Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего профессионального образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника

Автоматизированное рабочее место мастера по ремонту компьютерной техники

Преподаватель _____ С. Ю. Пичковская

Руководитель ВКР _____ С. Н. Титовский
подпись, дата

Студент ЗКИ18– 07Б, 031838262 _____ А. В. Дешеков
номер групп, зачетной книжки *подпись, дата*

Красноярск 2023

СОДЕРЖАНИЕ

РЕФЕРАТ

Выпускная квалификационная работа по теме «Автоматизированное рабочее место мастера по ремонту компьютерной техники» содержит 59 страниц текстового документа, 32 рисунка, 1 таблица, 3 приложения, 21 использованный источник, презентацию из 11 слайдов.

АВТОМАТИЗАЦИЯ, ПРОГРАММА, ПРОЕКТИРОВАНИЕ, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, POSTGRESQL, REACT.

Цель – создание автоматизированного рабочего места для мастера по ремонту компьютерной техники.

Для достижения цели поставлены следующие задачи:

- Обзор предметной области;
- Выбор средств разработки;
- Разработка базы данных;
- Разработка серверной части системы;
- Разработка клиентской части системы.

ВВЕДЕНИЕ

В настоящее время наблюдаются тенденции развития сетей, в том числе Интернета. Общество проводит все больше и больше времени в глобальной сети. Возможно, это связано с автоматизацией работы человека.

Автоматизированное рабочее место – программно-технический комплекс, предназначенный для автоматизации деятельности определенного вида.

Автоматизация стремительно проникает во все сферы промышленности и сферы услуг. Не исключением является сфера ремонта компьютерной техники.

Большой интерес к автоматизации возникает из-за желания минимизировать потери рабочего времени каждого члена компании, сэкономить на их численности, исключить ошибки, поднять уровень сервиса.

Автоматизация некоторых обязанностей работника – большой шаг к более быстрой и качественной работе целой компании.

Для начала стоит выявить требования, характерные для отрасли ремонта компьютерной техники. Автоматизация должна позволять легко производить расчёты и контролировать имеющиеся и закупаемые материалы, необходимые для выполнения заказов.

Ведение истории выполненных заказов позволит удобно просматривать прошлые заказы и оформлять отчёты по суммарной выручке за разные периоды времени.

Быстрое и удобное оформление заказов, выдача справок клиентам – всё это позволяет повысить уровень обслуживания. Автоматизация ускорит работу, а значит становится возможным увеличить количество клиентов и, соответственно, прибыль.

Целью данной работы является создание автоматизированного рабочего места для мастера по ремонту компьютерной техники.

Для достижения цели поставлены следующие задачи:

- Обзор предметной области;

- Выбор средств разработки;
- Разработка базы данных;
- Разработка серверной части системы;
- Разработка клиентской части системы.

1 Общие сведения

1.1 Анализ предметной области

Автоматизированная информационная система (АИС) – это комплекс специализированных компьютерных программ, предназначенных для сбора и обработки информации и позволяющие человеку выполнять его производственные задачи при минимальном участии. Основной целью автоматизированной системы является хранение, поиск, передача информации по запросам пользователей.

В настоящее время деятельность большого количества сервисных центров уже автоматизирована. Для этого используются как готовые программные продукты в данной сфере, так и разрабатываются новые индивидуальные программы.

Автоматизация сервисного центра зависит и от специфики работы, так как есть сервисные центры, которые специализируются на ремонте одного вида техники, а есть и многопрофильные сервисные центры. Поэтому при внедрении АРМ необходимо учитывать основные бизнес-процессы предприятия:

- связанные с ремонтом техники (прием на ремонт, помещение на склад, сроки ремонта и т.д.);

- связанные с клиентами (регистрация в базе, оповещение, оплата);

- связанные с заказом материалов (оформление заявок, сроки, оплата);

- связанные с сотрудниками организации (учет рабочего времени, график, начисление з\п и другое);

- связанные с отчетностью (формирование отчетов, справок);

- связанные с дополнительными услугами (продажа аксессуаров, запасных частей). [1]

Автоматизация сервисного обслуживания в конечном итоге, позволит скоординировать все бизнес-процессы, обеспечит контроль за потоком информации.

Автоматизированные системы в сфере услуг по ремонту компьютерной техники используется для управления производственными процессами и имеют несомненные достоинства: экономическая выгода, оперативность, безопасность, комфортность общения, соответствие стандартам. Для мастера по ремонту компьютерной техники, в частности, позволит:

- увеличить эффективность работы;
- обеспечит экономию времени персонала и клиентов;
- легко контролировать заказы и оплаты от клиентов;
- вести учет расхода и поступления материалов;
- вести базы данных клиентов;
- создавать необходимые справки заказчикам. [2]

Функционирование мастерской по ремонту компьютерной техники связано использованием большого количества информации и документов, обработка которых вручную, в свою очередь, тормозит всю работу. На бумажных носителях затрудняется поиск и хранение информации об услугах и их свойствах, о клиентах и проведенных операциях с ними. Это связано с большим количеством операций, которые необходимо выполнить. Вследствие этого появляется необходимость автоматизации процесса ведения баз данных клиентов, учета и контроля поступающих заказов в мастерскую, учета расхода и поступления материалов. Очевидны преимущества и удобство поиска нужной информации о клиенте, например, по сравнению с поиском той же информации в бумажных папках. Точно также поиск информации о любой выполненной операции можно найти со всеми подробностями в электронной базе быстро и удобно.[2] Также неоспоримым преимуществом использования автоматизированной системы является безопасность и надежность хранения информации.

Производительность и эффективность любого предприятия и мастерской по ремонту компьютерной техники, в частности, зависит от программного обеспечения (ПО). При его разработке лучше руководствоваться конкретными требованиями к программе.

Программа автоматизированного рабочего места мастера по ремонту компьютерной техники автоматизирует работу мастера, позволит быстро находить нужную информацию о заказах, регистрировать новых клиентов, вести учет рабочего времени сотрудников, а также повысит скорость при выполнении операций.

1.2 Автоматизированное рабочее место

Автоматизированное рабочее место (АРМ) – совокупность программных и технических ресурсов, осуществляющих обработку данных и автоматизирующих функции управления в конкретной отрасли деятельности. Это место пользователя, специалиста, оборудованное технических и программным обеспечением и обеспечивает автоматизацию работы. Наиболее простые и распространённые варианты АРМ созданы на базе персональных компьютеров и дополнены различными вспомогательными электронными устройствами: накопителями, устройствами вывода данных и т.д. Также в АРМ используют разные операционные системы и прикладные программы, предназначенные для данного профиля деятельности пользователя. В большинстве своем, АРМ предназначены для пользователей, у которых отсутствуют специальные навыки пользования вычислительной техникой.

Автоматизированное рабочее место создается для рациональной управленческой деятельности, для решения конкретных задач и выполнения определенного набора функций. Создание автоматизированных рабочих мест предполагает, что основные операции по накоплению, хранению и переработки

информации возлагаются на вычислительную технику, а специалист выполняет часть операций вручную.

Техническое обеспечение АРМ предполагает надежность технических средств, организацию удобных для пользователя режимов работы, способность обработать в заданное время необходимый объем данных, должна быть обеспечена оперативность ввода, обработки и поиска документов, обмен информацией внутри организации и за ее пределами, безопасность для здоровья пользователя и комфортность обслуживания.[2]

Программное обеспечение АРМ должно быть ориентировано на профессиональный уровень пользователя и специализацию. В ПО входят операционные системы, сервисные программы, стандартные программы, прикладные программы.

АРМ выполняет следующие группы функций:

- Информационно-справочное обслуживание;
- Функции учета;
- Арифметические функции;
- Функции анализа и регулирования.

Самой распространенной функцией автоматизированного рабочего места является функция информационно-справочного обслуживания и особенность ее реализации зависит от нужд конечного пользователя.

Свойства АРМ:

- Доступность. Доступный для пользователя набор технических, программных и информационных инструментов;
- Возможность создания и сопровождение проектов автоматизированной обработки данных;

Осуществление пользователем обработки данных.

Для каждого объекта управления необходимо создание АРМ, которое соответствует назначению, однако принципы проектирования любых АРМ едины:

- Устойчивость. Этот принцип основывается на том, что система должна

продолжать выполнять свои функции независимо от воздействия на нее различных внутренних или внешних факторов;

- Системность. В соответствии с этим принципом АРМ рассматривается как система, структура которой определяется функциональным назначением;
- Гибкость. Данный принцип означает адаптируемость системы к возможным изменениям вследствие модульности построения всех подсистем и стандартизации их элементов;
- Эффективность. Этот принцип следует рассматривать как интегральный показатель уровня реализации всех принципов, отнесенный к затратам по созданию и эксплуатации системы.

Внедрение АРМ имеет следующие достоинства:

- автоматизация труда;
- экономия времени при принятии управленческих решений;
- повышение безопасности на производстве;
- повышение производительности труда.

Результативность АРМ зависит от того, как правильно распределены функции и нагрузки между специалистом и машинными средствами обработки информации. АРМ в этом случае не только повысит эффективность работы и производительность труда, но и обеспечит комфортабельность рабочего места специалиста. Но все же человек в системе АРМ остается главным звеном.

Использование автоматизированного рабочего места не только оптимизирует работу специалиста, но и уменьшается возможность ошибок, вследствие уменьшения влияния человеческого фактора.[3]

1.3 Обзор аналогов

На сегодняшний день среди готовых программ, предназначенных для сервисных центров, можно обратить внимание на следующие: Вулкан-М,

РемОнлайн, HelloClient, ServiceCRM, Gincore, LiveSklad, HubFx FSM, ServiceSpeedUP, WorkPan, РосБизнесСофт и др.

Рассмотрим некоторые из них.

Gincore – облачная программа для сервисного центра или мастерской, которая поможет устранить пересорт и недостачу на складах, привести в порядок бухгалтерские документы, повысить уровень обслуживания клиентов и повысить средний чек. [4] Различные отчеты помогут принять правильное решение.

Недостатком Gincore является то, что в этой системе нельзя оформлять заказы и вести историю выполненных заказов.

LiveSklad – облачное решение для автоматизации сервисного центра. Система позволит организовать финансовый учет, учет заказов и необходимых деталей, зарплаты и закупок. [5]

С помощью этой программы можно:

- принимать на ремонт изделие и отслеживать процесс ремонта до этапа выдачи клиенту;
- отслеживать истории по каждому заказу;
- вести учет клиентов;
- отправлять клиентам и сотрудникам СМС, email, Телеграмм уведомления о событиях в системе.

Доступен мгновенный поиск деталей по актуальным прайсам.

Программа позволяет рассчитать зарплату, снять остатки со склада, подготовить финансовую и налоговую отчетность. Программа синхронизируется с большинством моделей онлайн-касс, что позволит сэкономить на кассовом оборудовании.

Недостатком LiveSklad является то, что в этой системе нет возможности вести учет рабочего времени мастера.

HelloClient – облачная программа и мобильное приложение для сервисных центров, мастерских и бизнесов услуг, гибко подстраивается под особенности бизнеса. [6] Можно добавлять сотрудников, у каждого из которых свой логин и

пароль для входа в систему, можно подключить онлайн-кассу, интеграция со сканерами, мессенджерами, телефонией и др. есть онлайн-поддержка, доступна каждому пользователю.

В программе можно назначать ответственных за заказ сотрудников, добавлять работы и товары в заказы, устанавливать статусы и принимать оплату. Единая база клиентов с историей обращения и покупок, контроль всех денежных операций в программе.

Недостатком HelloClient является то, что нельзя вести учет расходуемых и закупаемых товаров.

Главным назначение подобного рода программ считают обработку информации на рабочих местах, использование баз данных. Все программы обладают проблемно-профессиональной ориентацией и позволяют специалисту осуществлять повторяющиеся операции. Создание программного продукта основывается на его функциональном назначении, учитывая общие принципы.

Сравнительная характеристика аналогов представлена в таблице 1.

Таблица 1 – Сравнительная характеристика аналогов

Характеристика	**Gincore**	**LiveSklad**	**HelloClient**
Учет материалов	+	+	-
Возможность просмотра истории заказов	-	+	+
Выдача справок клиентам	+	-	-
Создание заказов	-	+	+
Работа с клиентами	-	+	+
Счетчик рабочего времени	-	-	-

В результате сравнения аналогов, были выявлены недостатки существующих решений, которые будут исправлены в разрабатываемой системе.

1.4 Постановка задачи разработки

Разрабатываемая система будет содержать в себе следующую функциональность:

- учет материалов за счет ведения базы данных расходуемых и закупаемых материалов;
- просмотр истории выполненных заказов и составление финансовых отчетов за указанных промежуток времени;
- создание справок заказчику о том, что его заказ взят в работу;
- создание заказов;
- ведение базы данных клиентов;
- счетчик таймер рабочего времени.

2. Проектирование

На этапе проектирования необходимо разработать функциональную схему АРМ, диаграмму вариантов использования, выбрать архитектуру системы, провести сравнительный анализ средств разработки и выбрать подходящие под данную работу. Кроме этого, необходимо разработать базу данных.

2.1 Функциональная схема АРМ

Функциональная схема – графическая модель, которая показывает, какие процессы протекают в системе. [7]

На рисунке 1 представлена функциональная диаграмма IDEF0 для автоматизированного рабочего места мастера по ремонту компьютерной техники.

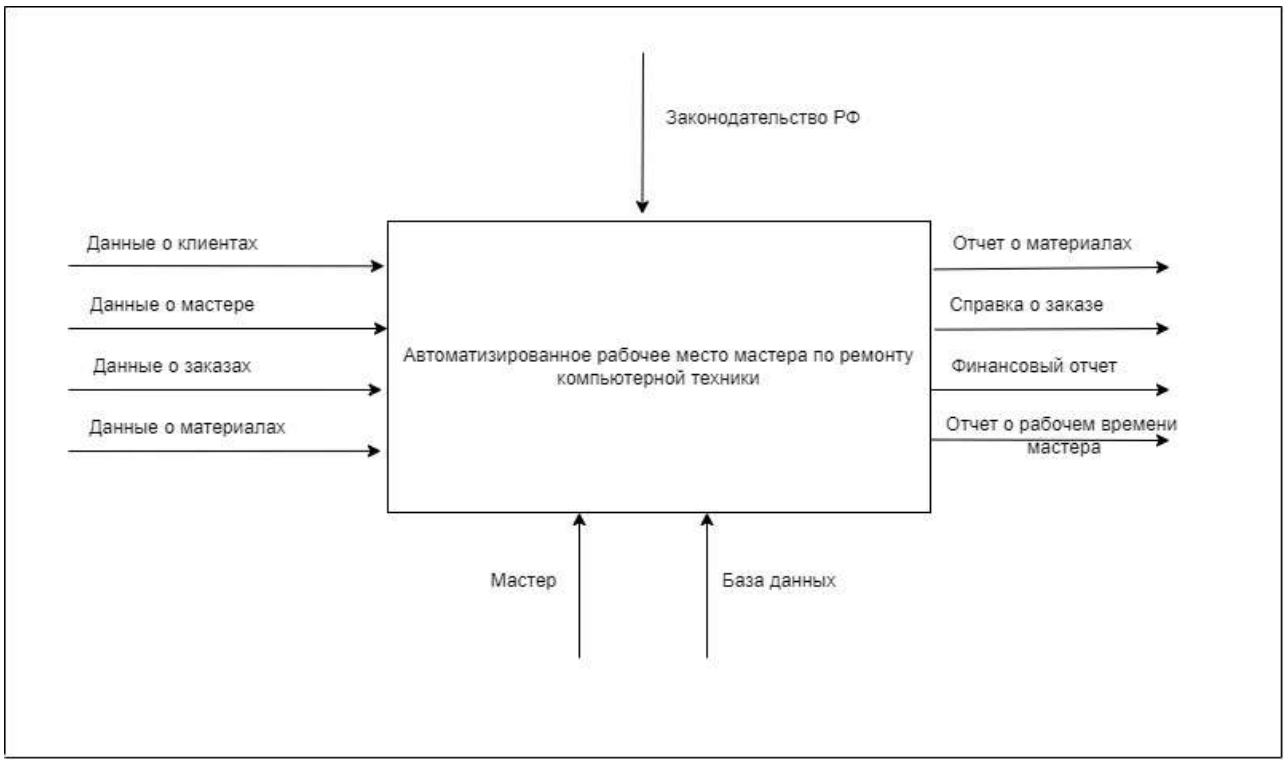

Рисунок 1 – Диаграмма IDEF0

На вход поступают данные о клиентах, заказах, материалах и мастере. Процесс управляется законодательством Российской Федерации. В процессе участвует мастер, так как эта информационная система разрабатывается для самозанятых специалистов по ремонту компьютерной техники, а также база данных. СУБД необходима для сохранения информации и генерации отчётов и справок. На выходе получаются отчеты о материалах, о рабочем времени, финансовый отчет, справки о заказах.

На рисунке 2 представлена декомпозиция диаграммы для получения справки о заказе.

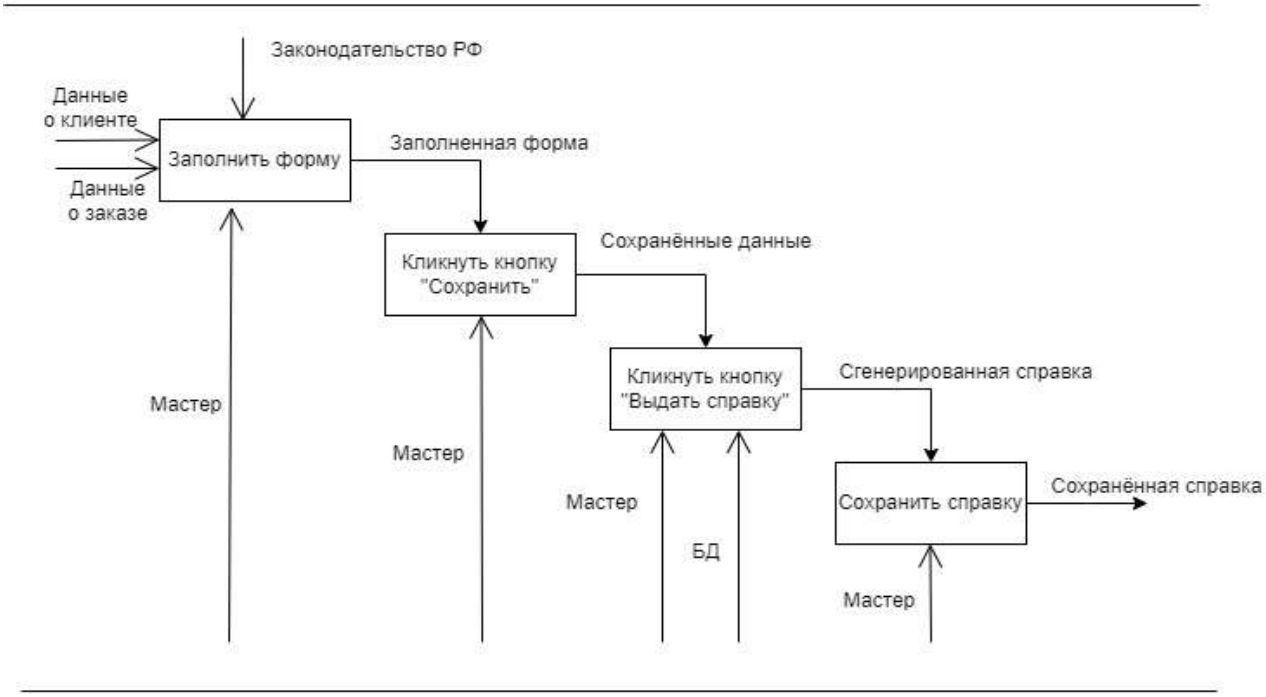

Рисунок 2 – Декомпозиция диаграммы IDEF0

Из диаграммы видно, как происходит получение справки о заказе. Сначала пользователю необходимо заполнить форму при оформлении заказа и сохранить ее. После этого необходимо кликнуть кнопку «Выдать справку». В это время система сделает запрос запрос в базу данных и сгенерирует справку. В конце мастеру нужно будет необходимо только сохранить ее.

2.2 Проектирование вариантов использования

Для автоматизированного рабочего места будет только одна группа пользователей – мастер по ремонту компьютерной техники. Диаграмма вариантов использования представлена на рисунке 3.[8]

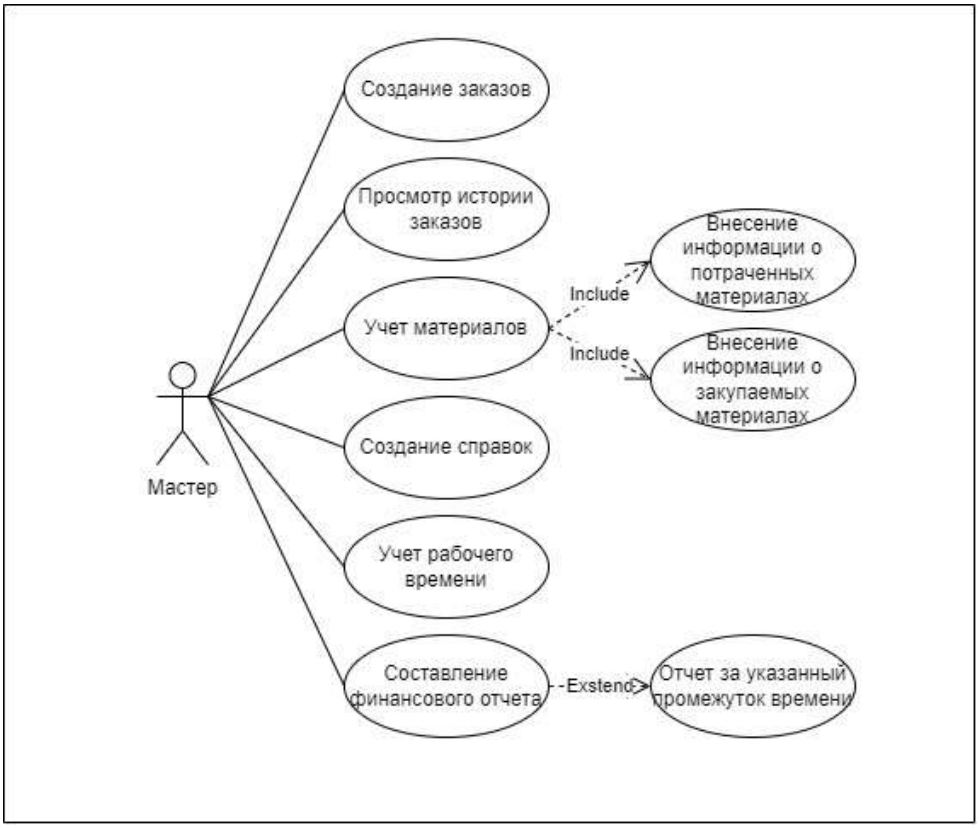

Рисунок 3 – Диаграмма вариантов использования

С помощью разрабатываемой системы пользователи смогут создавать заказы и выдавать справки клиентам о том, что заказ взять в работу, просматривать историю заказав, ведя при этом базу данных клиентов, вести учет рабочего времени. Также можно будет вести учет материалов, внося информацию о закупаемых материалах и расходуемых материалах. Кроме этого, автоматизированное рабочее место поможет составлять финансовые отчеты за указанные промежутки времени. Все это поможет автоматизировать рутинную работу мастера и освободить время на выполнение заказов клиентам.

2.3 Сравнительный анализ средств разработки

Для разработки серверной части стоял выбор между двумя самыми популярными языками программирования – PHP и Nest.js.

PHP и Nest.js – это два различных инструмента для разработки веб-приложений на разных языках программирования. PHP — это скриптовый язык программирования, который часто используется для создания серверных приложений.[9] Nest.js, с другой стороны, является фреймворком для создания масштабируемых и эффективных серверных приложений на языке JavaScript.[10]

Одним из главных преимуществ Nest.js является его модульная архитектура, которая обеспечивает легкую расширяемость и масштабируемость приложений. Nest.js также предлагает различные инструменты для тестирования и отладки приложений.

PHP также имеет широкое применение в веб-разработке благодаря своей простоте и поддержке большого количества веб-серверов и баз данных. Он также имеет большое сообщество разработчиков и множество готовых библиотек и фреймворков.

Однако, PHP не всегда может обеспечивать высокую производительность и масштабируемость приложений в сравнении с другими языками программирования, такими как Node.js.

В целом, выбор между PHP и Nest.js зависит от требований к производительности, масштабируемости и функциональности приложения, а также от опыта и предпочтений разработчика. Для разработки автоматизированного рабочего места мастера по ремонту компьютерной техники было отдано предпочтение Nest.js.

Для разработки клиентской части приложения выбор был между двумя самыми популярными фреймворками – Vue.js и React.[11]

Vue.js и React – это два из самых популярных фреймворков JavaScript для создания пользовательских интерфейсов. Оба фреймворка имеют свои преимущества и недостатки, и выбор между ними зависит от требований к проекту и предпочтений разработчика.

Vue.js отличается от React тем, что он имеет более простой синтаксис и легче в изучении.[12] Он также имеет меньший размер и быстрее в работе, что делает его привлекательным выбором для создания небольших и средних по размеру приложений.

React, с другой стороны, обладает более широким набором инструментов для разработки сложных и масштабируемых приложений.[13] Он также имеет большое сообщество разработчиков и множество готовых компонентов и библиотек.

В целом, выбор между Vue.js и React зависит от многих факторов, таких как размер проекта, требования к производительности и масштабируемости, опыт разработчика и предпочтения. Однако, оба фреймворка предоставляют инструменты для создания высококачественных пользовательских интерфейсов и являются отличным выбором для разработки веб-приложений.

Для разработки база данных системы выбор был из двух самых популяерных СУБД – MySQL и PostgreSQL.[14]

MySQL и PostgreSQL – две популярные реляционные системы управления базами данных (СУБД), которые обладают сходствами и различиями.

MySQL – это бесплатная и открытая СУБД, которая предоставляет простой и быстрый способ хранения и управления данными.[15] Она также поддерживает множество языков программирования и используется в таких приложениях, как WordPress и Drupal.

PostgreSQL, с другой стороны, обеспечивает более высокую производительность и надежность в сравнении с MySQL. Она также поддерживает множество функций, включая транзакции, целостность данных и многопользовательский доступ.[16]

Одним из главных отличий PostgreSQL от MySQL является то, что PostgreSQL поддерживает более сложные типы данных, такие как массивы, JSON и геометрические типы. Она также предоставляет различные инструменты для обеспечения безопасности и администрирования баз данных, включая механизмы аутентификации, авторизации, резервного копирования и восстановления данных.

Для разработки системы выбор был сделан в пользу PostgreSQL.

2.4 Архитектура системы

Система будет разработана с помощью клиент-серверной архитектуры.

Клиент-серверная архитектура – это модель, которая используется для разработки программного обеспечения, где приложение разделяется на две части: клиентскую и серверную.[17]

Клиентская часть – это программа, которая выполняется на стороне пользователя и обеспечивает взаимодействие пользователя с приложением.

Серверная часть – это программа, которая выполняется на стороне сервера и обеспечивает обработку данных, которые были получены от клиентской части.

Основные преимущества клиент-серверной архитектуры:

- Распределение нагрузки между клиентом и сервером, что позволяет улучшить производительность и обеспечить более эффективное использование ресурсов;
- Централизованное управление данными, что обеспечивает более надежное хранение данных;
- Улучшенная безопасность, так как сервер может контролировать доступ пользователей к данным.[17]

Серверная часть будет разработана с помощью Nest.js.

Nest.js – это фреймворк для создания масштабируемых и эффективных серверных приложений на языке JavaScript. Он предоставляет инструменты для организации приложения по принципам модульности и обеспечения расширяемости.

В Nest.js клиент-серверная архитектура строится на основе обмена информацией между клиентом и сервером посредством HTTP-запросов. Клиент отправляет запросы на сервер, а сервер обрабатывает эти запросы и отправляет обратно ответы, которые клиент может использовать для отображения данных.

С помощью Nest.js можно создавать как монолитные, так и микросервисные архитектуры, обеспечивая легкую масштабируемость и поддержку большого количества запросов. Он также предоставляет инструменты для тестирования и отладки приложений, что делает его привлекательным выбором для разработки серверных приложений на JavaScript.

Клиентская часть системы будет разработана с помощью Vue.js.

Vue.js – это прогрессивный фреймворк JavaScript для создания пользовательских интерфейсов. Он позволяет создавать многокомпонентные приложения на основе компонентов, что облегчает поддержку и расширение приложений. Vue.js также предоставляет инструменты для управления состоянием приложения и маршрутизации, что делает его привлекательным выбором для разработки фронтенд-части веб-приложений.

Кроме того, Vue.js обладает легковесным размером и высокой производительностью, что позволяет создавать быстрые и отзывчивые пользовательские интерфейсы. Vue.js также имеет удобный синтаксис и хорошо документирован, что упрощает процесс изучения и использования фреймворка.

В целом, Vue.js является мощным инструментом для создания пользовательских интерфейсов и выбором многих разработчиков веб-приложений.

Для разрабатываемой системы была выбрана база данных PostgreSQL.

PostgreSQL – это реляционная система управления базами данных (СУБД), которая используется для хранения и управления большим объемом структурированных данных. Она поддерживает множество функций, включая транзакции, целостность данных, многопользовательский доступ и масштабируемость.

PostgreSQL является бесплатной и открытой СУБД, которая доступна для использования на многих операционных системах. Она также поддерживает множество языков программирования и позволяет использовать различные типы данных, такие как числа, строки, даты и времена, геометрические типы и т.д.

PostgreSQL также предоставляет инструменты для обеспечения безопасности и администрирования баз данных, включая механизмы аутентификации, авторизации, резервного копирования и восстановления данных.

В целом, PostgreSQL является мощным инструментом для хранения и управления данными, который может быть использован в различных приложениях и проектах.

2.5 База данных

Для разработки автоматизированного рабочего места была создана база данный, представленная на рисунке 4.

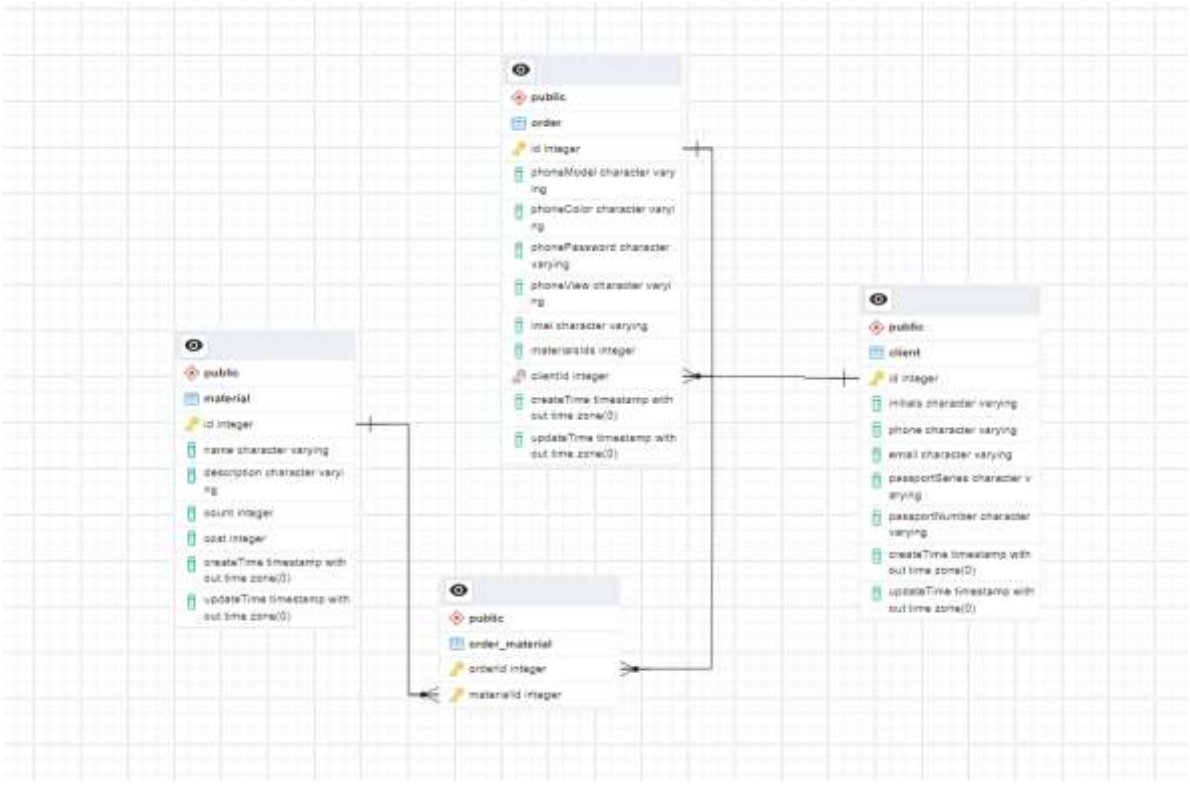

Рисунок 4 – База данных для АРМ

База данных состоит их четырёх таблиц: «Клиент», «Заказ», «Материал», «Заказ-материал».

Таблица «Заказ-материал» необходима для связи таблицы заказ и таблицы материал многие ко многим. Связь многие ко многим – это отношение между двумя таблицами, в котором каждая запись в одной таблице может быть связана с несколькими записями в другой таблице, и наоборот. Для реализации связи многие ко многим необходимо создать третью таблицу, которая будет содержать только ключи (идентификаторы) связанных записей из двух других таблиц. Эта третья таблица называется таблицей-связкой и используется для хранения информации о связях между записями в двух других таблицах, в разрабатываемой системе такой таблицей является таблица «Заказ-материал».

Таблица «Материал» содержит в себе следующие поля: название материала (текстовый тип данных), описание материала (текстовый тип данных), количество (числовой тип данных) и стоимость (числовой тип данных). Эта таблица необходима для описания покупаемых материалов и для контроля за количеством оставшихся материалов на складе.

Таблица «Заказ» содержит в себе следующие поля: модель телефона (текстовый тип данных), цвет телефона (текстовый тип данных), пароль телефона (текстовый тип данных), состояние телефона (текстовый тип данных) – в данном поле можно указать, например, что стекло разбито, идентификатор телефона (текстовый тип данных), материалы (числовой тип данных).

Таблица «Клиент» содержит в себе следующие поля: инициалы клиента (текстовый тип данных), телефон (текстовый тип данных), адрес электронной почты (текстовый тип данных), паспортные данные клиента (текстовый тип данных).

Таблица «Клиент» связана с таблицей «Заказ» связью один ко многим. Связь один ко многим – это отношение между двумя таблицами в базе данных, в котором одна запись в одной таблице может быть связана с несколькими записями в другой таблице. Таблица клиентов может быть связана с таблицей заказов, где

каждый клиент может иметь несколько заказов. Для реализации связи один ко многим, в таблицу заказов добавляется внешний ключ, который ссылается на первичный ключ таблицы клиентов.

3. Реализация

3.1 Backend

Backend разрабатываемого приложения был выполнен по чистой архитектуре.

Чистая архитектура (Clean Architecture) — это подход к проектированию серверной части приложения, который помогает создавать приложения, которые легко тестируются, поддерживаются и масштабируются.[18] В чистой архитектуре приложение разделяется на слои, каждый из которых имеет свою собственную задачу и зависит только от абстракций, а не от конкретных реализаций.

Для разрабатываемо приложения было создано два слоя:

Домен (Domain) – определяет ключевые сущности и бизнес-правила приложения. Этот слой не зависит от деталей реализации, таких как базы данных или пользовательский интерфейс.

Инфраструктура (Infrastructure) – содержит код, который реализует детали доступа к данным, такие как базы данных и внешние API.

Также были созданы следующие сущности: клиент, материал, заказ. Каждая сущность имеет модель и контроллер.

Модель – компонент, который представляет данные приложения и включает в себя структуру данных.

Контроллер – это компонент, который отвечает за обработку запросов к приложению. Контроллер получает запрос от клиента, обрабатывает его и возвращает ответ. Контроллер может использовать модели для получения или изменения данных, необходимых для обработки запроса.

Обычно модель и контроллер работают вместе, чтобы предоставить клиенту необходимые данные. Контроллер обрабатывает запросы и получает данные из модели, а затем возвращает эти данные клиенту в нужном формате. Это позволяет легко организовать работу с данными и обработку запросов в приложении.

На рисунке 5 и 6 показана модель сущности клиент.

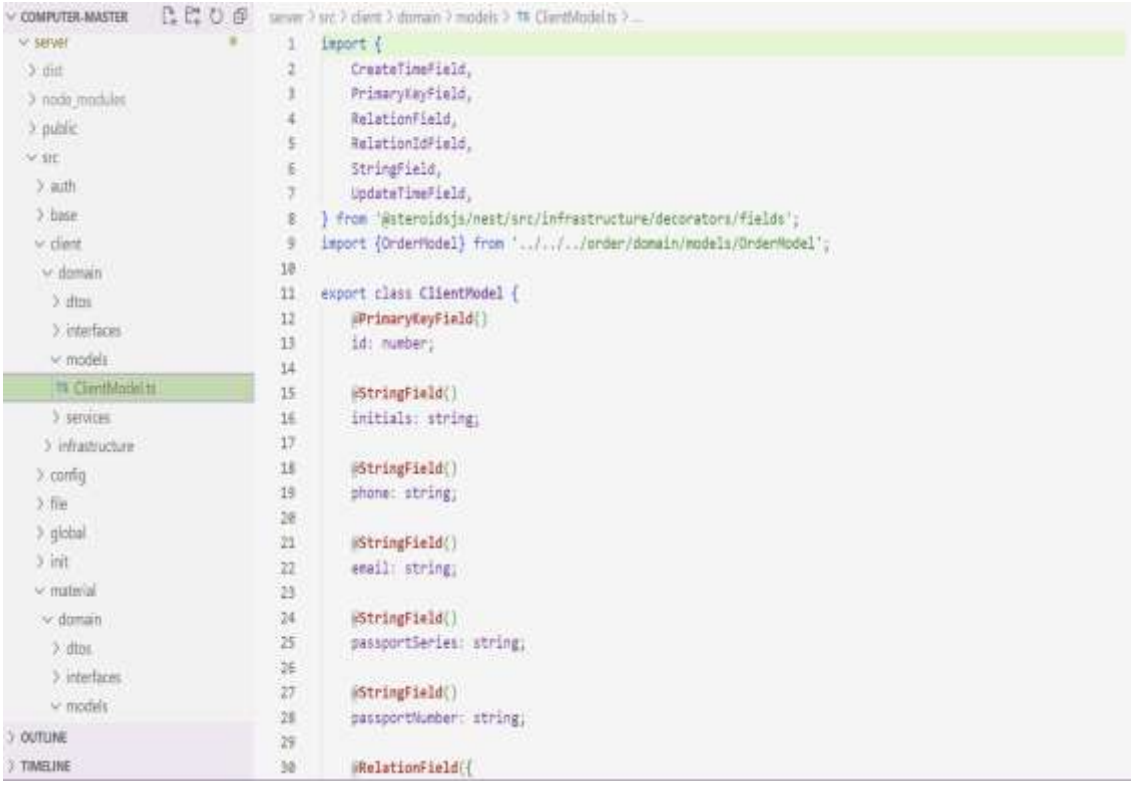

Рисунок 5 – Модель сущности клиент

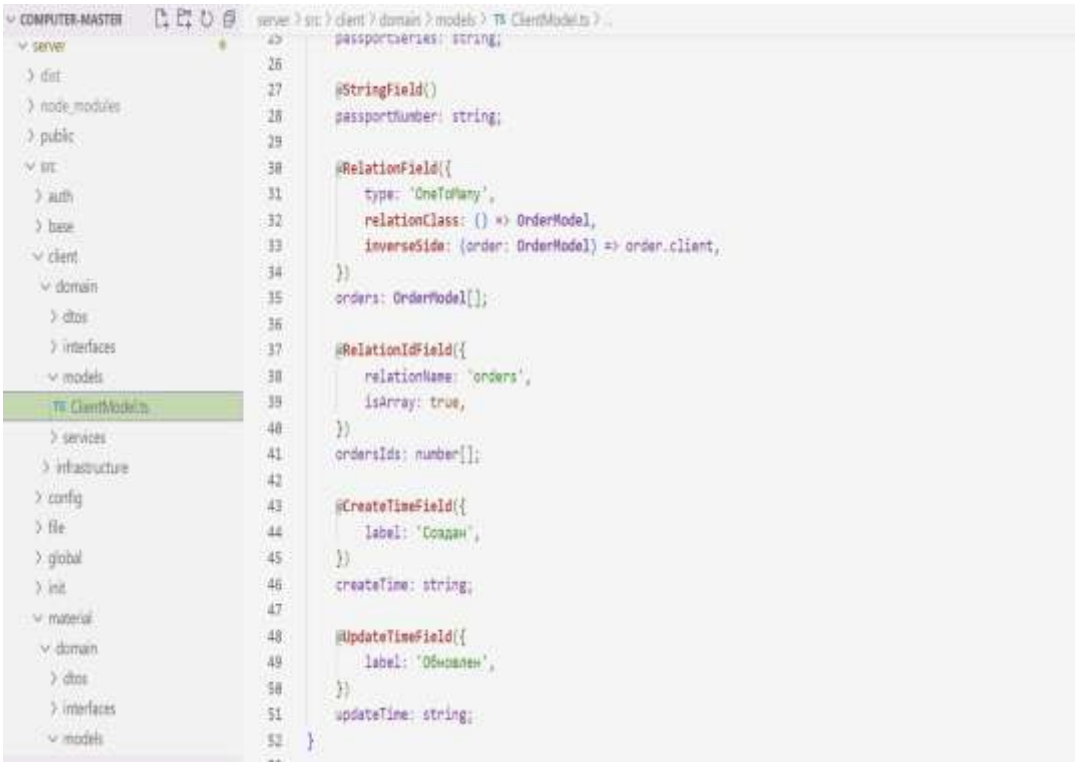

Рисунок 6 – Модель сущности клиент (продолжение)

На рисунках 7 и 8 показан контроллер сущности клиент.

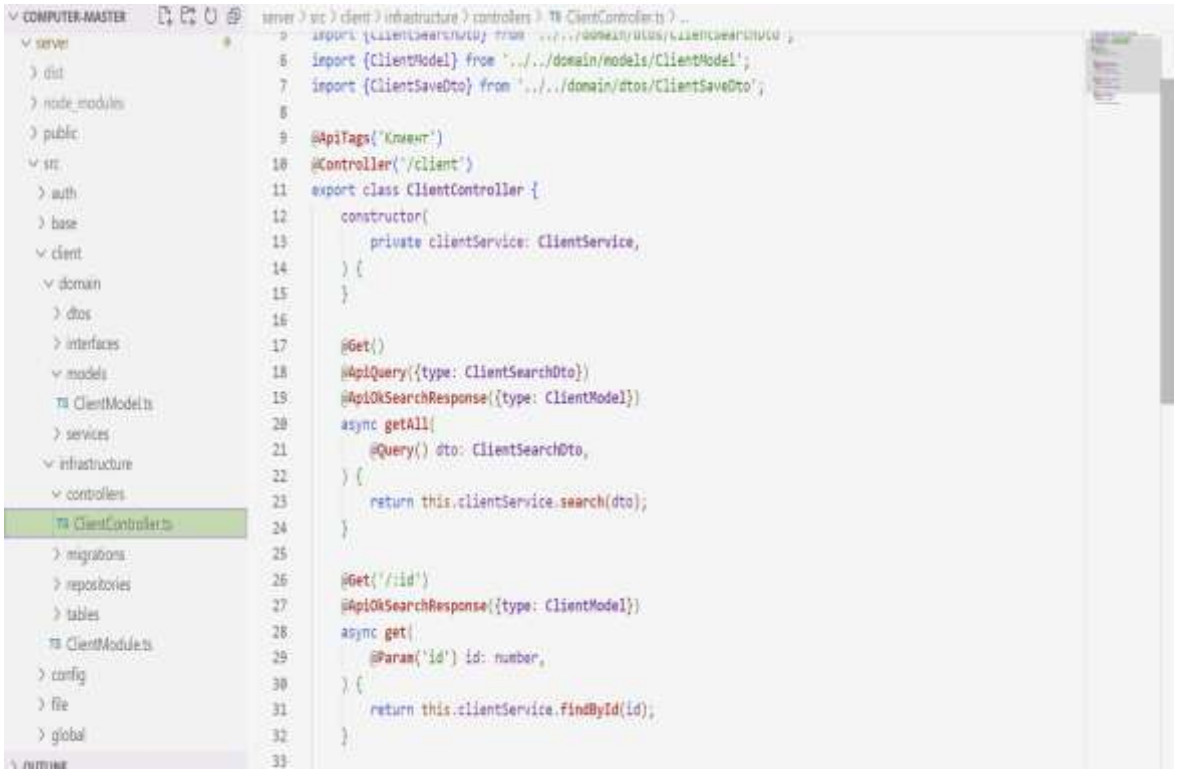

Рисунок 7 – Контроллер сущности клиент

Рисунок 8 – Контроллер сущности клиент (продолжение)

GET и POST — это два наиболее распространенных HTTP-метода, которые используются в контроллерах при написании серверной части приложения.

Метод GET используется для получения данных с сервера.[19] Когда клиент отправляет GET-запрос на сервер, он запрашивает определенную страницу или данные с определенного ресурса. GET-запросы могут содержать параметры

запроса, которые используются для передачи дополнительной информации на сервер.

Метод POST используется для отправки данных на сервер.[19] Когда клиент отправляет POST-запрос на сервер, он отправляет данные, которые будут обработаны на сервере. POST-запросы могут содержать тело запроса, которое содержит данные, которые будут отправлены на сервер.

В контроллерах, GET-запросы обычно используются для получения данных, таких как список пользователей или детали конкретного пользователя. POST-запросы обычно используются для отправки данных на сервер, таких как создание нового пользователя или обновление существующего пользователя.

Для сущности клиент метод Get() позволяет получить список всех клиентов. Метод Get(‘/:id) позволяет получить данные только одного клиента. Метод Post() позволяет создать нового клиента.

На рисунке 9 показана модель для сущности материал.

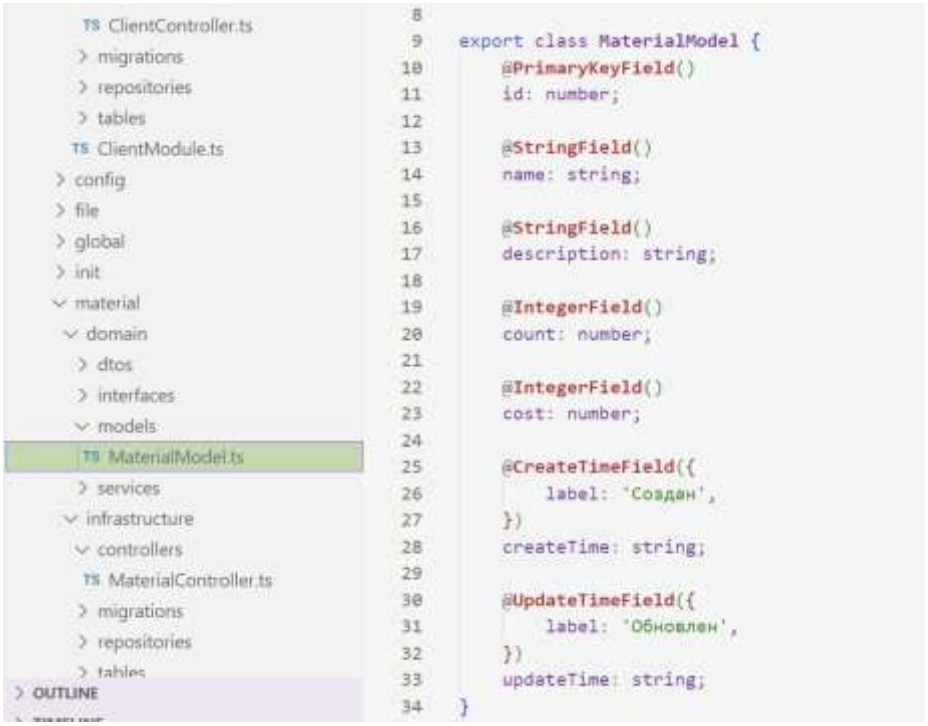

Рисунок 9 – Модель сущности материал

На рисунках 10 и 11 показан контроллер сущности материал.

Рисунок 10 – Контроллер сущности материал

Рисунок 11 – Контроллер сущности материал (продолжение)

Метод Get() позволяет получить список всех материалов. Метод Get(‘/:id’) позволяет получить один материал. Метод Post() позволяет создать новый материал. Метод Post(‘/:id’) позволяет редактировать выбранный материал.

На рисунках 12 и 13 показана модель сущности заказ.

Рисунок 12 – Модель сущности заказ

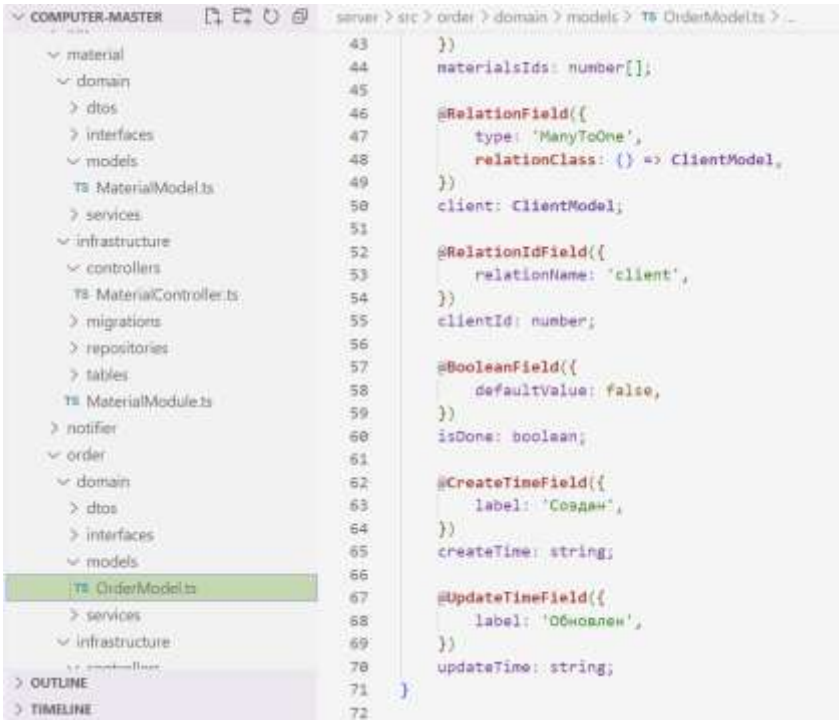

Рисунок 13 – Модель сущности заказ (продолжение)

На рисунке 14 и 15 показан контроллер сущности заказ.

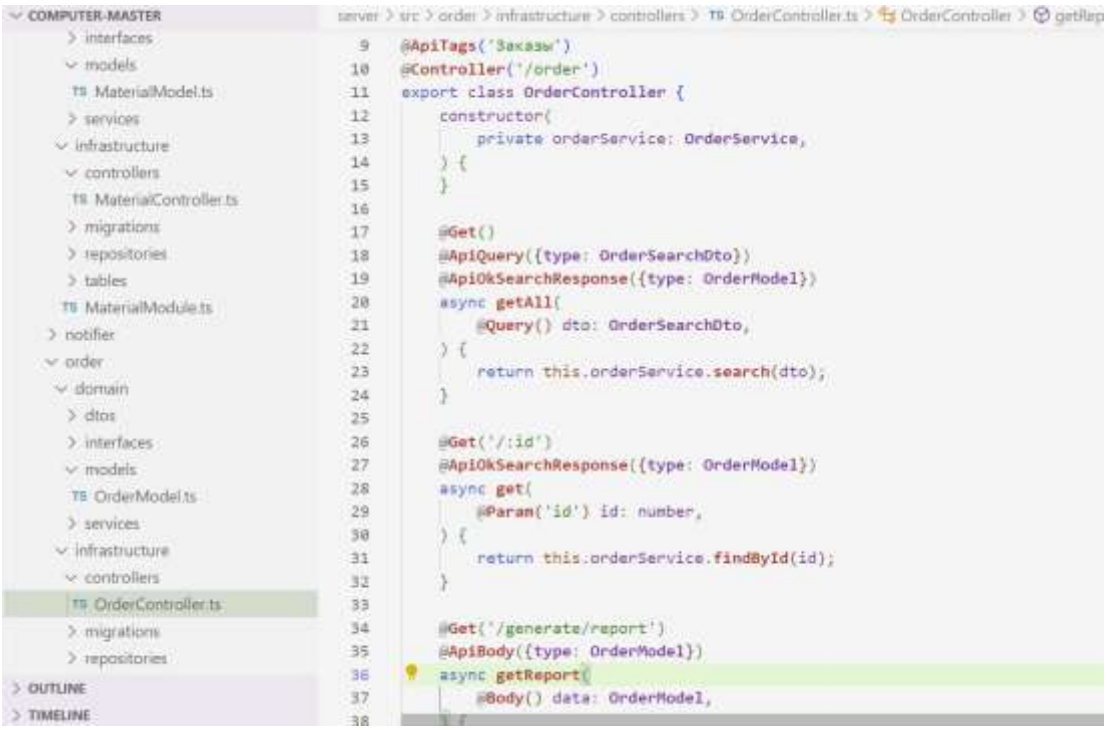

Рисунок 14 – Контроллер сущности заказ

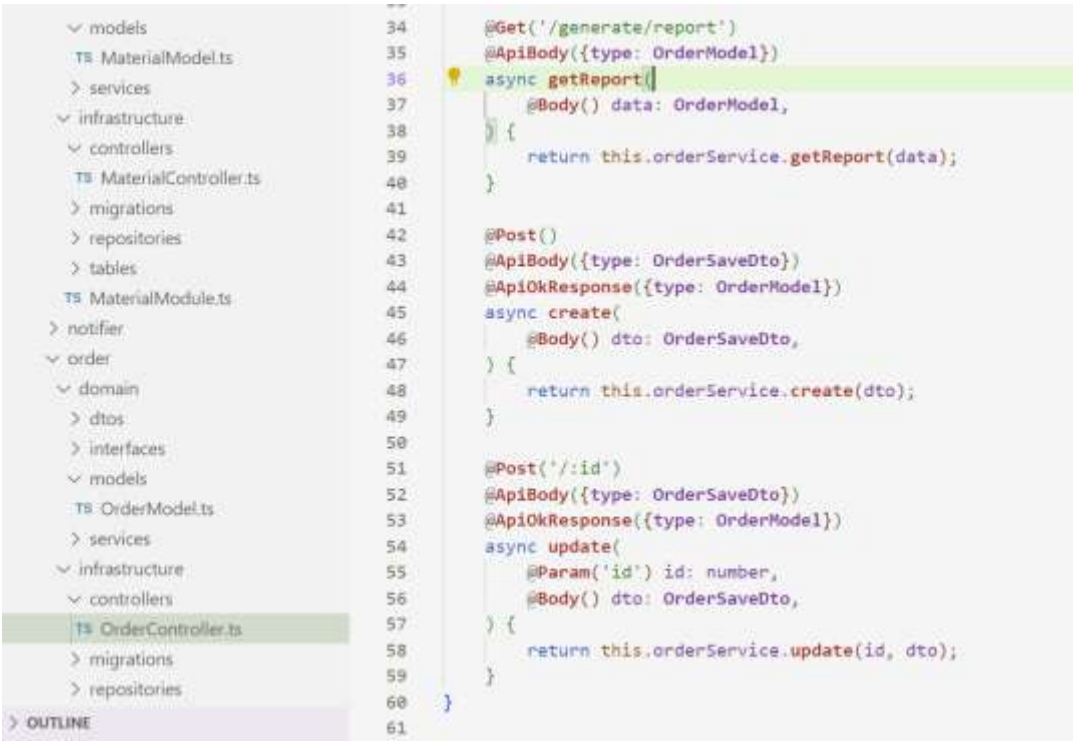

Рисунок 15 – Контроллер сущности заказ (продолжение)

Метод Get() позволяет получить список всех заказов. Метод Get(‘/:id’) позволяет получить один заказ. Метод Get(‘/generate/report’) позволяет создать

справку для клиента о том, что его заказ взять в работу. Метод Post() позволяет создать новый заказ. Метод Post(‘/:id’) позволяет редактировать выбранный заказ.

Три основных сущности приложения – Заказы, Материалы, Клиенты – запрограммированы в серверной части посредством методологии Dependency Injection, и объединены в контейнеры. Dependency Injection (DI) - это концепция программирования, которая позволяет управлять зависимостями между объектами в приложении.[20] Она используется для уменьшения связанности между компонентами приложения и улучшения тестируемости и переиспользуемости кода. Контейнеры DI предоставляют инструменты для управления зависимостями между объектами в приложении. Контейнер DI предоставляет объекты, которые зарегистрированы в контейнере, и автоматически управляет зависимостями между ними. Контейнер DI позволяет создавать объекты с необходимыми зависимостями без явного создания объектов зависимостей. Это позволяет улучшить масштабируемость и гибкость приложения. Контейнеры DI используются в различных языках программирования, включая Java, C# и PHP, и предоставляют мощный инструмент для управления зависимостями в приложении. Контейнеры DI могут быть сконфигурированы для работы с различными типами зависимостей, включая объекты, интерфейсы и классы. Контейнеры могут использоваться как для конфигурирования и управления зависимостями внутри приложения, так и для создания новых экземпляров классов. Контейнеры DI могут быть полезны при работе с большими и сложными приложениями, которые имеют множество зависимостей между компонентами. Они также могут быть использованы для облегчения разработки и тестирования приложений. Контейнеры DI предоставляют простой и эффективный способ управления зависимостями в приложении, что позволяет разработчикам сосредоточиться на решении бизнес-задач вместо того, чтобы тратить время на управление зависимостями между компонентами.[20] В целом, использование DI и контейнеров DI является важной практикой для создания гибких, масштабируемых и тестируемых приложений.

В Nest.js, DI является ключевой функциональностью и используется для создания гибких и масштабируемых приложений. Вместе с DI, Nest.js также предоставляет модули, которые являются фундаментальными блоками для организации кода в приложении.

Модули в Nest.js являются классами, которые определяют, какие зависимости должны быть созданы и как они должны быть сконфигурированы. Модули могут содержать провайдеры, которые предоставляют экземпляры классов, сервисы, которые обрабатывают бизнес-логику, и контроллеры, которые обрабатывают HTTP-запросы. Модули также могут иметь зависимости от других модулей, что позволяет организовывать приложение в виде древовидной структуры зависимостей.

DI в Nest.js позволяет использовать модули для управления зависимостями между компонентами приложения. Nest.js предоставляет инжектор, который автоматически создает и управляет экземплярами классов, определенных в модулях. Это позволяет разработчикам создавать модули с необходимыми зависимостями и легко использовать их в других модулях и компонентах приложения.

В целом, использование DI и модулей в Nest.js позволяет создавать гибкие, масштабируемые и легко тестируемые приложения. DI и модули позволяют организовывать код в приложении и управлять зависимостями между компонентами, что упрощает разработку и обслуживание приложений.

Главный DI модуль приложения показан на рисунках 16 и 17.

```
@Module({
    imports: [
        ConfigModule.forRoot({
            load: config,
        }),
        TypeOrmModule.forRootAsync({
            imports: [ConfigModule],
            inject: [ConfigService],
            useFactory: (configService: ConfigService) => (configService.get('database') as PostgresConnectionOptions),
        } as TypeOrmModuleAsyncOptions),
        process.env.APP_ENVIRONMENT === 'dev' && ServeStaticModule.forRoot({
            rootPath: process.env.APP_ROOT_FILES_DIR,
            serveRoot: process.env.APP_STATIC_URL_PREFIX,
            exclude: ['/api*'],
        }),
        process.env.APP_SENTRY_DSN && SentryModule.forRoot({
            dsn: process.env.APP_SENTRY_DSN,
            environment: process.env.APP_ENVIRONMENT,
        }),
```

Рисунок 16 – Главный DI модуль приложения

```
            ScheduleModule.forRoot(),
            GlobalModule,
            InitModule,
            CommandModule,
            FileModule,
            AuthModule,
            UserModule,
            NotifierModule,
            MaterialModule,
            OrderModule,
            ClientModule,
        ].filter(Boolean),
        providers: [
            MigrateCommand,
            {
                provide: APP_FILTER,
                useClass: ValidationExceptionFilterCustom,
            },
            {
                provide: APP_FILTER,
                useClass: RequestExecutionExceptionFilter,
            },
        ],
    })
    export class AppModule {
    }
```

Рисунок 17 – Главный DI модуль приложения (продолжение)

На рисунке 18 показан метод, с помощью которого реализовано создание справки.

```
@Get('/:id/report')
@ApiBody({type: OrderModel})
async getReport(
    @Param('id') id: number,
): Promise<string> {
    const file = await this.orderService.getReport(id);
    return path.join(path.resolve(__dirname, '../../../../../'), file.url);
}
```

Рисунок 18 – Метод создания отчёта

Для того, чтобы получить справку необходимо отправить Get() запрос, который вызовет метод getReport().

Метод getReport() показан на рисунке 19.

```
async getReport(id: number) {
    const template = new OrderTemplate();

    const order = await this.createQuery().with('client').where({id}).one();

    const content = await template.generate(order);

    const fileName = `Заказ-${id}.pdf`;

    return this.saveFile(fileName, content);
}
```

Рисунок 19 – Метод getReport()

Метод getReport() создает шаблон справки. После этого ищет по идентификатору необходимый заказ и генерирует справку. После генерации вызывает метод для сохранения справки. Этот метод показан на рисунке 20.

```
private async saveFile(fileName: string, content: Buffer): Promise<FileModel> {
    return this.fileService.upload(
        DataMapper.create<FileStreamSourceDto>(FileStreamSourceDto, {
            fileName,
            fileMimeType: 'application/pdf',
            stream: content,
        }),
    );
}
```

Рисунок 20 – Метод saveFiled()

После сохранения пользователю возвращается путь к справке.

3.2 Frontend

Для разработки клиентской части приложения было решено не использовать Redux, так как при работе с этой библиотекой приложение получается громоздким из-за бойлерплейта, дополнительное время уходит на ручной ввод похожего по функционалу кода. Также приходится вручную добавлять типизацию и информацию о запросах. Кроме этого, при ручном вводе всегда высока вероятность появления ошибок из-за человеческого фактора.

Проанализировав это и изучив другие способы разработки было принято решение использовать библиотеку React Query, которая применяет хуки для работы с API. А также эта библиотека отличается гибкостью в настройке и большому функционалу по сравнению с useSWR, RTK Query, наличием собственных удобных Devtools, удобством работы со списками, пагинацией.[21]

React Query имеет некоторые особенности.

Вместо использования компонентов высшего порядка (HOC), библиотека предлагает использовать хуки, что делает код более читаемым и понятным.

Еще одной особенностью React Query является то, что данные по умолчанию не хранятся в локальном хранилище (localStorage, sessionStorage), а остаются только в рантайме. Однако, при необходимости, их можно удобным образом персистить в любом клиентском хранилище.

Для более эффективного управления состоянием приложения, рекомендуется использовать подход с «умными» и «глупыми» компонентами для разделения ответственности между компонентами.

С помощью React Query можно создавать отдельный хук для каждого запроса. При этом, можно указать ключ, по которому должен храниться ответ запроса, и задать конфигурацию. Благодаря этому, управление состоянием приложения становится более гибким и удобным в использовании.[21]

На рисунке 21 показан файл routes.ts. В этом файле определяются маршруты, которые пользователь может запрашивать, и какой компонент должен быть отображен для каждого маршрута.

Рисунок 21 – Маршруты приложения

Было создано три маршрута – заказы, клиенты, материалы. Для каждого маршрута указан соответствующий компонент, который будет отображаться при запросе этого маршрута. Когда пользователь переходит по маршруту, компонент, указанный для этого маршрута, будет отображен на странице.

На рисунке 22 показан файл server.ts. Этот файл отвечает за настройку обращений к серверу по API.

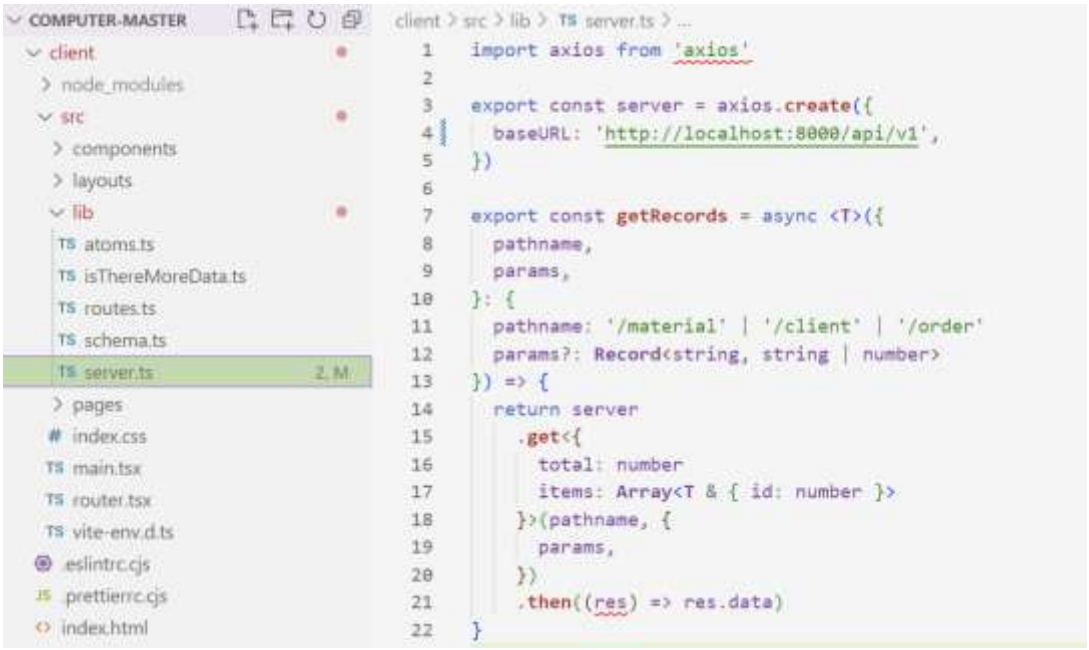

Рисунок 22 – Файл server.ts

На рисунке 23 показана форма для добавления материалов.

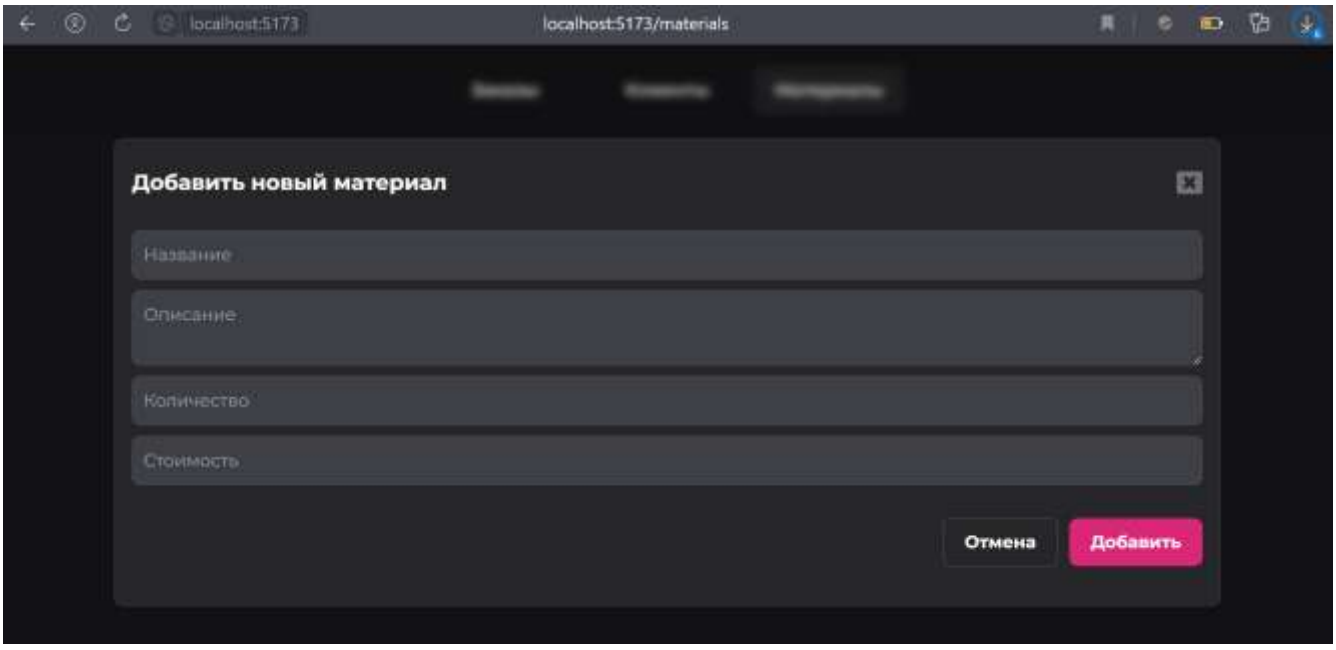

Рисунок 23 – Форма для добавления материалов

На рисунке 24 показана страница с добавленными материалами.

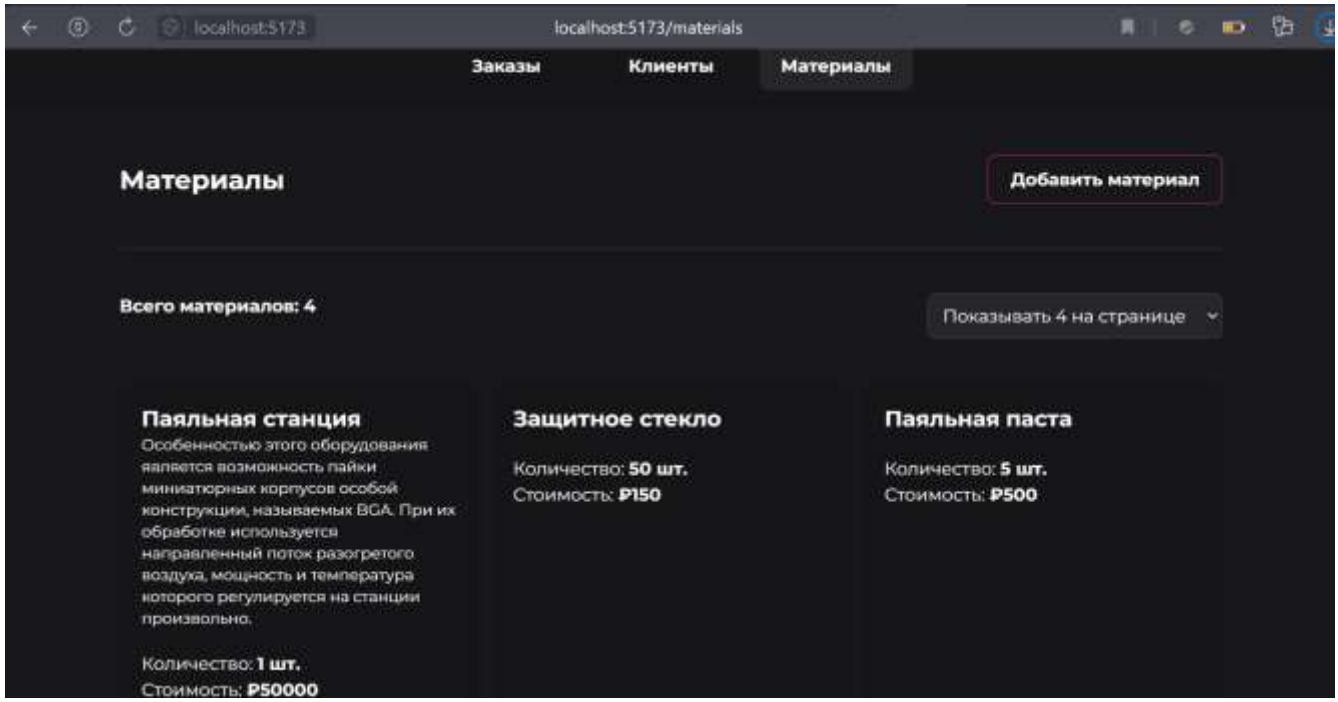

Рисунок 24 – Страница с материалами

На рисунке 25 показана форма для добавления нового клиента.

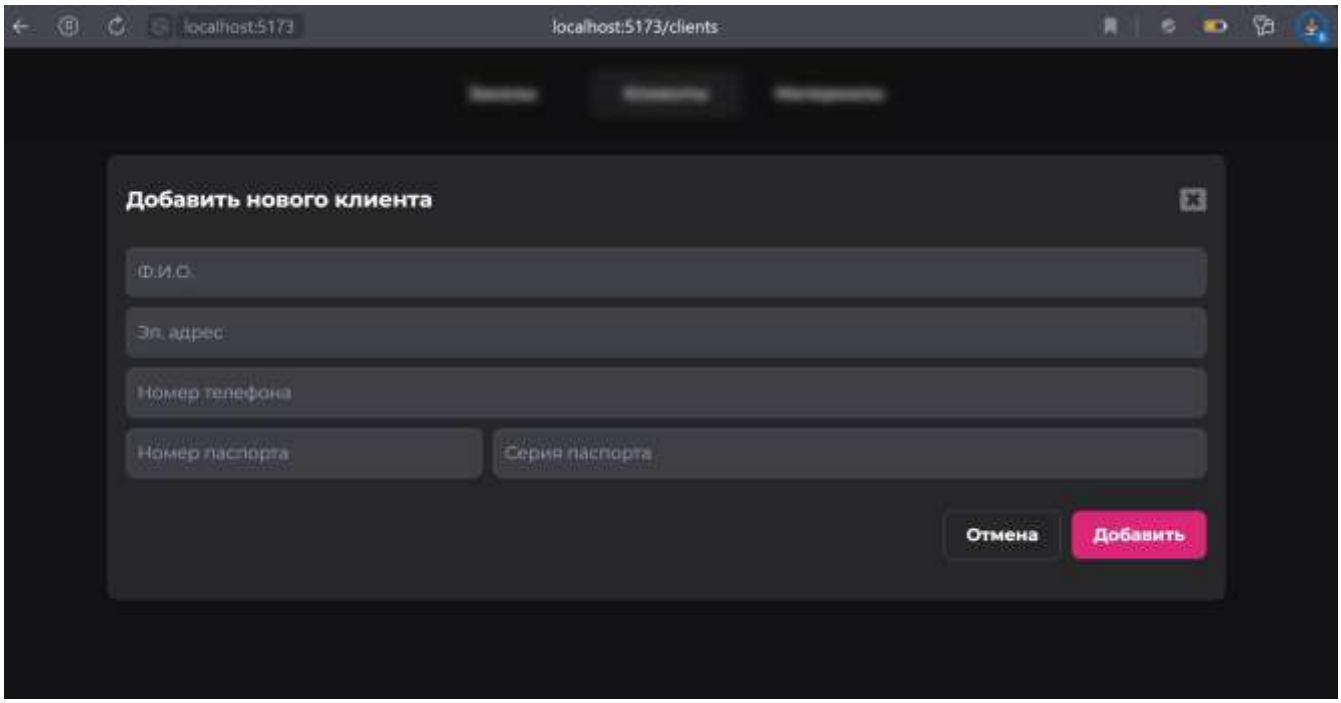

Рисунок 25 – Форма для добавления нового клиента

В форме добавлена валидация полей. Это показано на рисунке 26.

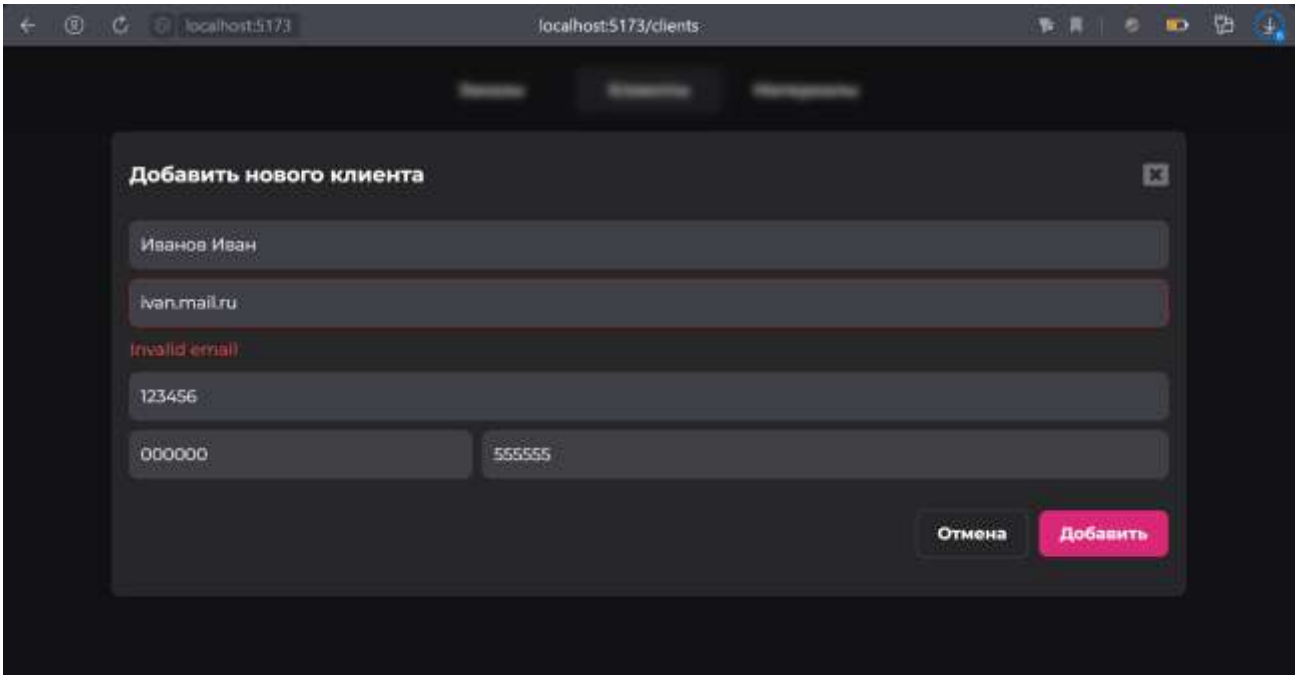

Рисунок 26 – Ошибка валидации

На рисунке 27 показана страница с существующими клиентами.

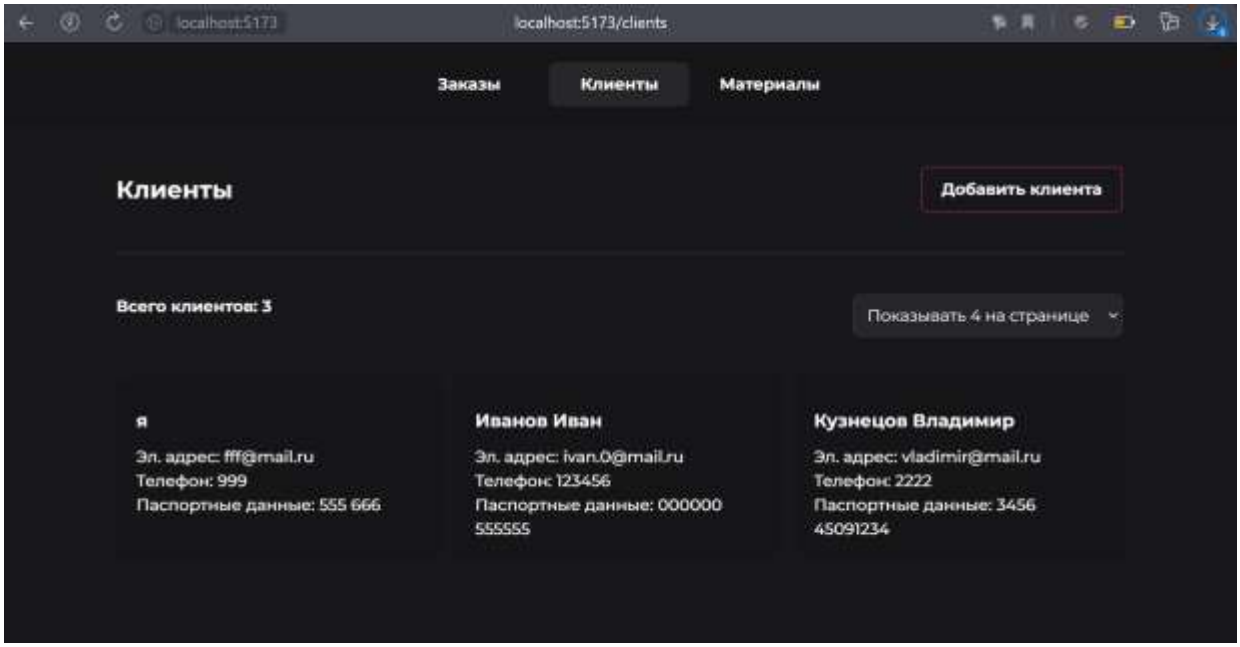

Рисунок 27 – Страница с клиентами

На рисунке 28 показана форма для создания заказа.

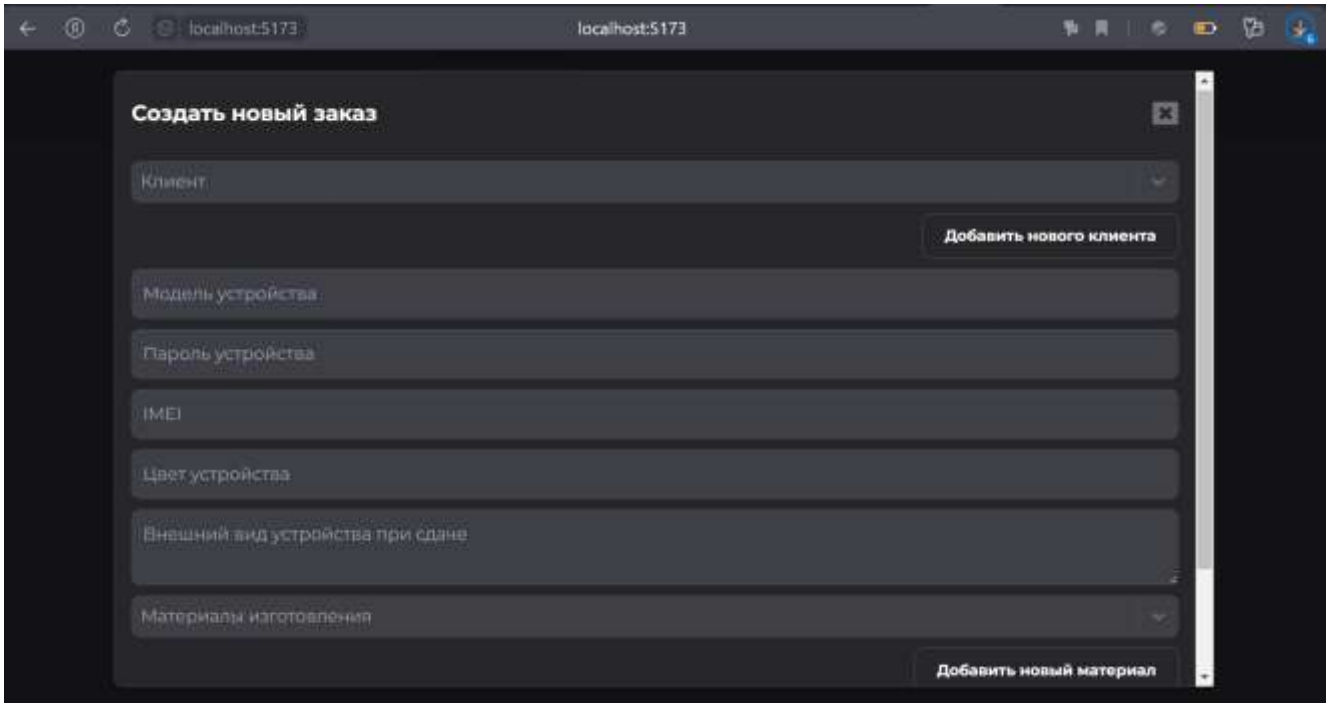

Рисунок 28 – Форма для создания заказа

Существующие заказы показаны на рисунке 29.

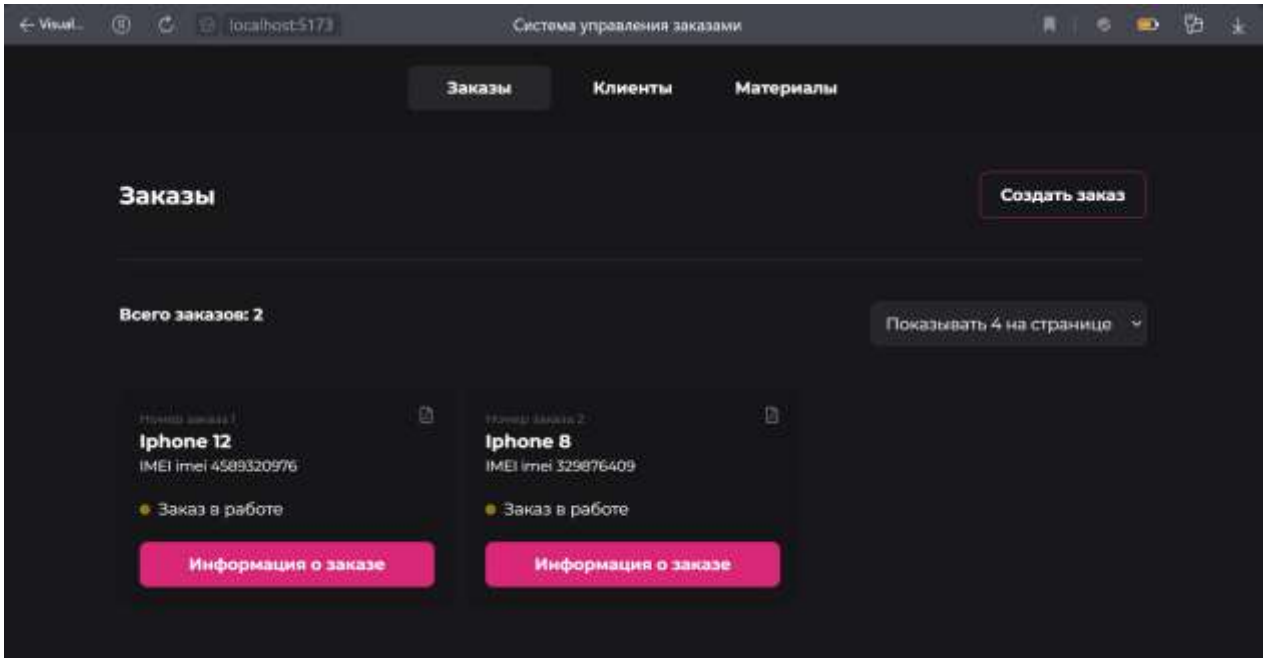

Рисунок 29 – Заказы

При нажатии на кнопку «Подробнее» откроется окно, показанное на рисунке 30.

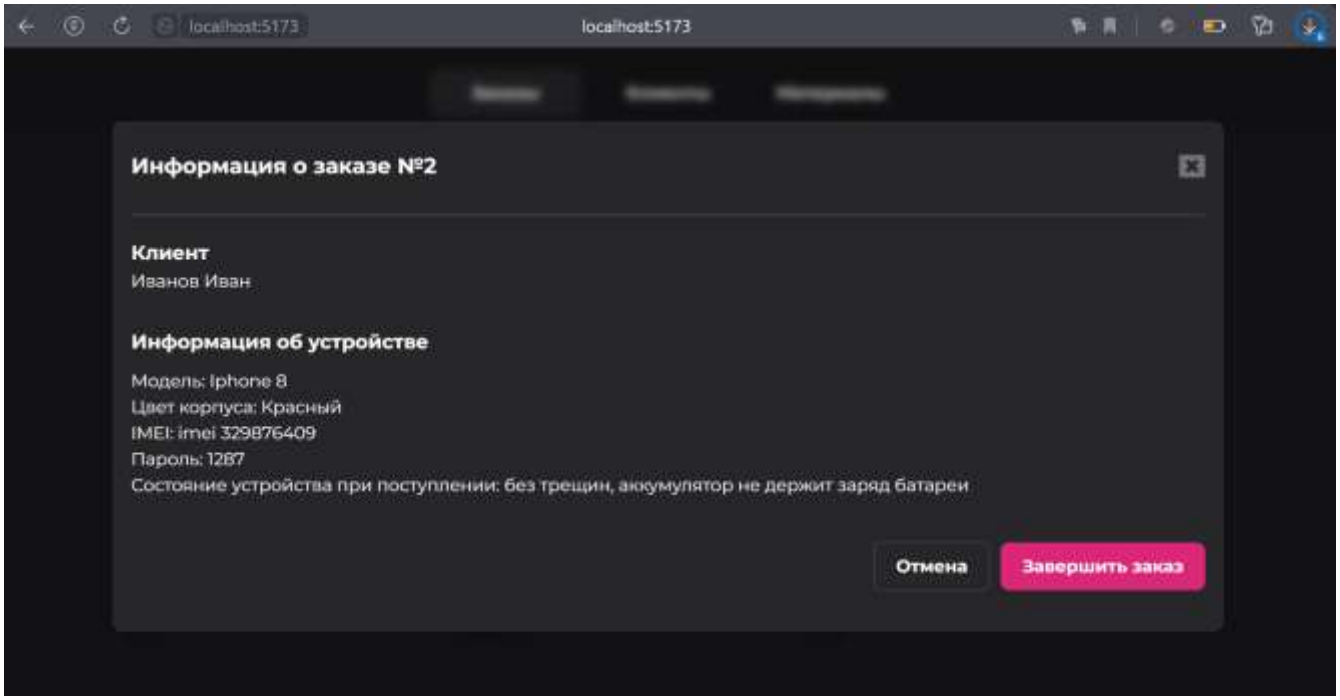

Рисунок 30 – Подробная информация о заказе

На рисунке 31 показано окно, которое открывается при нажатии на кнопку генерации справки.

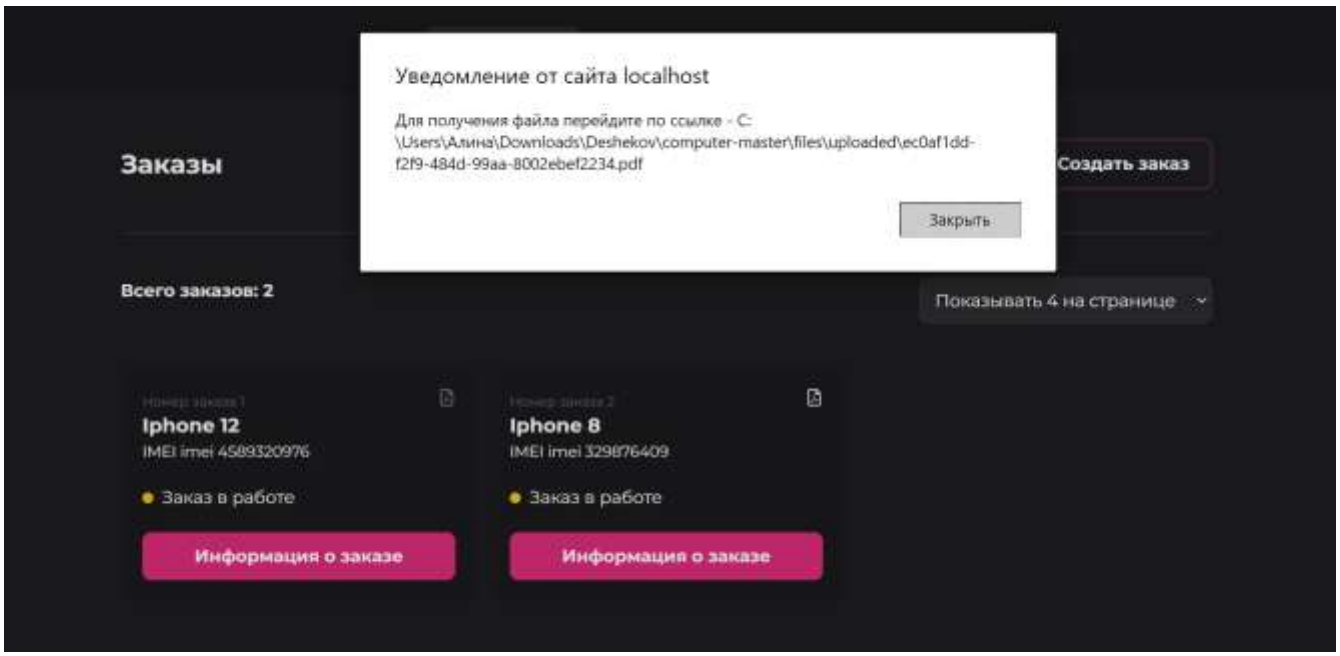

Рисунок 31 – Генерация справки

На рисунке 32 показана сгенерированная справка.

ЛИСТ ЗАКАЗА #2

ФИО: Иванов Иван

Модель телефона: Iphone 8

Цвет телефона: Красный

Внешний вид телефона: Без трещин, аккумулятор не держит заряд батареи

ИМЕИ: imei 329876409

Дата создания: 2023-06-01 17:47:42

_____ (подпись, ФИО)

Рисунок 32 – Справка

ЗАКЛЮЧЕНИЕ

Для выполнения выпускной квалификационной работы по теме «Автоматизированное рабочее место мастера по ремонту компьютерной техники» была поставлена цель – разработать АРМ. Для достижения этой цели были поставлены следующие задачи: обзор предметной области; выбор средств разработки; разработка базы данных; разработка серверной части системы; разработка клиентской части системы.

Для реализации первой задачи было рассмотрено три автоматизированных рабочих места, выявлены преимущества и недостатки. На основе этого были сформулированы требования для разработанного приложения.

Для реализации второй задачи были рассмотрены различные средства разработки. Были изучены преимущества и недостатки каждого варианта. Исходя из этого были определены наиболее подходящие средства разработки для создания автоматизированного рабочего места.

После рассмотрения аналогов и средств разработки была начата реализация приложения. Была создана база данных, разработана серверная и клиентская части системы на основе выявленных требований.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. CRM для сервисных центров / A2IS : сайт. – URL: https://a2is.ru/catalog/programmy-dlya-kompanij-po-remontu-kompyuterov (дата обращения: 20.02.2023).
2. Автоматизированное рабочее место / STUDFILE : сайт. – URL: https://studfile.net/preview/7170105/ (дата обращения: 20.02.2023).
3. Информационные технологии конечного пользователя / INTUIT : сайт. – URL: https://intuit.ru/studies/courses/3609/851/lecture/31652 (дата обращения: 20.02.2023).
4. Программа учёта, автоматизации и управления магазином, сервисным центром, складом / GINCORE : сайт. – URL: https://gincore.net/ (дата обращения: 29.02.2023).
5. CRM система для малого бизнеса / LIVESKLAD : сайт. – URL: https://livesklad.com/ (дата обращения: 05.03.2023).
6. HelloClient – программа для сервисных центров / HELLOCLIENT : сайт. – URL: https://helloclient.ru/ (дата обращения: 05.03.2023).
7. IDF0 Знакомство с нотацией и пример использования / TRINION : сайт. – URL: https://trinion.org/blog/idef0-znakomstvo-s-notaciey-i-primer-ispolzovaniya (дата обращения: 10.03.2023).
8. Использование диаграммы вариантов использования UML при проектировании программного обеспечения / HABR : сайт. – URL: https://habr.com/ru/articles/566218/ (дата обращения: 10.03.2023).
9. PHP: Hypertext Preprocessor / PHP : сайт. – URL: https://www.php.net/ (дата обращения: 20.03.2023).
10. Документация | NestJS – прогрессивный фреймворк Node.js / NESTJS : сайт. – URL: https://nestjs.ru/ (дата обращения: 20.03.2023).
11. React или Angular или Vue.js — что выбрать? / HABR : сайт. – URL: https://habr.com/ru/post/476312/ (дата обращения: 25.03.2023).

12. Vue.js Прогрессивный JavaScript-фреймворк / VUEJS : сайт. – URL: https://ru.vuejs.org/ (дата обращения: 01.04.2023).
13. React / REACT : сайт. – URL: https://react.dev/ (дата обращения: 01.04.2023).
14. MySQL и MongoDB — когда и что лучше использовать / HABR : сайт. – URL: https://habr.com/ru/post/322532/ (дата обращения: 05.04.2023).
15. MYSQL: что это за сервер базы данных, пример / SKILLFACTORY : сайт. – URL: https://blog.skillfactory.ru/glossary/mysql/ (дата обращения: 05.04.2023).
16. PostgreSQL: самая совершенная в мире реляционная база данных с открытым исходным кодом / POSTGRESQL : сайт. – URL: https://www.postgresql.org/ (дата обращения: 05.04.2023).
17. Клиент-серверная архитектура в картинках / HABR : сайт. – URL: https://habr.com/ru/articles/495698/ (дата обращения: 10.04.2023).
18. Чистая архитектура / HABR : сайт. – URL: https://habr.com/ru/articles/269589/ (дата обращения: 10.04.2023).
19. Чем отличаются http-методы get и post / HTMLACADEMY : сайт. – URL: https://htmlacademy.ru/blog/php/get-vs-post (дата обращения: 20.04.2023).
20. Dependency injection / HABR : сайт. – URL: https://habr.com/ru/articles/350068/ (дата обращения: 23.04.2023).
21. Руководство по React Query / MY-JS : сайт. – URL: https://my-js.org/docs/guide/react-query/ (дата обращения: 25.04.2023).

ПРИЛОЖЕНИЕ А

Отчет «Антиплагиат»

Отчет о проверке на заимствования №1

Автор: Дешеков Алексей Валерьевич
Проверяющий: Захаров Павел Алексеевич
Организация: Сибирский федеральный университет

Отчет предоставлен сервисом «Антиплагиат» - http://sfukras.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 270542
Начало загрузки: 18.06.2023 17:06:33
Длительность загрузки: 00:00:11
Имя исходного файла: vkr_deshekov.docx
Название документа: Выпускная квалификационная работа
Размер текста: 50 кБ
Тип документа: Выпускная квалификационная работа
Символов в тексте: 50954
Слов в тексте: 5851
Число предложений: 430

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 18.06.2023 17:06:45
Длительность проверки: 00:01:51
Комментарии: не указано
Поиск с учетом редактирования: да
Проверенные разделы: основная часть с. 1,3-43, содержание с. 2, библиография с. 44-45, приложение с. 46-58
Модули поиска: ИПС Адилет, Библиография, Сводная коллекция ЭБС, Интернет Плюс*, Сводная коллекция РГБ, Цитирование, Переводные заимствования (RuEn), Переводные заимствования по eLIBRARY.RU (EnRu), Переводные заимствования по коллекции Гарант: аналитика, Переводные заимствования по коллекции Интернет в английском сегменте, Переводные заимствования по Интернету (EnRu), Переводные заимствования по коллекции Интернет в русском сегменте, Переводные заимствования издательства Wiley , eLIBRARY.RU, СПС ГАРАНТ: аналитика, СПС ГАРАНТ: нормативно-правовая документация, IEEE, Медицина, Диссертации НББ, Коллекция НБУ, Перефразирования по eLIBRARY.RU, Перефразирования по СПС ГАРАНТ: аналитика, Перефразирования по Интернету, Перефразирования по Интернету (EN), Перефразированные заимствования по коллекции Интернет в английском сегменте, Перефразированные заимствования по коллекции Интернет в русском сегменте, Перефразирования по коллекции издательства Wiley , Патенты СССР, РФ, СНГ, СМИ России и СНГ, Модуль поиска "sfukras", Шаблонные фразы, Кольцо вузов, Издательство Wiley, Переводные заимствования

СОВПАДЕНИЯ 16,59%
САМОЦИТИРОВАНИЯ 0%
ЦИТИРОВАНИЯ 5,69%
ОРИГИНАЛЬНОСТЬ 77,72%

Рисунок А.1 – Отчет о проверке на заимствования

ПРИЛОЖЕНИЕ Б

Код модулей и контроллеров сущностей

Код файла ClientModel.ts:

```
import {
    CreateTimeField,
    PrimaryKeyField,
    RelationField,
    RelationIdField,
    StringField,
    UpdateTimeField,
} from '@steroidsjs/nest/src/infrastructure/decorators/fields';
import {OrderModel} from '../../../order/domain/models/OrderModel';

export class ClientModel {
    @PrimaryKeyField()
    id: number;

    @StringField()
    initials: string;

    @StringField()
    phone: string;

    @StringField()
    email: string;

    @StringField()
    passportSeries: string;

    @StringField()
    passportNumber: string;

    @RelationField({
        type: 'OneToMany',
        relationClass: () => OrderModel,
        inverseSide: (order: OrderModel) => order.client,
    })
    orders: OrderModel[];

    @RelationIdField({
        relationName: 'orders',
        isArray: true,
    })
    ordersIds: number[];

    @CreateTimeField({
        label: 'Создан',
```

```
    })
    createTime: string;

    @UpdateTimeField({
        label: 'Обновлен',
    })
    updateTime: string;
}
```

Код файла ClientController.ts:

```
import {Body, Controller, Get, Param, Post, Query} from '@nestjs/common';
import {ApiBody, ApiOkResponse, ApiQuery, ApiTags} from '@nestjs/swagger';
import {ApiOkSearchResponse} from
'@steroidsjs/nest/src/infrastructure/decorators/ApiOkSearchResponse';
import {ClientService} from '../../domain/services/ClientService';
import {ClientSearchDto} from '../../domain/dtos/ClientSearchDto';
import {ClientModel} from '../../domain/models/ClientModel';
import {ClientSaveDto} from '../../domain/dtos/ClientSaveDto';

@ApiTags('Клиент')
@Controller('/client')
export class ClientController {
    constructor(
        private clientService: ClientService,
    ) {
    }

    @Get()
    @ApiQuery({type: ClientSearchDto})
    @ApiOkSearchResponse({type: ClientModel})
    async getAll(
        @Query() dto: ClientSearchDto,
    ) {
        return this.clientService.search(dto);
    }

    @Get('/:id')
    @ApiOkSearchResponse({type: ClientModel})
    async get(
        @Param('id') id: number,
    ) {
        return this.clientService.findById(id);
    }

    @Post()
    @ApiBody({type: ClientSaveDto})
    @ApiOkResponse({type: ClientModel})
    async create(
```

```
        @Body() dto: ClientSaveDto,
    ) {
        return this.clientService.create(dto);
    }
}
```

Код файла MaterialModel.ts:

```
import {
    CreateTimeField,
    IntegerField,
    PrimaryKeyField,
    StringField,
    UpdateTimeField,
} from '@steroidsjs/nest/src/infrastructure/decorators/fields';

export class MaterialModel {
    @PrimaryKeyField()
    id: number;

    @StringField()
    name: string;

    @StringField()
    description: string;

    @IntegerField()
    count: number;

    @IntegerField()
    cost: number;

    @CreateTimeField({
        label: 'Создан',
    })
    createTime: string;

    @UpdateTimeField({
        label: 'Обновлен',
    })
    updateTime: string;
}
```

Код файла MaterialController.ts:

```
import {Body, Controller, Get, Param, Post, Query} from '@nestjs/common';
import {ApiBody, ApiOkResponse, ApiQuery, ApiTags} from '@nestjs/swagger';
```

```
import {ApiOkSearchResponse} from
'@steroidsjs/nest/src/infrastructure/decorators/ApiOkSearchResponse';
import {MaterialService} from '../../domain/services/MaterialService';
import {MaterialSearchDto} from '../../domain/dtos/MaterialSearchDto';
import {MaterialModel} from '../../domain/models/MaterialModel';
import {MaterialSaveDto} from '../../domain/dtos/MaterialSaveDto';

@ApiTags('Материалы')
@Controller('/material')
export class MaterialController {
    constructor(
        private materialService: MaterialService,
    ) {
    }

    @Get()
    @ApiQuery({type: MaterialSearchDto})
    @ApiOkSearchResponse({type: MaterialModel})
    async getAll(
        @Query() dto: MaterialSearchDto,
    ) {
        return this.materialService.search(dto);
    }

    @Get('/:id')
    @ApiOkSearchResponse({type: MaterialModel})
    async get(
        @Param('id') id: number,
    ) {
        return this.materialService.findById(id);
    }

    @Post()
    @ApiBody({type: MaterialSaveDto})
    @ApiOkResponse({type: MaterialModel})
    async create(
        @Body() dto: MaterialSaveDto,
    ) {
        return this.materialService.create(dto);
    }

    @Post('/:id')
    @ApiBody({type: MaterialSaveDto})
    @ApiOkResponse({type: MaterialModel})
    async update(
        @Param('id') id: number,
        @Body() dto: MaterialSaveDto,
    ) {
        return this.materialService.update(id, dto);
    }
}
```

Код файла OrderModel.ts:

```
import {
    BooleanField,
    CreateTimeField,
    PrimaryKeyField,
    RelationField,
    RelationIdField,
    StringField,
    UpdateTimeField,
} from '@steroidsjs/nest/src/infrastructure/decorators/fields';
import {ClientModel} from '../../../client/domain/models/ClientModel';
import {MaterialModel} from '../../../material/domain/models/MaterialModel';

export class OrderModel {
    @PrimaryKeyField()
    id: number;

    @StringField()
    phoneModel: string;

    @StringField()
    phoneColor: string;

    @StringField()
    phonePassword: string;

    @StringField()
    phoneView: string;

    @StringField()
    imei: string;

    @RelationField({
        type: 'ManyToMany',
        relationClass: () => MaterialModel,
        isOwningSide: true,
        inverseSide: () => MaterialModel,
    })
    materials: MaterialModel[];

    @RelationIdField({
        relationName: 'materials',
        isArray: true,
    })
    materialsIds: number[];

    @RelationField({
```

```
        type: 'ManyToOne',
        relationClass: () => ClientModel,
    })
    client: ClientModel;

    @RelationIdField({
        relationName: 'client',
    })
    clientId: number;

    @BooleanField({
        defaultValue: false,
    })
    isDone: boolean;

    @CreateTimeField({
        label: 'Создан',
    })
    createTime: string;

    @UpdateTimeField({
        label: 'Обновлен',
    })
    updateTime: string;
}
```

Код файла OrderController.ts:

```
import {Body, Controller, Get, Param, Post, Query} from '@nestjs/common';
import {ApiBody, ApiOkResponse, ApiQuery, ApiTags} from '@nestjs/swagger';
import {ApiOkSearchResponse} from
'@steroidsjs/nest/src/infrastructure/decorators/ApiOkSearchResponse';
import {OrderService} from '../../domain/services/OrderService';
import {OrderSearchDto} from '../../domain/dtos/OrderSearchDto';
import {OrderModel} from '../../domain/models/OrderModel';
import {OrderSaveDto} from '../../domain/dtos/OrderSaveDto';
import * as path from 'path';

@ApiTags('Заказы')
@Controller('/order')
export class OrderController {
    constructor(
        private orderService: OrderService,
    ) {
    }

    @Get()
    @ApiQuery({type: OrderSearchDto})
    @ApiOkSearchResponse({type: OrderModel})
```

```
    async getAll(
        @Query() dto: OrderSearchDto,
    ) {
        return this.orderService.search(dto);
    }

    @Get('/:id')
    @ApiOkSearchResponse({type: OrderModel})
    async get(
        @Param('id') id: number,
    ) {
        return this.orderService.findById(id);
    }

    @Get('/:id/report')
    @ApiBody({type: OrderModel})
    async getReport(
        @Param('id') id: number,
    ): Promise<string> {
        const file = await this.orderService.getReport(id);
        return path.join(path.resolve(__dirname, '../../../../../'), file.url);
    }

    @Post()
    @ApiBody({type: OrderSaveDto})
    @ApiOkResponse({type: OrderModel})
    async create(
        @Body() dto: OrderSaveDto,
    ) {
        return this.orderService.create(dto);
    }

    @Post('/:id')
    @ApiBody({type: OrderSaveDto})
    @ApiOkResponse({type: OrderModel})
    async update(
        @Param('id') id: number,
        @Body() dto: OrderSaveDto,
    ) {
        return this.orderService.update(id, dto);
    }
}
```

ПРИЛОЖЕНИЕ В

Слайды презентации

Рисунок В.1 – Титульный слайд

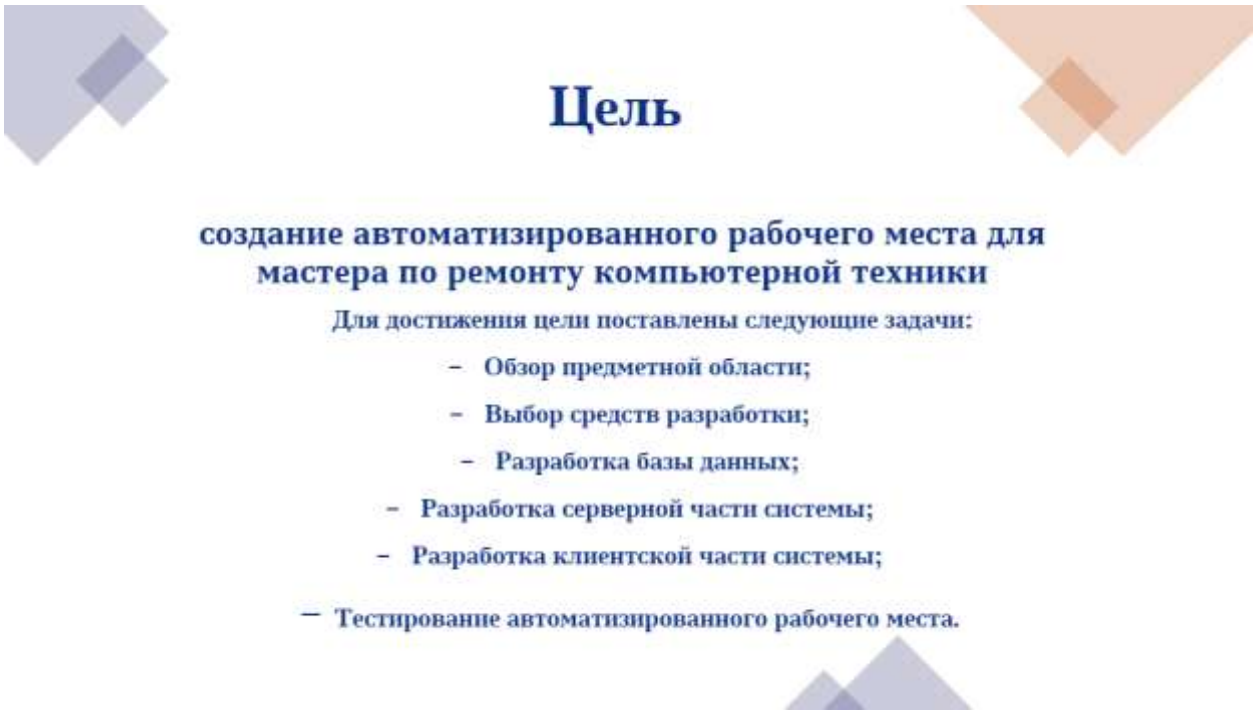

Рисунок В.2 – Слайд «Цель и задачи»

Сравнение аналогов

Характеристика	Gincore	LiveSklad	HelloClient
Учет материалов	+	+	-
Возможность просмотра истории заказов	-	+	+
Выдача справок клиентам	+	-	-
Создание заказов	-	+	+
Работа с клиентами	-	+	+
Счетчик рабочего времени	-	-	-

Рисунок В.3 – Слайд «Сравнение аналогов»

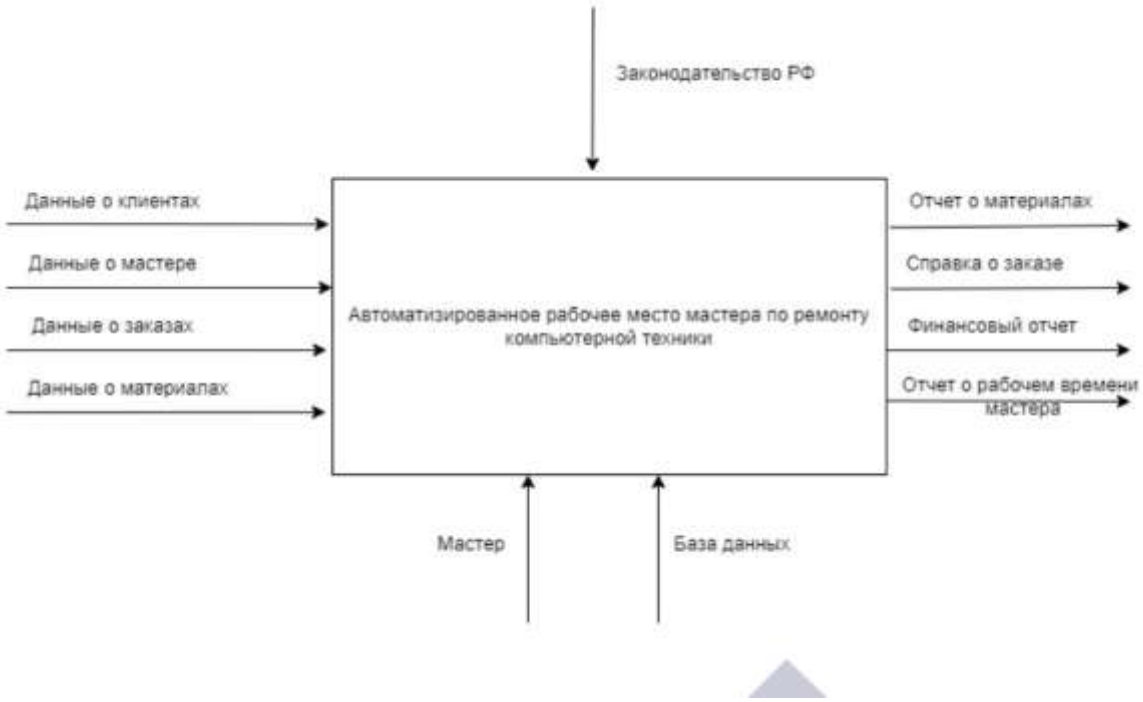

Рисунок В.4 – Слайд «Диаграмма IDEF0»

Рисунок В.5 – Слайд «Декомпозиция диаграммы IDEF0»

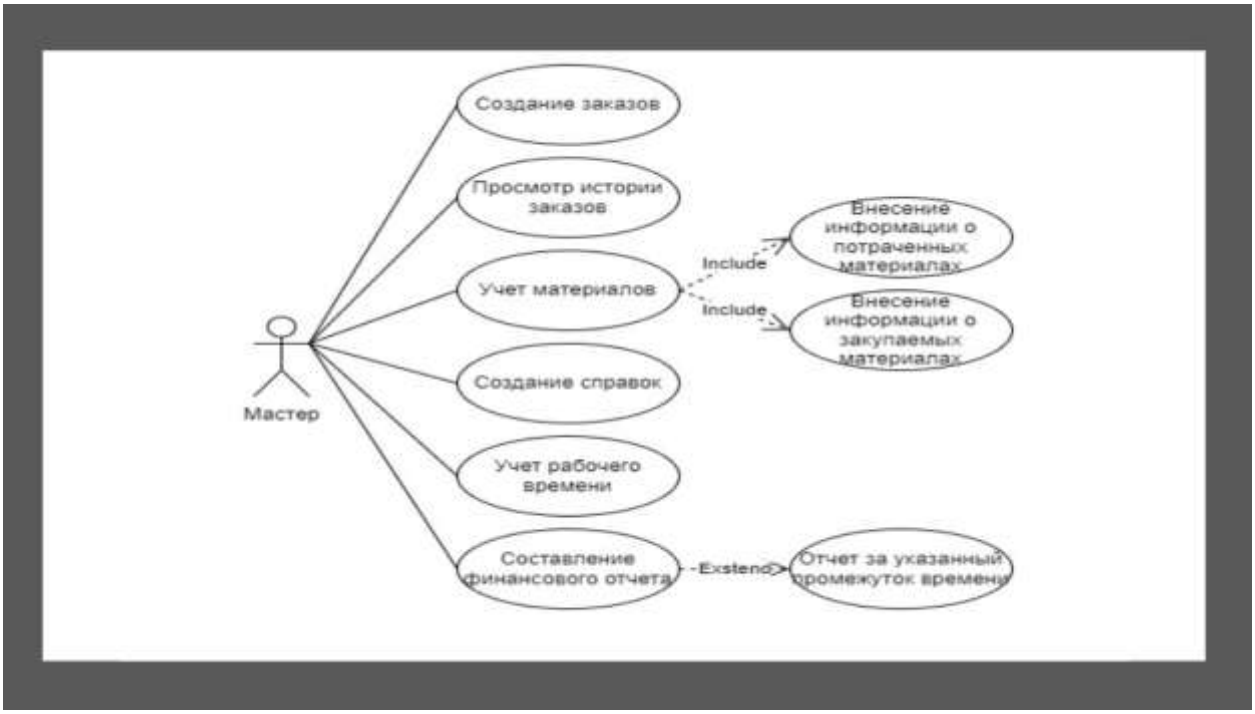

Рисунок В.6 – Слайд «Диаграмма вариантов использования»

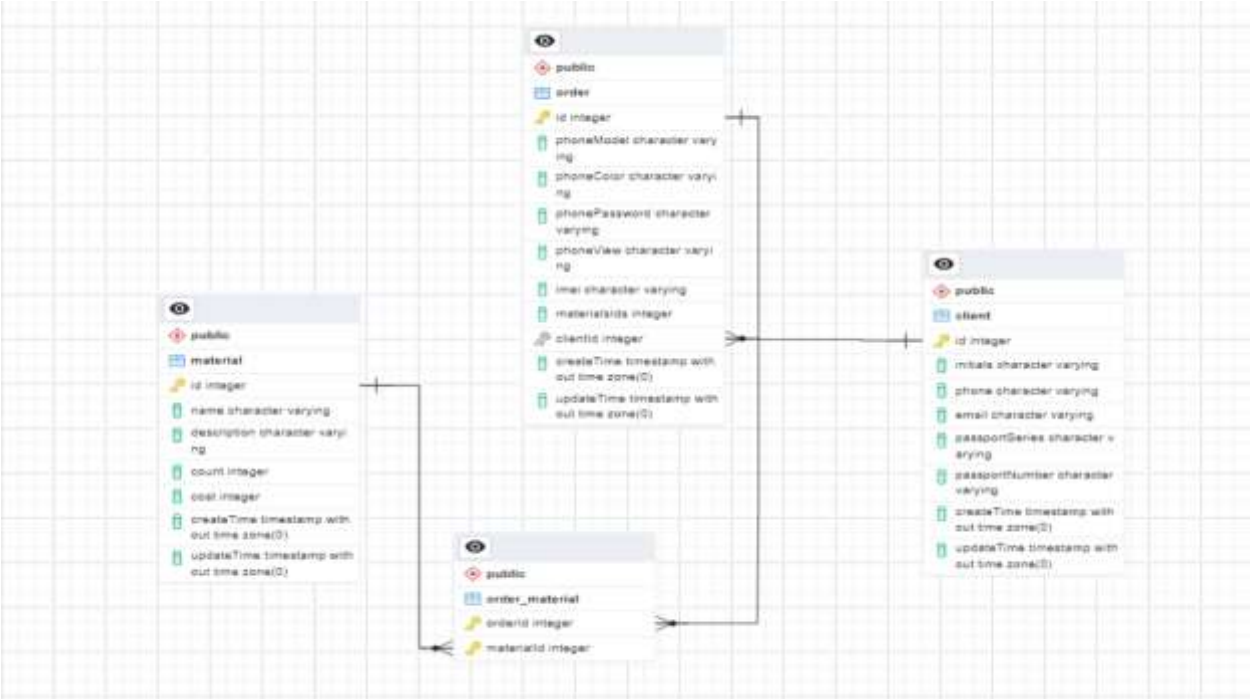

Рисунок В.7 – Слайд «База данных»

Рисунок В.8 – Слайд «Средства разработки»

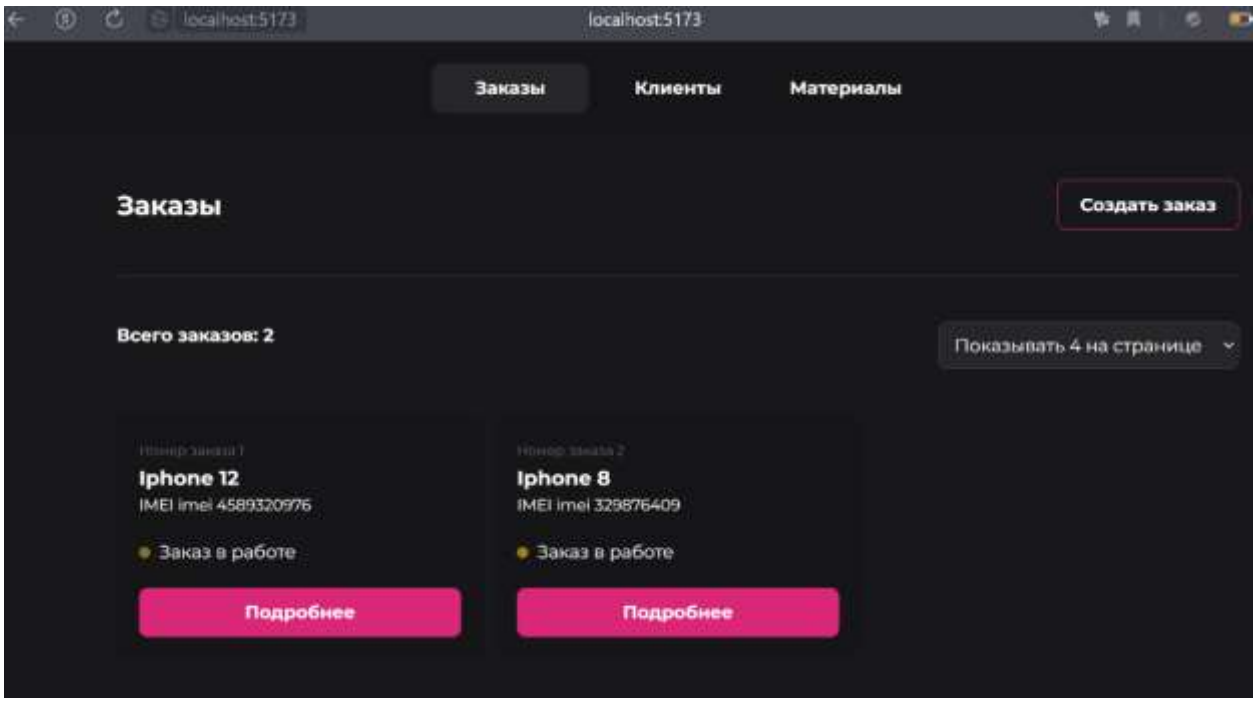

Рисунок В.9 – Слайд «Пример интерфейса»

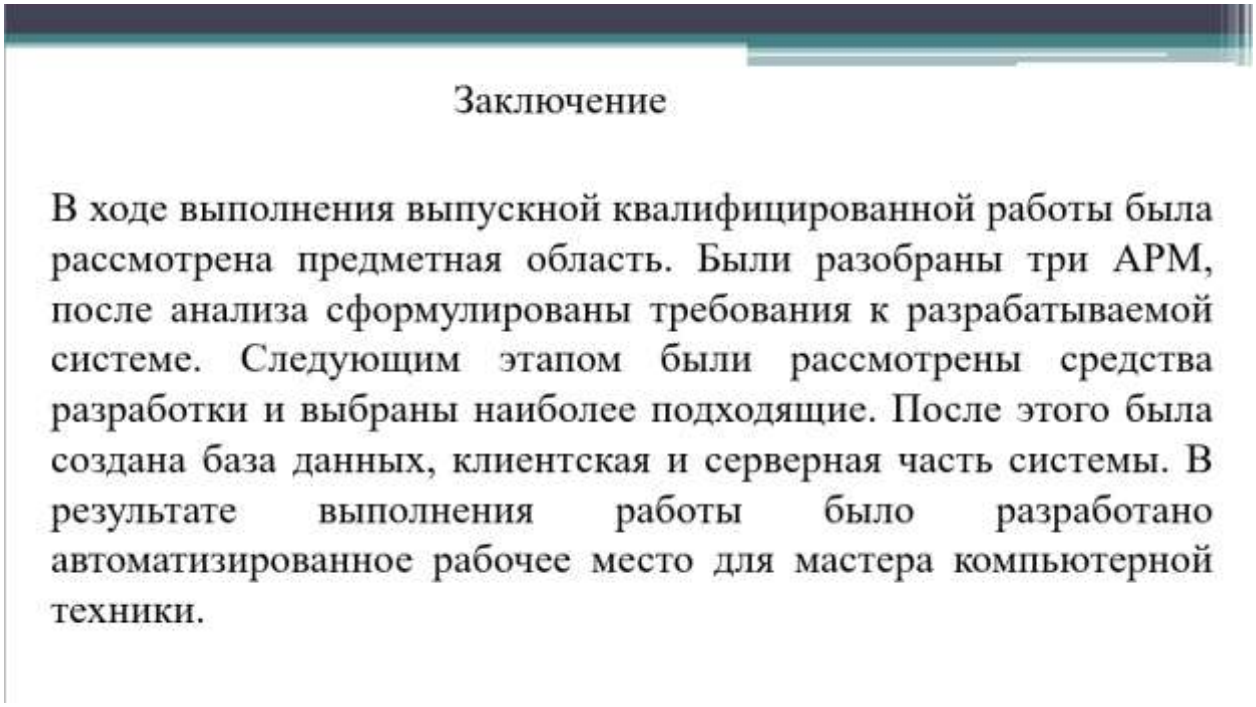

Рисунок В.10 – Слайд «Заключение»

Рисунок В.11 – Заключительный слайд

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
"19" 06 2023 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Автоматизированное рабочее место мастера по ремонту компьютерной техники

Руководитель	_____ подпись	19.06.23 дата	доцент, канд. техн. наук должность, ученая степень	С. Н. Титовский
Выпускник	_____ подпись	19.06.23 дата		А. В. Дешеков
Нормоконтролёр	_____ подпись	19.06.23 дата		С. Н. Титовский

Красноярск 2023