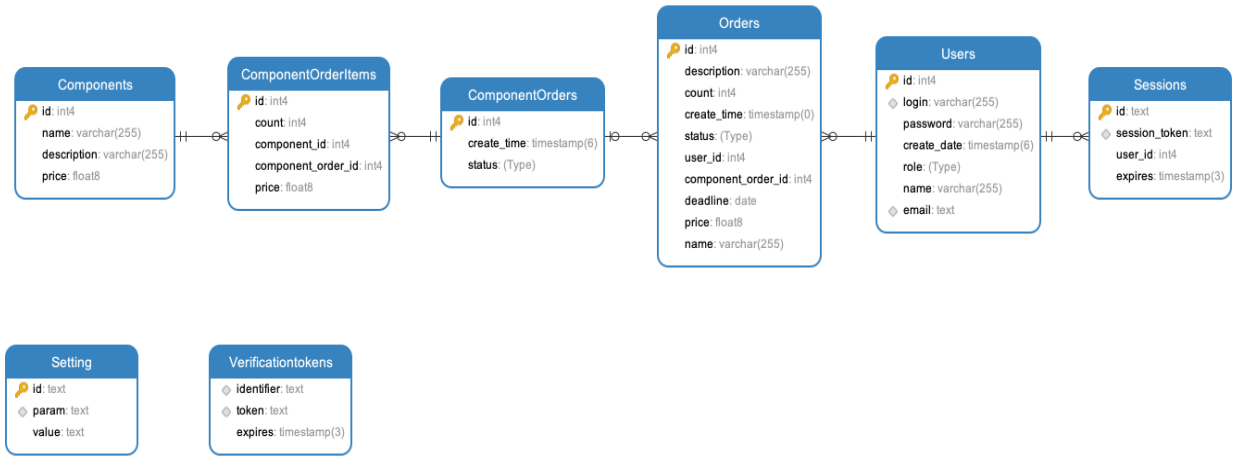

Рисунок 8 – Схема связей таблиц в базе данных

На основе перечисленных требований опишем таблицы и их связи [5], [9], [10], [16].

Таблица Compnents (Компоненты), будет содержать следующие поля:

- id – Идентификатор;
- name – Имя компонента;
- description – Описание компонента;
- price – Цена за единицу.

Таблица Users (Пользователи), будет содержать следующие поля:

- id – Идентификатор;
- login – Логин пользователя для входа в систему;
- password – Пароль для входа в систему;
- role – Роль пользователя, в системе будут следующие роли:
  a. admin – Администратор системы;
  b. manager – Менеджер по работе с клиентами;
  c. developer – Сотрудник производства;
  d. storekeeper – Сотрудник склада;
- create_date – Дата создания пользователя.

Таблица Orders (Заказы), будет содержать следующие поля:

- id – Идентификатор;
- description – Описание заказа;
- count – Количество единиц продукции;
- create_time – Дата создания заказа;
- price – Стоимость заказа;
- status – Статус, одно из значений:
  a. init – Статус присваивается только что созданному заказу;
  b. wait_components – Статус присваиваться заказу после того как был отправлена заявка на компоненты;
  c. assembling – В этот статус переходит заказ, когда получены компоненты и началась сборка;
  d. ready – Статус «Заказ готов»;
- user_id – Внешний ключ, ссылающийся на таблицу Пользователей

(Users);

– deadline – Заказ должен быть выполнен до этой даты;
– component_order_id – внешний ключ, ссылающийся на таблицу Заказов компонентов (ComponentOrders).

Таблица CompnentOrder (Заказы на компоненты), будет содержать следующие поля:

– id – Идентификатор;
– create_time – Дата создания заказа;
– status – Статус, одно из значений:
  a. init – Статус присваивается только что созданному заказу;
  b. wait_components – Статус присваивается заказу после того как был отправлена заявка на компоненты;
  c. ready – Статус «Заказ готов».

Таблица CompnentOrderItems (Заказы на конкретные компоненты), будет содержать следующие поля:

– id – Идентификатор;
– count – Количество единиц продукции;
– price – Стоимость партии компонентов;
– component_order_id – внешний ключ, ссылающийся на таблицу Заказов компонентов (ComponentOrders);
– component_id – внешний ключ, ссылающийся на таблицу Заказов компонентов (Components).

2.4.2 Описание жизненного цикла заказа

В таблицу Orders пользователь с ролью manager (Менеджер по работе с клиентами) может добавить заказ. Заказу автоматически присваивается статус init. После того как заказ был принят сотрудником с ролью developer (Сотрудник цеха) заказ переводится на этого сотрудника. Дальше developer формирует заявку на компоненты, выбирая нужные данные из таблицы Components. Каждый из компонентов имеет отдельную запись в таблице

CompnentOrderItems, и все новые записи завязываются с конкретной записью в таблице CompnentOrder, она в свою очередь связывается с записью из таблице Orders. Дальше этот заказ переводится на сотрудника с ролью storekeeper (Сотрудник склада), и статус заказа меняется на wait_components. После получения необходимых компонентов storekeeper переводит заказ обратно на developer-a. Developer получает необходимые компоненты и приступает к сборке продукции и переводит заказ в статус assembling. По завершению сборки продукции статус заказа меняется на ready.

2.4.3 Выбор технологий разработки информационной системы

Технологии, на которых будет идти разработка информационной системы и дальнейшее ее обслуживание и развитие, можно разделить на несколько групп. Технологии, которые касаются Фронтенда, Бэкенда и те, которые позволяют ускорить процесс разработки и тестирования. Фронтенд (англ. front-end) - клиентская сторона пользовательского интерфейса к программно-аппаратной части сервиса. Бэкенд (англ. back-end) - программно-аппаратная часть сервиса, отвечающая за функционирование его внутренней части.

Фронтенд технологии, которые будут использоваться для разработки пользовательского интерфейса, не зависимы от бэкенда. Это значит, что бэкенд не отвечает за работу пользовательского интерфейса, он лишь предоставляет API на получения и обработку данных. При этом пользователь имеет доступ к статическим файлам на сервере, которые отвечают за работу интерфейса и доступ к этим файлам может получить любой пользователь, если у него есть ссылка на эти файлы.

Для разработки фронтенда информационной системы учета заказов производственной фирмы используются следующие технологии:

- язык разметки HTML;
- язык описания внешнего вида документа CSS;
- язык программирования TypeScript. Нужен для типизации данных;
- JavaScript библиотека с открытым исходным кодом React. Помогает

упросить разработку пользовательского интерфейса [19];

- библиотека UI компонентов Ant.design. Эта библиотека содержит стилизованные компоненты пользовательского интерфейса такие как кнопка, поле ввода данных, поле выбора данных, всплывающие окна и т. д.;
- Webpack сборщик модулей JavaScript с открытым исходным кодом. Был разработан для JavaScript, но может преобразовывать и другие ресурсы, такие как HTML, CSS и изображения.
- React Query библиотека для синхронизации, кэширования, получения, обновления данных с сервером. Совместима с React приложениями;
- Jest - среда для тестирования JavaScript. Jest хорошо документирован, имеет большое комьюнити и требует минимальных настроек перед началом работы. В случае падения тестов предоставляет детальное описание причин;
- Cypress - JavaScript-фреймворк для тестирования. При использовании этого инструмента тестировщик может видеть процесс прохождения теста в реальном времени на экране приложения cypress. Приложение состоит из двух фреймов, в одном фрейме открывается тестируемый продукт в другом запускаются заранее написанные тесты.

На сегодняшний день эти технологии являются стандартным решением для разработки пользовательских интерфейсов.

Для разработки бэкенда информационной системы учета заказов производственной фирмы используются следующие технологии:

- объектно-реляционная система управления базами данных PostgreSQL;
- Next.JS открытый JavaScript фреймворк, позволяет значительно ускорить разработку [21];

– Prisma ORM. Открытая ORM для Node.js и TypeScript. Позволяет создавать удобные в использовании модели данный [20];
– NodeJS программная платформа, основанная на движке V8 выполняющая роль веб-сервера [11];

Использование JavaScript-а (NodeJS) на стороне сервера способно значительно удешевить и ускорить разработку, так как этот язык является весьма популярным для написания пользовательских интерфейсов, и разработчик интерфейсов в этом случае сможет также вести разработку на стороне сервера.

Технологии и сервисы, которые ускоряют процесс разработки:

– Git – распределённая система управления версиями. Нужна для контроля версий проекта и изменений в проекте;
– Visual Studio Code – редактор исходного кода. Бесплатный редактор кода поддерживает множество языков программирования и множеством плагинов, которые расширяют его возможности по анализу кода и отладке программы;
– Trello – облачная программа для управления проектами небольших групп. Нужна для постановки задач членам команды;
– GitLab – веб-инструмент жизненного цикла DevOps с открытым исходным кодом.

Описанные выше технологии использовались в процессе разработки информационной системы учета заказов для производственной фирмы.

Выводы по главе 2

Проведено исследование функциональной части системы. Описано взаимодействие пользователя с системой и жизненный цикл заказа. Выбрана подходящая архитектура системы, а также создана схема базы данных с описанием необходимых таблиц и их полей.

Глава 3 Разработка системы учета заказов для производственной фирмы

3.1 Разработка программного продукта

3.1.1 Структура проекта

На рисунке 9 показана структура файлов и папок в проекте. В папке pages хранится код, отвечающий за пользовательский интерфейс, в папке pages/api хранится код, отвечающий за API приложения.

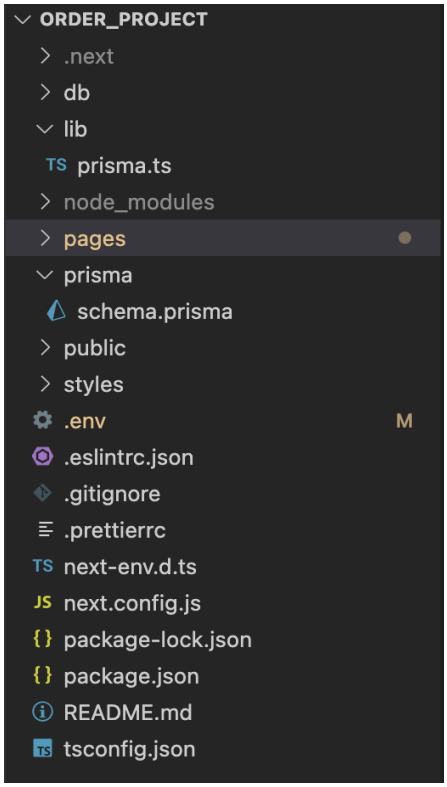

Рисунок 9 – Структура файлов и папок проекта

В фале prisma/schema.prisma описана модель данных (см. приложение А). При изменении моделей данных или связей между сущностями для синхронизации с PostgreSQL используется команда: npx prisma db push.

Файл .env содержит системные настройки, такие как DATABASE_URL путь к базе данных с указанием логина, пароля. Файл package.json содержит

внешние зависимости проекта [8], [14], [15].

В файле pages/api/order/[id].ts (см. приложение Б) содержится код, который отвечает за создание, редактирование, удаление, чтение заказов. Также были разработаны модули для работы с такими сущностями как пользователи, настройки, компоненты.

3.1.2 Страница авторизации

При входе в систему неавторизованный пользователь попадает на страницу авторизации рисунок 10.

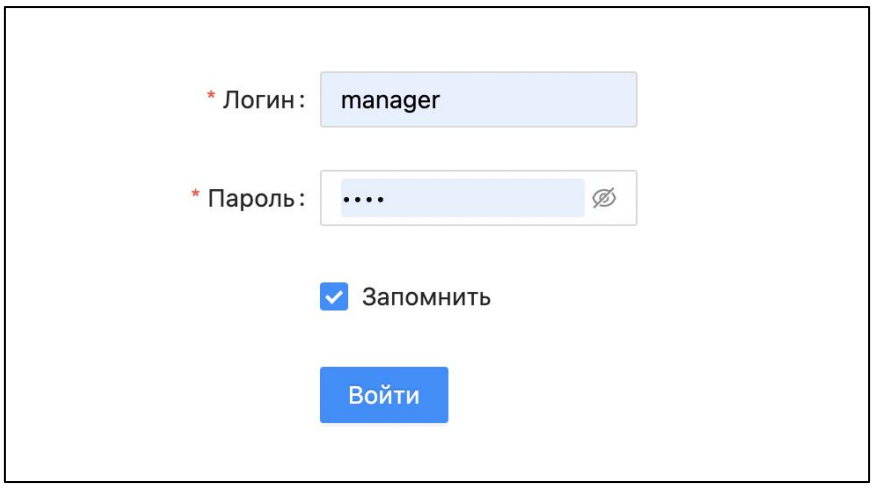

Рисунок 10 – Страница авторизации

Страница авторизации имеет форму для заполнения. Пользователь вводит свой логин и пароль и нажимает на кнопку «Войти». Данные отправляются на сервер. В случае успешной авторизации на сервере, сервер в ответе отправляет JSON Web Token (JWT), который сохраняется на стороне клиента и все последующие запросы происходят с отправкой этого токена в заголовке Authorization.

3.1.3 Страница списка заказов

После входа в систему пользователь видит страницу заказов (рисунок 11).

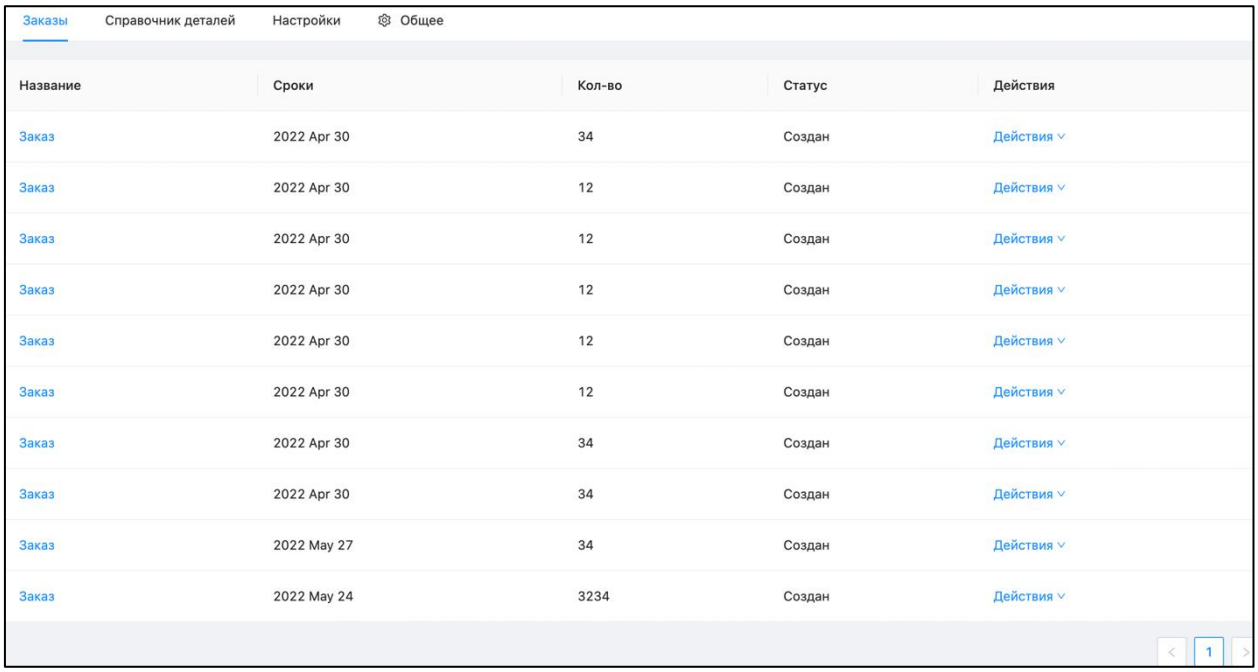

Рисунок 11 – Страница заказов

На странице заказов пользователь может просмотреть список заказов, создать новый заказ, отредактировать существующий. При клике на иконку в столбце «Действия» появляется список из доступных действий. Если заказ находиться в статусе ready (Завершен), он может быть отредактирован или удален только сотрудником с ролью admin.

3.1.4 Страница создания заказа

При нажатии на кнопку «Создать заказ» пользователь переходит на соответствующую страницу (рисунок 12).

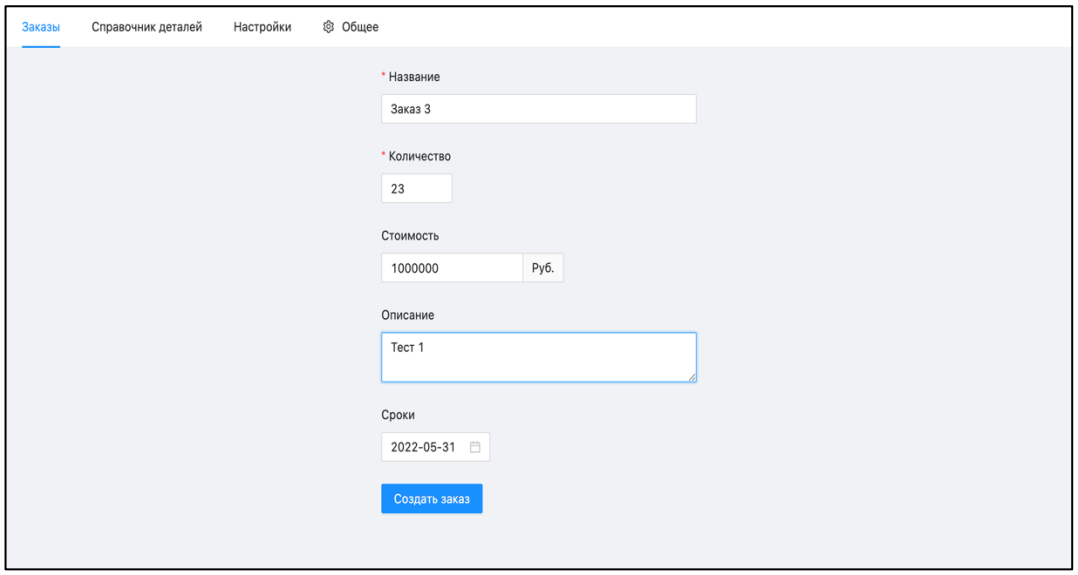

Рисунок 12 – Страница создания заказа

Заказ создает менеджер и некоторые поля на этом этапе не доступны. Менеджер заполняет форму добавляя количество необходимой продукции и описание заказа, вводит предварительные сроки.

3.2 Тестирование информационной системы

3.2.1 Способы тестирования информационной системы

Тестирования информационной системы позволяет контролировать качество релизов системы. Почти любая программа разрабатывается поэтапно, и на каждом этапе нужно сохранить в рабочем состоянии функционал, написанный ранее. Этого не всегда просто достичь, так как некоторые компоненты программы могут иметь тесную связанность друг с другом.

Тестирование может быть автоматическим и ручным. При ручном тестировании (manualtesting) вручную выполняются тесты. Ручное тестирование не требует серьезных знаний о технологиях на основе которые была разработана системы.

Автоматизированное тестирование предполагает использование

специального программного обеспечения (помимо тестируемого) для контроля выполнения тестов и сравнения ожидаемого фактического результата работы программы. Этот тип тестирования помогает автоматизировать часто повторяющиеся, но необходимые для максимизации тестового покрытия задачи.

Есть несколько видов автоматического тестирования:

- автоматизация тестирования кода (Code-driven testing) – тестирование на уровне программных модулей, классов и библиотек (фактически, автоматические юнит-тесты);
- автоматизация тестирования графического пользовательского интерфейса (Graphical user interface testing) – специальная программа (фреймворк автоматизации тестирования) позволяет генерировать пользовательские события – нажатия клавиш, клики мышкой, и отслеживать реакцию программы на эти действия – соответствует ли она спецификации;
- автоматизация тестирования API (Application Programming Interface). Тестируются интерфейсы, предназначенные для взаимодействия, например, с другими программами или с пользователем. Здесь опять же, как правило, используются специальные фреймворки.

Далее будет рассмотрен пример автоматического тестирования пользовательского интерфейса и ручное тестирование API.

3.2.2 Автоматизация тестирования пользовательского интерфейса

Для автоматического тестирование пользовательского интерфейса можно использовать такой инструмент как Cypress. При помощи Cypress можно писать End-to-end, Unit и Интеграционные тесты. Он относительно прост в изучении и настройке [7].

Перед установкой Cypress нужно установить Node.JS и менеджер пакетов npm, сам Cypress устанавливается при помощи команды: npm install cypress --save-dev.

На рисунке 13 приведен пример теста на заполнения формы авторизации.

```
3
4   describe("Тест формы авторизации", () => {
5     it("Можно ли заполнить форму", () => {
6       cy.visit("http://localhost:8080/");
7       cy.get("form");
8
9       cy.get('input[name="name"]').type("Олег");
10      cy.get('input[name="password"]').type("qwerty12345");
11    });
12  });
```

Рисунок 13 – Тест на возможность заполнения формы

Тест будет пройден если удастся перейти на страницу, которая будет доступна по адресу http://localhost:8080/, получить форму и заполнить имя и пароль в этой форме.

Тесты написаны на следующие пользовательские сценарии (Use case):

– у пользователя должна быть возможность авторизации в системе, для этого он должен пройти на страницу авторизации заполнить форму и нажать на кнопку «Войти», после чего в случае успешной авторизации, он будет автоматически перенаправлен на страницу заказов;
– у пользователя должна быть возможность просмотра заказов. Перейдя на страницу списка заказов и кликнув на конкретный заказ, пользователь должен увидеть всю информацию по выбранному заказу на новой странице;
– у пользователя должна быть возможность создания заказов. Для этого, находясь на странице заказов, он должен нажать на кнопку «Создать заказ». После чего он будет перенаправлен на форму создания заказа. Заполнив обязательные поля формы и нажав на кнопку формы «Создать заказ», пользователь должен быть перенаправлен обратно на страницу списка заказов. На странице

заказов должен присутствовать новый заказ;

– у пользователя должна быть возможность редактирования заказа в статусе отличном от ready. Для этого на странице списка заказов, он должен кликнуть на иконку «Действия», и в выпадающем списке действий выбрать редактирование. После чего он будет перенаправлен на форму редактирования заказа. Отредактировав заказ и нажав на кнопку «Сохранить», пользователь должен вернуться обратно на список заказов.

3.2.3 Ручное тестирование API

Ручное тестирование API можно проводить при помощи такого инструмента как Postman, интерфейс программы с запросом на получение списка заказов представлен на рисунке 14.

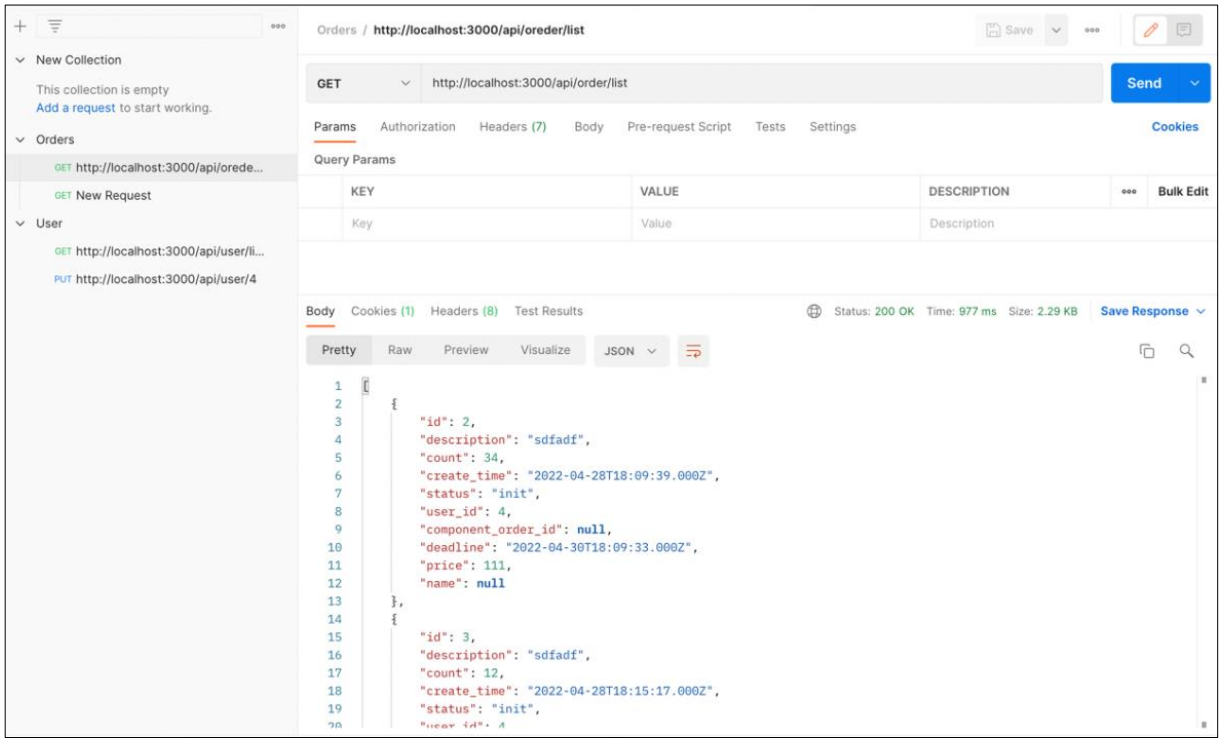

Рисунок 14 – Интерфейс программы Postman

При помощи Postman можно отправлять запросы на сервер и получать ответы от сервера. Любой запрос пользователя на сервер можно

воспроизвести при помощи этого инструмента.

Если нужно протестировать ответ на получение списка заказов, можно запустить сервер локально на машине разработчика или на удаленной машине. Запрос на получения списка будет выглядеть следующим образом GET {host}/orders/list. После отправки этого запроса должен прийти JSON-файл со списком заказов.

Перед релизом продукта необходимо проверить работоспособность API, для этого отправляются запросы на создание, удаление, обновление и чтение таких сущностей как заказы, пользователи, компоненты, настройки.

3.2.4 Пример ручного тестирование модуля заказов

На рисунке 15 был приведен пример успешного ответа на запрос получения списка заказов. На рисунке 15 приведен пример успешного ответа на запрос создания заказа. Запрос был успешен так как получен статус 201 и сообщением «row inserted with id - 14». При повторном запросе списка заказов GET http://loclhost:3000/api/order/list заказ с id равным 14 присутствует в списке заказов.

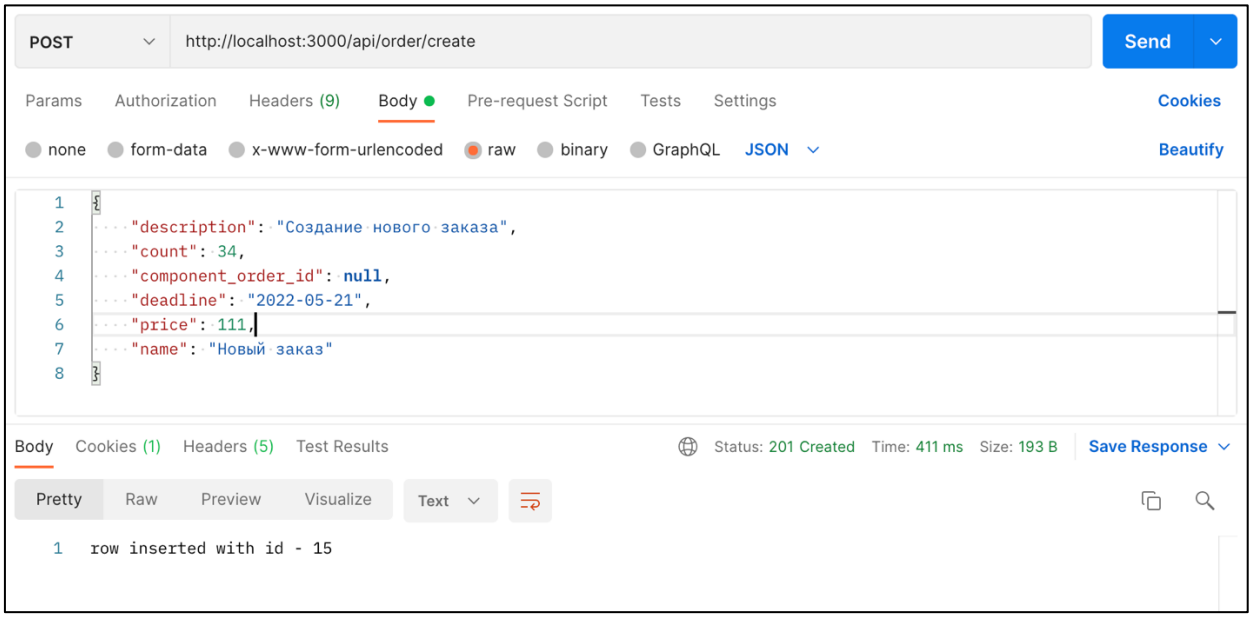

Рисунок 15 – Ответ на запрос создание заказа

У этого же заказа с id равным 14 были изменены поля description, count, price, name. На рисунке 16 приведен успешный ответ на запрос изменения полей заказа.

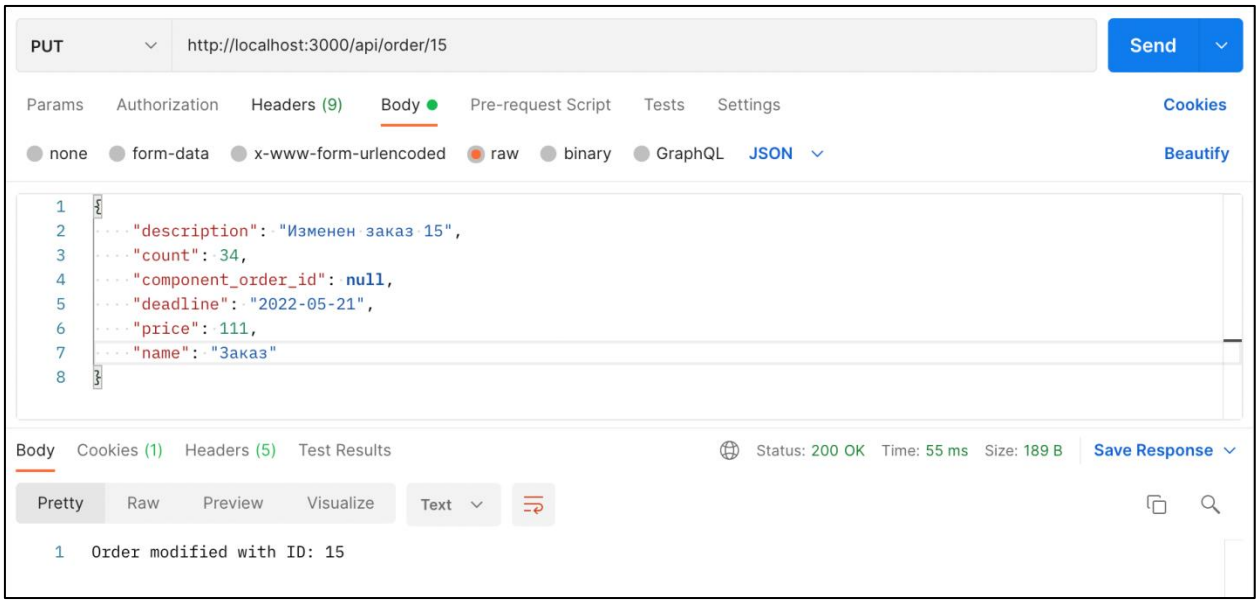

Рисунок 16 – Ответ на запрос редактирования заказа

На рисунке 17 приведен успешный ответ на запрос удаления заказа с id, равным 14.

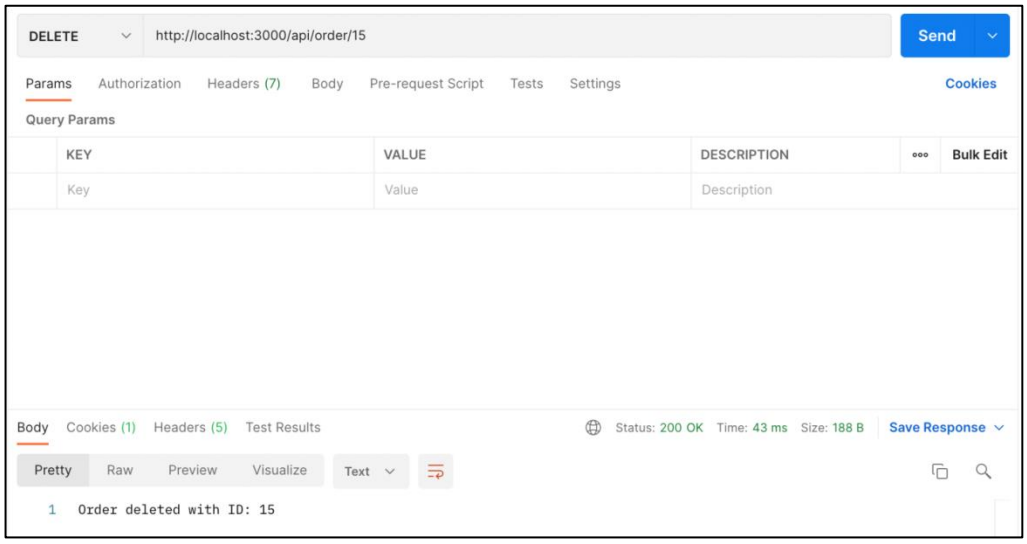

Рисунок 17 – Ответ на запрос удаление заказа

Заказ был успешно удален и больше не присутствует в списке заказов. Повторный вызов запроса на удаление заказа с id равным 14 вызовет ошибку.

3.3 Расчет экономической эффективности

3.3.1 Выбор методики расчета экономической эффективности

Чтобы оценить экономическую эффективность разработки, внедрения и дальнейшей поддержки информационной системы нужно:

- выяснить стоимость разработки проекта;
- сравнить показатели экономической эффективности текущего решения и внедряемого;
- сделать вывод на основе сравнения эффективности.

Задачи, которые необходимо решить для разработки информационной системы приведены в таблице 2.

Таблица 2 – Задачи для разработки информационной системы

Задачи	Затраты, дней
Разработка требований	5
Разработка макетов пользовательского интерфейса	5
Разработка Backend-а	10
Разработка Frontend-а	5
Тестирование и отладка системы	5

Помимо трудовых ресурсов должны учитываться и материальные затраты. При расчете стоимости разработки предполагается, что у разработчика уже есть ПК, на котором ведется разработка и предустановленное ПО. Сотрудник работает удаленно, и компания не тратит деньги на офис.

Если считать, что рабочий день длится 8 часов, то можно оценить общее количество человеко-часов на разработку системы. Использую данные из таблицы 3, получаем 240 человеко-часов. В таблице 3 приведены зарплатные расходы на разработку исходя из профиля сотрудника, времени и средней зарплаты согласно статистике сервиса Хабр Карьера за второе полугодие 2021 года.

Относительно таблицы 3 стоит отметить несколько важных моментов:

- среднее количество отработанных сотрудником часов в месяц 160;
- сотрудники оформлены в штат компании;
- в таблице приведена медианная зарплата в рублях в месяц после уплаты налогов.

Итог подсчитан по следующему правилу: медианная зарплата в месяц, деленная на 160 и умноженная на количество человеко-часов.

Таблица 3 – Зарплатные расходы на разработку

Роль	Человеко-часов	Медианная зарплата в месяц в рублях	Итого
Full Stack разработчик	120	150 000	112 500
Тестировщик	40	105 000	26 250
Дизайнер	40	64 000	16 000
Аналитик	40	121 000	30 250
Общий итог после налогов			185 000
Приблизительный итог до налогов			264 550

Итоговую сумму до уплаты налогов можно подсчитать приблизительно исходя из НДФЛ в 13% и остальных взносов в 30%. К остальным взносам относятся взносы в Пенсионный фонд (ПФР), Фонд обязательного медицинского страхования (ФОМС), Фонд социального страхования (ФСС),

Фонд социального страхования за травматизм (ФСС). Нужно учитывать, что в законодательстве предусмотрены разные процентные ставки по налогу и взносам в зависимости от статуса работника и других факторов. Поэтому ставка в 30% является приблизительной.

Таким образом, можно считать, что стоимость разработки информационной системы составляет 264550 рублей.

3.3.2 Расчет экономической эффективности проекта

Оценка экономической эффективности дает возможность выбрать наиболее подходящее решение из нескольких доступных вариантов. Нужно учитывать, что в процессе автоматизации процессов может быть достигнуто значительное снижение трудоемкости конкретных задач, рост производительности труда, прямое или косвенное улучшение условий работы сотрудников компании. И это в свою очередь сказывается на конкурентоспособности фирмы [3].

Экономическая эффективность включает в себя улучшение финансовой хозяйственной и производственной работы фирмы, уменьшение затрат при обработке информации. Оценка экономической эффективности проекта заключается в определении разницы трудовых и стоимостных показателей между внедряемым подходом и текущим.

К трудовым характеристикам относится ΔТ – абсолютное снижение трудовых затрат за год (1).

$$\Delta T = T0 - T1, \quad (1)$$

где Т0 – трудовые затраты рассчитанные в часах за год при использовании текущего решения;

Т1 – трудовые затраты рассчитанные в часах за год при использовании нового решения.

Получаем: ΔТ = 5 928 - 5187 = 741 ч.

На текущий момент в подразделении есть минимальный набор

сотрудников, которые необходимы для осуществления бизнес-процесса. Это менеджер по работе с клиентами, сотрудник склада, сборщик. Внедряемая система позволяет сократить время на обработку информации для каждого из сотрудников в среднем на час в день.

Коэффициент относительного снижения трудовых затрат КТ рассчитаем по формуле (2).

$$КТ = \Delta Т / Т0 * 100\% \quad (2)$$

Получаем: КТ = 741 / 5 928 * 100% = 12,5%.

Индекс, показывающий уменьшение трудовых затрат или повышение производительности труда YT, можно получить, используя формулу (3).

$$YT = Т0 / Т1 \quad (3)$$

Результат расчета: YT = 5928 / 5187 = 1,14.

Абсолютное снижение стоимостных затрат ΔС в рублях за год, может быть рассчитано по формуле (4).

$$\Delta С = С0 - С1, \quad (4)$$

где С0 – стоимостные затраты, рассчитанные в рублях за год при использовании текущего решения;

С1 – стоимостные затраты, рассчитанные в рублях за год при использовании нового решения;

Получаем:

ΔС = 5 928 * 350 - 5187 * 350 = 2 074 800 - 1 815 450 = 259 350 р.

Здесь 350 – это среднее количество рублей, которые получает сотрудник за час работы.

Для расчета коэффициента относительного снижения стоимостных затрат КС, используем формулу (5).

$$КС = \Delta С / С0 * 100\% \quad (5)$$

Получаем: КС = (259 350 / 2 074 800) * 100% = 12,5%

Индекс снижения стоимостных затрат YC рассчитаем, используя формулу (6).

$$YC = С0 / С1 \quad (6)$$

Результат расчета: YC = 2 074 800 / 1 815 450 = 1,14.

Также необходимо рассчитать срок окупаемости затрат на внедрение проекта машинной обработки информации ($Т_{ОК}$). Используем формулу (7).

$$Т_{ОК} = КП / \Delta С \quad (7)$$

где КП – 264 550 рублей затраты на создание проекта (проектирование и внедрение) из таблицы 4 раздела 3.3.1.

$$Ток = 264\ 550 / 259\ 350 = 1,02.$$

Сводные результаты расчетов представлены в таблице 4.

Таблица 4 – Показатели эффективности от внедрения проекта автоматизации

	Затраты		Абсолютное изменение затрат	Коэффициент изменения затрат	Индекс изменения затрат
	Текущее решение	Внедряемое решение			
Трудоемкость	Т0 (час)	Т1 (час)	ΔТ = Т0 – Т1 (час)	КТ= ΔТ /Т0*100%	YT=T0/T1
	5 928	5187	741	12.5	1,14
Стоимость	С0 (руб.)	С1 (руб.)	ΔС = С0 - С1 (руб.)	КС = ΔС / С0 * 100%	YC = С0 / С1
	2 074 800	1 815 450	259 350	12,5	1,14

Согласно приведенным расчетом срок окупаемости проекта составит примерно год.

Не все улучшения можно оценить с экономической точки зрения. После внедрения средств автоматизации процессов, хранение и поиск информации становится относительно дешевым как с точки зрения финансов, так и временных затрат сотрудника. Это дает возможность использовать информацию для решения спорных вопросов с клиентами, постпродажного обслуживания, сбора статистики, масштабирования системы. Также некоторые улучшения, связанные с автоматизацией, могут быть основой для других дешевые в реализации улучшений, что в моменте дает конкурентное преимущество фирме.

Выводы по главе 3

В третьей главе описаны технологии, на основе которых создан проект и методы тестирования информационной системы. Приведено экономическое обоснования внедряемого решения, проведенные расчеты показали, что автоматизация экономически обоснована. Автоматизация позволяет значительно сократить время на обработку информации сотрудниками организации и уменьшить количество ошибок.

Заключение

В процессе написания ВКР были проанализированы бизнес-процессы подразделения организации и были выявлены процессы, которые нуждаются в автоматизации. К таким процессам можно отнести обработку заявок на сборку продукции, обработку заявок на получения комплектующих и сам процесс сборки продукции. Было принято решение о создании информационной системы, которая могла бы оптимизировать данные процессы и сократить количество ошибок со стороны сотрудников фирмы. Был разработан только тот функционал, который нужен для выполнения текущих бизнес-процессов фирмы. Простая в использовании система дала возможность компании сократить время на обучение персонала и оптимизировать часто используемые сценарии работы с программой.

Была разработана информационная система, которая уменьшила время на обработку информации и формирования отчетности, а также сократила количество ошибок, допускаемых сотрудниками в процессе заполнения различных форм. Для быстроты и удобства был выбран язык программирования JavaScript и система управления БД PostgreSQL, система контроля версий Git, редактор кода Visual Studio Code. JavaScript фреймворк Next.JS и ORM Prisma. Это позволило значительно уменьшить время на разработку и реализовать гибкую, удобную систему управления заказами.

Разработаны пользовательские тесты, которые тестирует как отдельные модули системы, так и работу системы в целом. На этапе разработки автоматическое тестирование позволило контролировать работоспособность системы в каждом из релизов, а при дальнейшем усовершенствовании системы позволит сохранить качество продукта на прежнем уровне.

Приведено экономическое обоснование реализации проекта, описана система расчета. Согласно этим данным, примерный срок окупаемости проекта составит один год.

Список используемой литературы и используемых источников

1. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 624 с.

2. Бёрнс Б. Распределенные системы. Паттерны проектирования. — СПб.: Питер, 2019. – 224 с.

3. Братченко С.А. Бизнес-планирование как эффективный инструмент управления компанией: монография. – М.: Издательский дом «НАУЧНАЯ БИБЛИОТЕКА», 2016. – 172 с.

4. Вайсфельд М. Объектно-ориентированный подход. 5-е межд. изд. — СПб.: Питер, 2020. – 256 с.

5. Голицына О. Системы управления базами данных: учеб. пособие / О. Л. Голицына, Т. Л. Партыка, И. И. Попов. – Гриф МО. – М.: ФОРУМ – ИНФРА-М, 2016. – 431 с.

6. Гриценко Ю.Б. Архитектура предприятия: учеб. пособие / Ю.Б. Гриценко. – Томск: Изд-во Томск. гос. ун-та систем управления и радиоэлектроники, 2014. – 260 с.

7. Джонсон Д., Деоган Д., Савано Д. Безопасно by design. — СПб.: Питер, 2021. - 432 с.

8. Дж. Клейнберг Дж., Е. Тардос. Алгоритмы. Разработка и применение. – Питер, 2106. – 800 с.

9. Домбровская Г., Новиков Б., Бейликова А. Оптимизация запросов в PostgreSQL / пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2022. – 278 с.

10. Ёсу М., Вальдуриес П. Принципы организации распределенных баз данных / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2021. – 672 с.

11. Каскиаро М., Маммино Л. Шаблоны проектирования Node.js / пер. с анг. А. Н. Киселева. – М.: ДМК Пресс, 2017. – 396 с.

12. Линц Г. Радикальное изменение бизнес-модели: Адаптация и выживание в конкурентной среде / Карстен Линц, Гюнтер Мюллер-Стивенс,

Александр Циммерман; Пер. с англ. – М.: Альпина Паблишер, 2019. – 311 с.

13. Моуэт Э. Использование Docker / пер. с англ. А. В. Снастина; науч. ред. А. А. Маркелов. – М.: ДМК Пресс, 2017. – 354 с.

14. Павлов Л. Структуры и алгоритмы обработки данных: учебник / Л. А. Павлов, Н. В. Первова. – 2-е изд., испр. и доп. – Санкт-Петербург: Лань, 2020. – 256 с.

15. Стоянович С., Симович А. Бессерверные приложения на JavaScript / пер. с англ. А. Н. Киселева. – М.: ДМК Пресс, 2020. – 394 с.

16. Сьоре Э. Проектирование и реализация систем управления базами данных / пер. с анг. А. Н. Киселева; научн. ред. Е. В. Рогов. – М.: ДМК Пресс, 2021. – 466 с.

17. Фрисби М. JavaScript для профессиональных веб-разработчиков. 4-е международное изд. – СПб.: Питер, 2022. – 1168 с.

18. Шёнталер Ф. Бизнес-процессы: Языки моделирования, методы, инструменты / Франк Шёнталер, Готфрид Фоссен, Андреас Обервайс, Томас Карле; пер. с нем. – М.: Альпина Паблишер, 2019. - 499 с.

19. A JavaScript library for building user interfaces [Электронный ресурс]. URL: https://reactjs.org/ (дата обращения: 10.04.2022).

20. Next-generation Node.js and TypeScript ORM [Электронный ресурс]. URL: https://www.prisma.io/docs/ (дата обращения: 07.05.2022).

21. The React Framework for Production [Электронный ресурс]. URL: https://nextjs.org/ (дата обращения: 07.05.2022).

Приложение А

Prisma модель данных

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
model ComponentOrderItems {
  id                Int            @id @default(autoincrement())
  count             Int
  component_id      Int
  component_order_id Int
  price             Float
  Components        Components     @relation(fields: [component_id], references: [id], onDelete: NoAction, onUpdate: NoAction, map: "component")
  ComponentOrders   ComponentOrders @relation(fields: [component_order_id], references: [id], onDelete: NoAction, onUpdate: NoAction, map: "component_order")
}

model ComponentOrders {
  id                Int                @id @default(autoincrement())
  create_time       DateTime           @default(now()) @db.Timestamp(6)
  status            components_status
  ComponentOrderItems ComponentOrderItems[]
  Orders            Orders[]
}

model Components {
  id                Int                @id @default(autoincrement())
  name              String?            @db.VarChar(255)
```

```
  description         String?              @db.VarChar(255)
  price               Float                @default(0)
  ComponentOrderItems ComponentOrderItems[]
 }

 model Orders {
  id                Int              @id @default(autoincrement())
  description       String?          @db.VarChar(255)
  count             Int
  create_time       DateTime         @default(now()) @db.Timestamp(0)
  status            order_status
  user_id           Int
  component_order_id Int?
  deadline          DateTime?        @db.Timestamp(6)
  price             Float            @default(0)
  name              String?          @db.VarChar(255)
  ComponentOrders   ComponentOrders? @relation(fields: [component_order_id],
references: [id], onDelete: Restrict, map: "components")
  Users             User             @relation(fields: [user_id], references: [id], map: "user")
 }

 model Session {
  id           String   @id @default(cuid())
  sessionToken String   @unique @map("session_token")
  user_id      Int  @map("user_id")
  expires      DateTime
  user         User     @relation(fields: [user_id], references: [id], onDelete: Cascade)

  @@map("Sessions")
 }
 model User {
  id          Int       @id @default(autoincrement())
```

```
  email       String?   @unique
  login       String    @db.VarChar(255) @unique
  password    String    @db.VarChar(255)
  create_date DateTime? @default(now()) @db.Timestamp(6)
  role        role
  name        String    @db.VarChar(255)
  Orders      Orders[]
  sessions    Session[]

  @@map("Users")
}

model VerificationToken {
  identifier String
  token      String   @unique
  expires    DateTime

  @@unique([identifier, token])
  @@map("Verificationtokens")
}
model Settings {
  id         String   @id @default(cuid())
  param      String
  value      String

  @@unique([id, param])
  @@map("Setting")
}

enum components_status {
  init
  wait_components
```

```
  ready
}

enum mood {
  initial
  atWork
  finished
}

enum order_status {
  init
  wait_components
  assembling
  ready
}

enum role {
  admin
  manager
  developer
  storekeeper
}
```

Приложение Б

Код, отвечающий за работу с заказами

```
import { NextApiResponse, NextApiRequest } from "next";
import { Prisma } from '@prisma/client'
import prisma from '../../../lib/prisma';
import { getSession } from "next-auth/react";

export enum EOrderStatus {
  INIT = "init",
  WAIT_COMPONENTS = "wait_components",
  ASSEMBLING = "assembling",
  READY = "ready",
}
export enum ERoles {
  ADMIN = "admin",
  MANAGER = "manager",
  DEVELOPER = "developer",
  STOREKEEPER = "storekeeper",
}

interface IOrder {
  name: string;
  description: string;
  count: number;
  user_id: number;
  component_order_id: number;
  deadline: string;
  price: number;
  create_time: string;
  status: EOrderStatus;
}

const isId = (str: string) => {
```

```
  let id = parseInt(str, 10);

  if (!Number.isInteger(id)) {
    return false;
  }

  return true;
};

const createItem = async (data: IOrder) => {
  const {
    name,
    description,
    count,
    user_id,
    component_order_id,
    deadline,
    price,
  } = data;
  const order = await prisma.orders.create({
    data: {
      name,
      description,
      count,
      status: EOrderStatus.INIT,
      user_id,
      component_order_id,
      deadline,
      price
    },
  })

  return order;
```

```
};
const getItem = async (id: number): Promise<Prisma.OrdersMinAggregateOutputType |
null> => {
  const item = await prisma.orders.findUnique({
   where: {
    id
   }
  });

  return item;
};

const getList = async (): Promise<Prisma.OrdersMinAggregateOutputType[]> => {
  const items = await prisma.orders.findMany();

  return items;
};

const updateItem = async (id: number, data: IOrder) => {
  const { name, description, price } = data;

  const item = await prisma.orders.update({
   where: {
    id
   },
   data: {
    name,
    description,
    price
   }
  });
```

```
  return item;
};

const deleteItem = async (id: number) => {
  const item = await prisma.orders.delete({
    where: {
      id
    }
  });

  return item;
};

async function createOrder (req: NextApiRequest, res: NextApiResponse, session: any) {
  const {
    query: { id },
    body,
    method,
  } = req;

  if (session.user.role !== ERoles.MANAGER) {
    res.status(500).send("Only manager can create orders!");
  }

  if (!body.name || !body.price) {
    res.status(500).send("Fieds name & price are required!");
  }

  const item = await createItem(body);

  res.status(201).send(`row inserted with id - ${ item.id }`);
}
```

```
async function deleteOrder (req: NextApiRequest, res: NextApiResponse, session: any) {
  const {
    query: { id },
    body,
    method,
  } = req;

  if (!Array.isArray(id) && isId(id)) {
    await deleteItem(parseInt(id));
    res.status(200).send(`component deleted with ID: ${id}`);
  } else {
    throw new Error("id is undefined!");
  }
}

async function changeOrder (req: NextApiRequest, res: NextApiResponse, session: any)
{
  const {
    query: { id },
    body,
    method,
  } = req;

  if (!body.name || !body.price) {
    throw new Error("Fieds name & price are required!");
  }

  if (!Array.isArray(id) && isId(id)) {
    const result = await updateItem(parseInt(id), body);
    res.status(200).send(`Component modified with ID: ${id}`);
  } else {
    throw new Error("Id is undefined!");
```

```
  }
}
async function getOrderList (req: NextApiRequest, res: NextApiResponse, session: any)
{
  const {
    query: { id },
    body,
    method,
  } = req;
  const result: any = await getList();
  res.status(200).json(result);
}
async function getOrderItem (req: NextApiRequest, res: NextApiResponse, session: any)
{
  const {
    query: { id },
    body,
    method,
  } = req;
  if (!Array.isArray(id) && isId(id)) {
    const result: any = await getItem(parseInt(id));
    res.status(200).json(result);
  } else {
    throw new Error("id is undefined!");
  }
}
export default async (req: NextApiRequest, res: NextApiResponse) => {
  const {
    query: { id },
    body,
    method,
} = req;
  const session = await getSession({ req });
```

```
 if (!session) {
  res.status(500).send("Unautorized user!");
 }
 try {

  if (method === "GET") {

   if (id === "list") {
    return getOrderList(req, res, session);
   }

   return getOrderItem(req, res, session);
  }

  if (method === "POST") {
    return createOrder(req, res, session);
  }

  if (method === "DELETE") {
   return deleteOrder(req, res, session);
  }

  if (method === "PUT") {
        return changeOrder(req, res, session);
  }
  res.setHeader("Allow", ["GET", "DELETE", "POST", "PUT"]);
  res.status(405).end(`Method ${method} Not Allowed`);

 } catch (err: any) {
  res.status(500).send(err && err.message);
  return;
 }
};
```