СОДЕРЖАНИЕ

[Перечень условных обозначений, символов и терминов 6](#_Toc122397032)

[Введение 8](#_Toc122397033)

[1Анализ и моделирование предметной области программного средства 9](#_Toc122397034)

[1.1 Описание предметной области 9](#_Toc122397035)

[1.2 Разработка функциональной модели предметной области 10](#_Toc122397036)

[1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 14](#_Toc122397037)

[1.4 Разработка информационной модели предметной области 15](#_Toc122397038)

[1.5 Модели представления программного средства и их описание 18](#_Toc122397039)

[1.6 Диаграмма последовательности 18](#_Toc122397040)

[1.7 Диаграмма деятельности 19](#_Toc122397041)

[1.8 Диаграмма развертывания 21](#_Toc122397042)

[2Проектирование и конструирование программного средства 23](#_Toc122397043)

[2.1 Постановка задачи 23](#_Toc122397044)

[2.2 Архитектурные решения 23](#_Toc122397045)

[2.3 Описание алгоритмов, реализующих бизнес-логику разрабатываемого программного средства 27](#_Toc122397046)

[2.4 Проектирование пользовательского интерфейса 29](#_Toc122397047)

[2.5 Обоснование выбора компонентов и технологий для реализации программного средства 30](#_Toc122397048)

[3Тестирование и проверка работоспособности программного средства 32](#_Toc122397049)

[4Руководство по развертыванию и использованию программного средства 37](#_Toc122397050)

[Заключение 50](#_Toc122397051)

[Список использованных источников 51](#_Toc122397052)

[Приложение А (обязательное) Отчет о проверке на заимствование в системе «Антиплагиат» 52](#_Toc122397053)

[Приложение Б (обязательное) Листинг кода алгоритмов, реализующих основную бизнес-логику 53](#_Toc122397054)

[Приложение В (обязательное) Листинг скрипта генерации базы данных 63](#_Toc122397055)

Перечень условных обозначений, символов и терминов

|  |  |  |  |
| --- | --- | --- | --- |
| БД (база данных) | | – представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины | |
| Информационная система | | – система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые), которые обеспечивают и распространяют информацию | |
| Нормальная форма | | – свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных | |
| Среда выполнения | | – вычислительное окружение, необходимое для выполнения компьютерной программы и доступное во время выполнения компьютерной программы | |
| СУБД (система управления базами данных) | | – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных | |
| ТЗ (техническое задание) | | – документ, содержащий требования заказчика к объекту разработки, определяющий порядок и условия её проведения | |
| *API* (*application programming interface*) | | – описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой | |
| *IDE* (*Integrated development environment*) | | – комплекс программных средств, используемый программистами для разработки программного обеспечения | |
| *IDEF* | | – методология функционального моделирования (англ. *function modeling*) и графическая нотация, предназначенная для формализации и описания бизнес-процессов | |
| *Java* | | – строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией *Sun* *Microsystems* | |
| *SQL* (*structured query language*) | | – язык структурированных запросов, декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных | |
| *MySQL Server* | | – свободная реляционная система управления базами данных | |
| *UML* (*Unified Modeling Language*) | | – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур | |
| *Декомпозиция* | | – разделение сложного объекта, системы, задачи на составные части, элементы. Она показывает из каких более мелких работ состоит основной процесс. | |
|  | |  | |
|  | |  | |
|  | |  | |

# Введение

Конкурентоспособность на рынке обусловлена наличием огромного числа участников, действующих в определенном секторе, и их стремлением получить наибольшую прибыль от ведения своей деятельности. Поэтому каждая организация сталкивается с проблемой борьбы за потребителя и конкуренцией в целом. Фирме необходимо подстраиваться под текущую ситуацию и развиваться высокими темпами, чтобы иметь возможность конкурировать с другими предприятиями и не потерпеть фиаско. Информационные технологии являются неотъемлемой частью любого направления деятельности, в том числе и торговли. Они способны повысить эффективность работы предприятия, повышая качество и скорость обслуживания. Повысив свою эффективность, фирма не только сможет составить достойную конкуренцию на рынке, но также и увеличит динамику развития, что является серьезным фактом для обоснования использования автоматизированных решений в осуществлении ее деятельности. Поэтому целью данного курсового проекта будет разработка и реализация программного продукта, который позволит автоматизировать деятельность по учету проданных товаров, исключит неструктурированное и неудобное хранение используемой документации, будет подсчитывать доход, а также анализировать эффективность работы магазина на основании выходных данных.

Для достижения поставленной цели необходимо выполнить ряд задач:

* провести анализ предметной области;
* изучить теоретические вопросы разработки и организации интернет-магазина;
* определить список функций, необходимых для эффективной работы программного продукта;
* составить полную схему алгоритма работы программы;
* спроектировать базу данных хранимой информации;
* реализовать серверную часть приложения, которая будет реализовывать бизнес логику, и будет выполнять работу с базой данных;
* реализовать клиентскую часть приложения, с удобным интерфейсом для пользователя;
* протестировать полученное программное средство и убедиться, что оно корректно реализует свою бизнес логику.

# Анализ и моделирование предметной области программного средства

## Описание предметной области

Основной целью деятельности магазина компьютерного оборудования является удовлетворение потребностей покупателя в приобретении товаров и услуг сферы компьютерной техники и электроники. Продукция данной сферы не производится на отечественном рынке, следовательно, немаловажную роль снабжения компьютерного магазина играют дистрибьюторы зарубежных фирм-производителей.

В свою очередь магазин компьютерной техники осуществляет реализацию продукции, т.е. занимается доведением товара до потребителя. Данный магазин —– юридически самостоятельная организация, деятельность которой направлена на продажу товаров компьютерной сферы различных производителей, с которыми был заключён договор на право представлять данную марку той или иной компьютерной продукции.

**Основные бизнес-процессы**

* закупка и поставка продукции в магазин;
* представление имеющейся продукции покупателю;
* продажа компьютерной продукции.

Закупка и поставка продукции в магазин является основным начальным процессом для, непосредственно, самой продажи продукции.

Представление имеющейся продукции покупателю включает: выставление товара на витрины магазина, предоставление характеристик соответствующего товара, а также предоставление возможности покупателю протестировать товар.

Продажа компьютерной продукции включает: заключение договоров купли-продажи, заявок на предоставление дополнительных услуг, и соответственно подготовку товара к выдаче. При ответе на вопрос, что это за материал – трикотаж, можно отметить, что он отличается от иных тканей превосходной растяжимостью во всех направлениях. При этом очень быстро после завершения приложения усилия материал возвращает свой изначальный размер и форму.

**Вспомогательные бизнес-процессы**

* поддержка сайта данного магазина;
* распространение рекламы;
* создание тестовых стендов.

Поддержка сайта включает в себя: обеспечение бесперебойной работы сайта, улучшение пользовательского интерфейса, разработка функционала, обновление каталога продукции.

Распространение рекламы состоит из: привлечения потенциальных клиентов различными способами (в сети Интернет, на билбордах, с помощью листовок, визиток и тп.).

Создание тестовых стендов —– оборудование стенда для возможности протестировать товар клиентом.

**Продажа компьютерной продукции через сайт данного магазина**

Выбрав необходимые товары пользователь обычно имеет возможность тут же на сайте выбрать метод оплаты и доставки. Совокупность отобранных товаров, способ оплаты и доставки представляют собой законченный заказ, который оформляется на сайте путем сообщения минимально необходимой информации о покупателе. Информация о покупателе может храниться в базе данных магазина если бизнес-модель магазина рассчитана на повторные покупки, или же отправляться разово. В интернет-магазинах, рассчитанных на повторные покупки, также ведется отслеживание возвратов посетителя и история покупок. Часто при оформлении заказа предусматривается возможность сообщить некоторые дополнительные пожелания от покупателя продавцу.

Интернет-магазины создаются с применением систем управления контентом сайтов, оснащенных необходимыми модулями. Крупные интернет-магазины работают на специально для них разработанных или адаптированных типовых системах управления. Средние и малые магазины обычно используют типовое коммерческое и свободное ПО. К примеру, широко известен свободный движок osCommerce.

Система управления контентом сайта интернет-магазина может быть коробочным продуктом, самостоятельно устанавливаемым на [хостинг](https://pandia.ru/text/category/hosting/)-площадку, может быть частной разработкой веб-студии, ей же обслуживаемой, или может быть программным сервисом, предоставляемым с помесячной оплатой.

Нужды администраторов интернет-магазина в складском, торговом, бухгалтерском и налоговом учете должны поддерживаться невидимой посетителям частью интернет-магазина — бэк-офисом. Экономически эффективной практикой создания интернет-магазинов является применение специализированных систем учета. Интернет-магазин обычно интегрирован с такими системами учета.

## Разработка функциональной модели предметной области

Основным процессом предметной области данного курсового проекта является осуществление продаж в магазине компьютерного оборудования. Он включает в себя множество особенностей и формальностей, требует особого внимания, выполнения всех требований и правил. Следовательно, существует необходимость в изложении всех пунктов и действий каждой из сторон данной операции (рисунок 1.1). Для этого был использован стандарт *IDEF*0 и программное средство *AllFusion* *Process* *Modeler*.

В результате анализа предметной области было получено, что в качестве входных параметров для системы выступают данные о пользователе и запросы пользователя. К управляющим воздействиям относятся правила пользования и законы. В качестве механизма осуществления главной функции выступают, администратор, который отвечает за изменение каталога, управление пользователями, контролем функционирования магазина, графический-интерфейс для осуществления связи клиента и магазина. Выходным параметром для данной системы являются электронный чек и отправка товара.



Рисунок 1.1 – Контекстный уровень диаграммы.

Далее представлена декомпозиция контекстной диаграммы, состоящая из трех блоков (рисунок 1.2):

1. Регистрация пользователя.
2. Добавление товаров в корзину.
3. Подтверждение заказа.

Декомпозиция – это разделение сложного объекта, системы, задачи на составные части, элементы. Она показывает из каких более мелких работ состоит основной процесс.

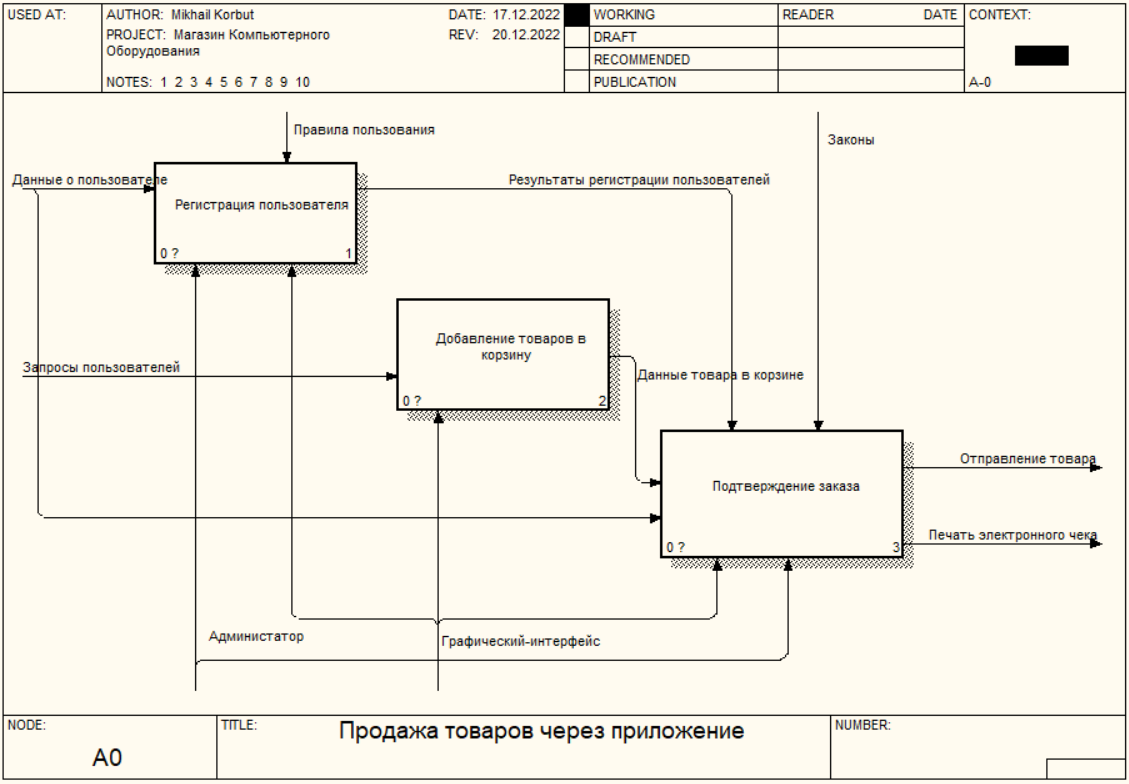


Рисунок 1.2 – Декомпозиция контекстной диаграммы.

Этап «Добавление товаров в корзину» разбит на два функциональных блока (рисунок 1.3):

1. Проверка наличия товара на складе.
2. Отправка данных о товаре в корзину.

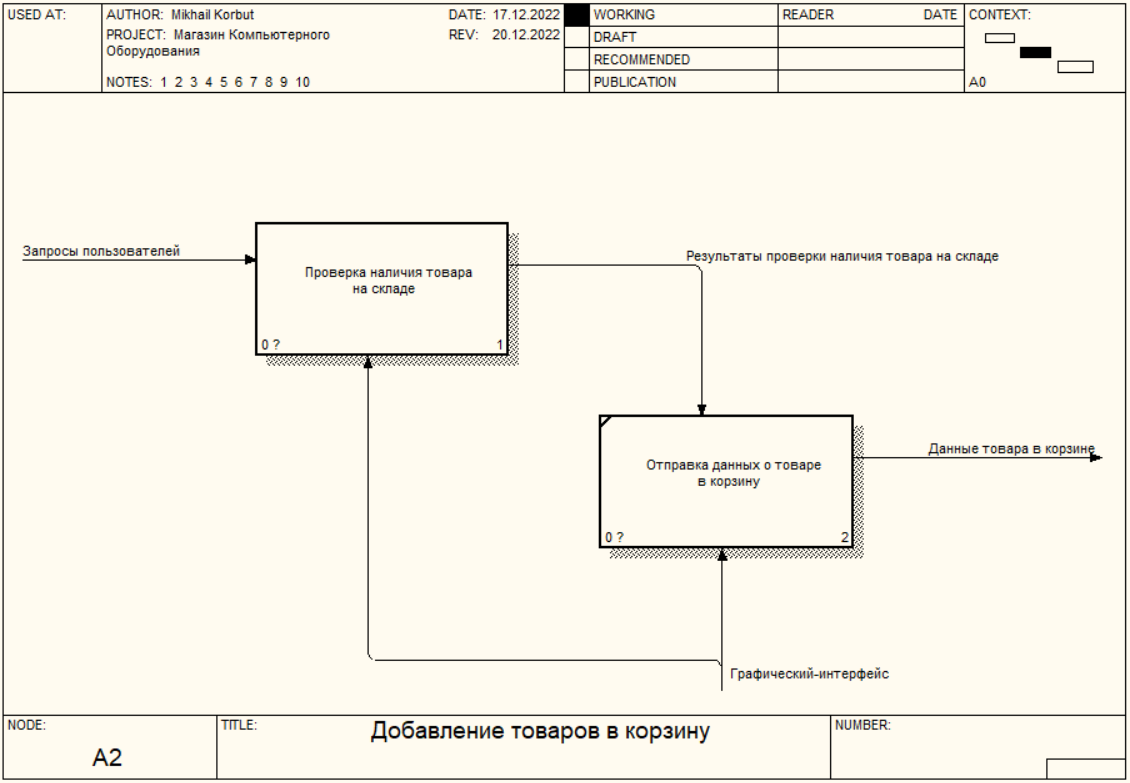


Рисунок 1.3 – Декомпозиция блока «Добавление товаров в корзину».

Этап «Проверка наличия товара на складе» разбит на два функциональных блока (рисунок 1.4):

1. Запрос информации о товаре на складе.
2. Анализ результатов запроса товара

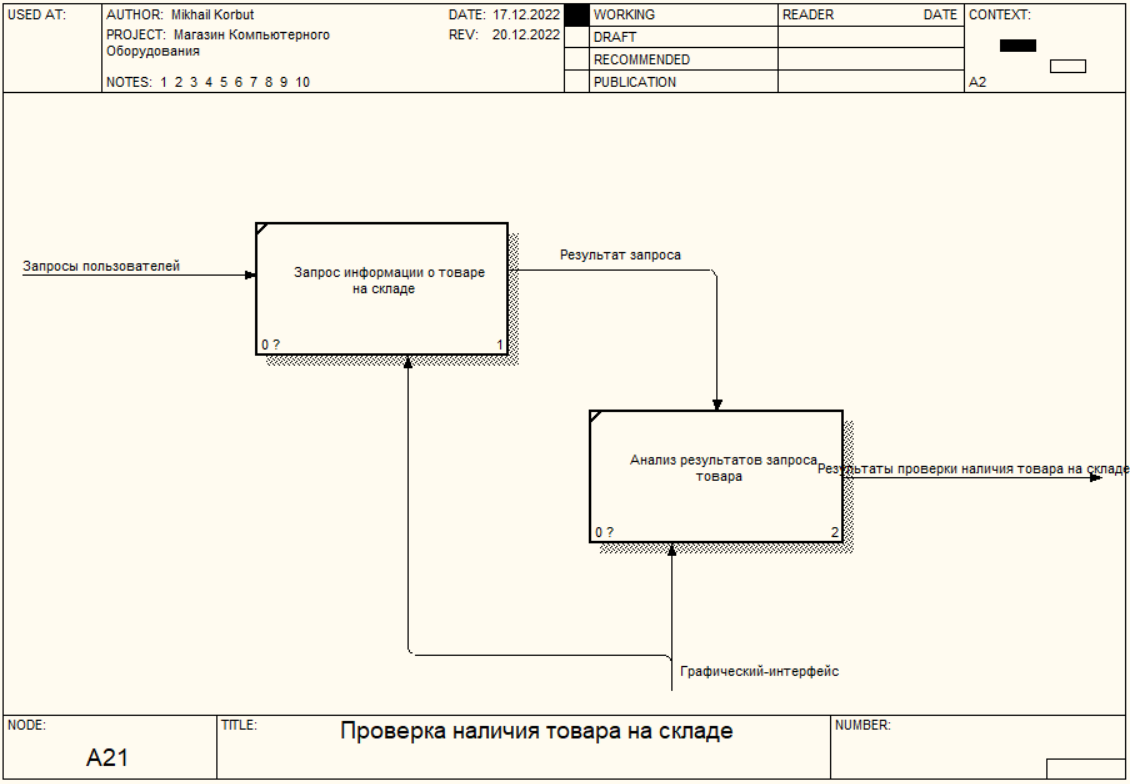


Рисунок 1.4 – Декомпозиция блока «Проверка наличия товара на складе».

Этап «Подтверждение заказа» разбит на три функциональных блока (рисунок 1.5):

1. Запрос информации о товаре на складе.
2. Подтверждение заказа пользователя.
3. Подтверждение заказа.

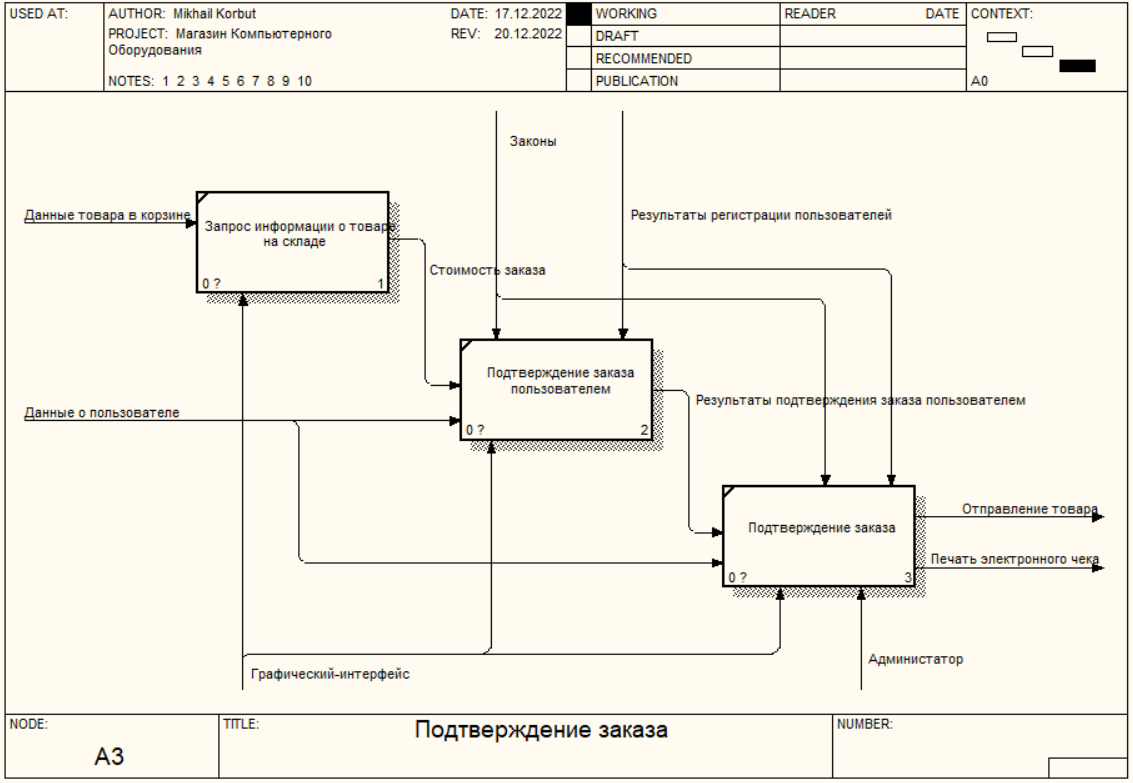


Рисунок 1.5 – Декомпозиция блока «Подтверждение заказа».

## Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований

Диаграмма вариантов использования (сценариев поведения, прецедентов) является исходным концептуальным представлением системы в процессе ее проектирования и разработки. Данная диаграмма состоит из актеров, вариантов использования и отношений между ними.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества актеров, взаимодействующих с системой с помощью так называемых вариантов использования. Каждый вариант использования определяет некоторый набор действий, совершаемых системой при взаимодействии с актером. При этом в модели никак не отражается то, каким образом будет реализован этот набор действий.

Оба актера могут зайти под своей ролью и, в зависимости от авторизации, им даются разные возможности (рисунок 1.6).

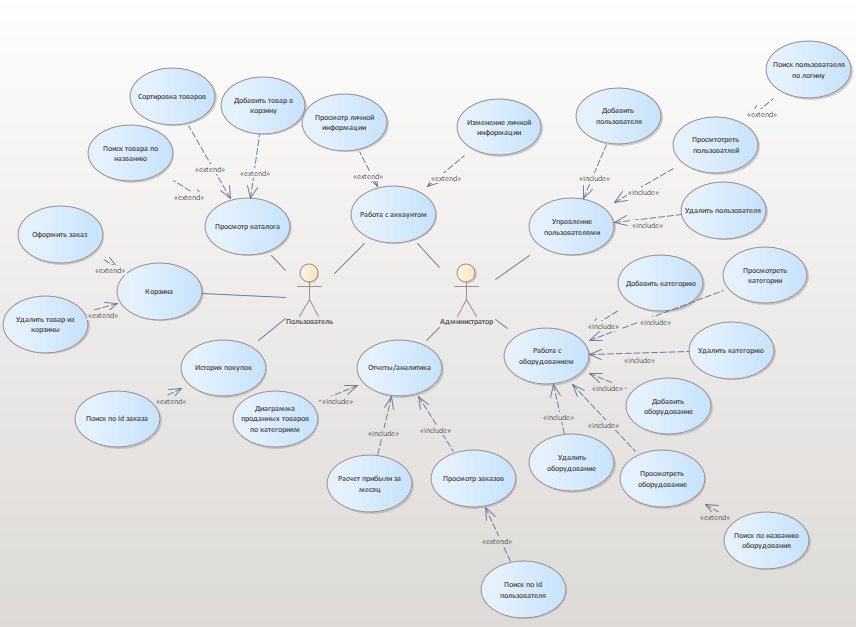


Рисунок 1.6 – Диаграмма вариантов использования.

При первом входе в приложение пользователь должен зарегистрироваться, после этого авторизоваться В приложении пользователь может просмотреть каталог товаров, выбрать интрересующий и добавить его в корзину. Чтобы оформить покупку пользователю нужно перейти в корзину и далее запонить необходимые поля. При необходимости пользователь может изменить данные о себе, свой логин и пароль в личном кабинете. Для изменения пароля необходимо ввести старый пароль, при нежелании дальше взаимодействовать с приложением.

Администратор имет возможности добавление категории, товаров. Также управление пользователями и просмотр отчетов по деятельности магазина.

## Разработка информационной модели предметной области

При проектировании системы было принято решение использовать следующие сущности:

* *admins*;
* *users*;
* *keys*;
* *equipment*;
* *basket*;
* *categories*;
* *Orders*.

Графическое отображение информационной модели приведено на диаграмме рисунке 1.7.

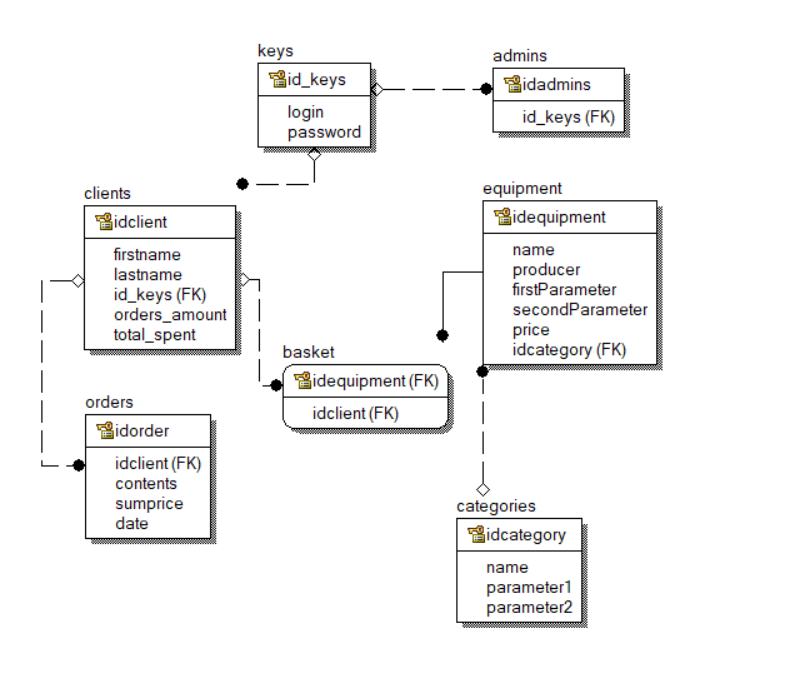


Рисунок 1.7 - Информационная модель системы.

Информация об аккаунтах хранится в сущности «keys». Данные в нее вносятся при регистрации, но также могут изменятся в последующем.

Для хранения информации о клиентах есть сущность «clients», она исключает повторение данных.

Cущности «categories», «equipment» хранят в себе данные о категориях и оборудовании соответственно. Данные могут изменятся в последующем.

В сущности «orders» хранится информация о заказах клиента, каждый заказ привязан к id определенного клиента.

Сущность *keys* содержит в себе следующие атрибуты:

* *id\_keys* – хранит уникальный номер пользователя;
* *login* –хранит уникальный логин пользователя;
* *password* – хранит пароль пользователя;

Сущность *admins* содержит в себе следующие атрибуты:

* *idadmin*– хранит уникальный идентификатор администратора;
* *id\_keys(FK)* - атрибут унаследованный от сущности *keys;*

Сущность *clients* содержит в себе следующие атрибуты:

* *idclient*– хранит уникальный идентификатор клиента;
* *firstname* – хранит имя клиента;
* *secondname* – хранит фамилию клиента;
* *orders\_amount* – хранит количество заказов клиента;
* *total\_spent* – хранит сумму стоимостей всех заказов клиента;

Сущность *categories* содержит в себе следующие атрибуты:

* *idcategory*– уникальный идентификатор категории;
* *name* – название категории;
* *parameter1* – первый параметр;
* *parameter2* – второй параметр;

Сущность *equipment* содержит в себе следующие атрибуты:

* *idequipment*– уникальный идентификатор оборудования;
* *name* – обхват ладони;
* *producer* –название производителя;
* *idcategory(FK)* - атрибут унаследованный от сущности *category*;
* *firstParameter*-первый параметр;
* *secondParameter*-второй параметр;
* *price* – стоимость товара;

Сущность *basket* содержит в себе следующие атрибуты:

* *idequipment(FK)* - атрибут унаследованный от сущности *equipment*;

Сущность *orders* содержит в себе следующие атрибуты:

* *idorder*– уникальный идентификатор заказа;
* *idclient(FK)* - атрибут унаследованный от сущности *client*;
* *contents* – содержание заказа;
* *sumprice* – стоимость заказа;
* *date –* дата оформления;

Любые обычные отношения находятся в первой нормальной форме, значит, разработанная модель находится в первой нормальной форме. Когда отношение находится в первой нормальной форме и нет не ключевых атрибутов, которые зависят от части сложного ключа, тогда отношение находится во второй нормальной форме. Разработанная модель находится в первой нормальной форме, а также нет ключевых атрибутов, являющихся зависимыми от части сложного ключа, следовательно, модель находится во второй нормальной форме. Если все не ключевые атрибуты взаимно независимы и модель находится во второй нормальной форме, то модель находится в третьей нормальной форме. В разработанной модели все не ключевые атрибуты взаимно независимы, а также модель находится во второй нормальной форме, следовательно, модель находится и в третьей нормальной форме.

*SQL*-скрипт для генерации базы данных приведен в приложении B.

## Модели представления программного средства и их описание

Язык UML предназначен для описания моделей, причем для работы с этим языком используется специальные редакторы диаграмм. На UML можно содержательно описывать классы, объекты и компоненты в различных предметных областях, часто сильно отличающихся друг от друга [7].

Диаграммы UML – графическое представление набора элементов, изображенное чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

1.6 Диаграмма последовательности

Диаграмма последовательности является одной из разновидности диаграмм взаимодействия и предназначена для моделирования взаимодействия объектов системы во времени, а также обмена сообщениями между ними.

На диаграмме последовательности объекты в основном представляют экземпляры класса или сущности, обладающие поведением. В качестве объектов могут выступать пользователи, инициирующие взаимодействие, классы, обладающие поведением в системе или программные компоненты, а иногда и системы в целом. Данный вид диаграммы отображен на рисунке 1.8.

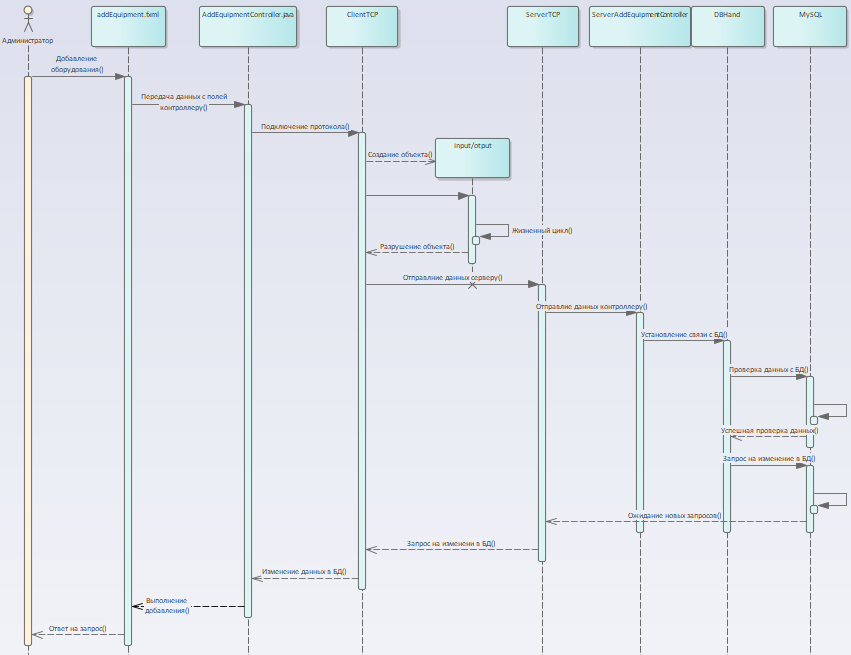


Рисунок 1.8 – Диаграмма последовательности процесса добавления оборудования.

На данной диаграмме администратор добавляет оборудование, вводит необходимые данные в окно графического приложения, которые считывает контроллер клиента и передает по протоколу TCP в объект, которые отправляется на сервер. Сервер обрабатывает данные, подключается к БД.

Там в свою проверяется отсутствие такого же товара и в случае его отсутствия происходит добавление оборудования в базу данных.

В случае успеха или ошибки сервер возвращает клиенту соответствующее сообщение, которое получает администратор на экране приложения.

1.7 Диаграмма деятельности

На диаграмме деятельности представлены переходы потоков управления от одной деятельности к другой внутри системы. Используется при моделировании функционирующей системы т.к. отражает передачу потока управления между объектами. Ее основное назначение – отражение бизнес-процессов объекта. Позволяет показать последовательность процесса, ветвление, синхронизацию процессов. Она позволяет проектировать алгоритм поведения объектов любой сложности, в том числе она может быть использована для составления блок-схем.

Данный процесс начинается, когда пользователь входит в приложение и решает зарегистрироваться. Вначале Клиентская часть должна загрузить форму для пользователя, на форме есть обязательные поля, отмеченные символом «\*» и дополнительные. После заполнения формы, Клиент считывает введенный логин и отправляет его на Сервер по протоколу TCP для проверки. Сервер подключается к БД и проверяет существует ли пользователь с таким логином и отправляет сообщение Клиенту об успешности. Если такой логин уже существует, пользователю выводится сообщение об ошибке, далее ему придется выбрать другой логин. Если логин не существует Клиентская часть считает оставшиеся данные и отправит их на Сервер, который сохранит их в БД. После успешной регистрации, клиентская часть закроет окно регистрации и откроет главное меню (меню входа и регистрации). Данный вид диаграммы отображен на рисунке 1.9.

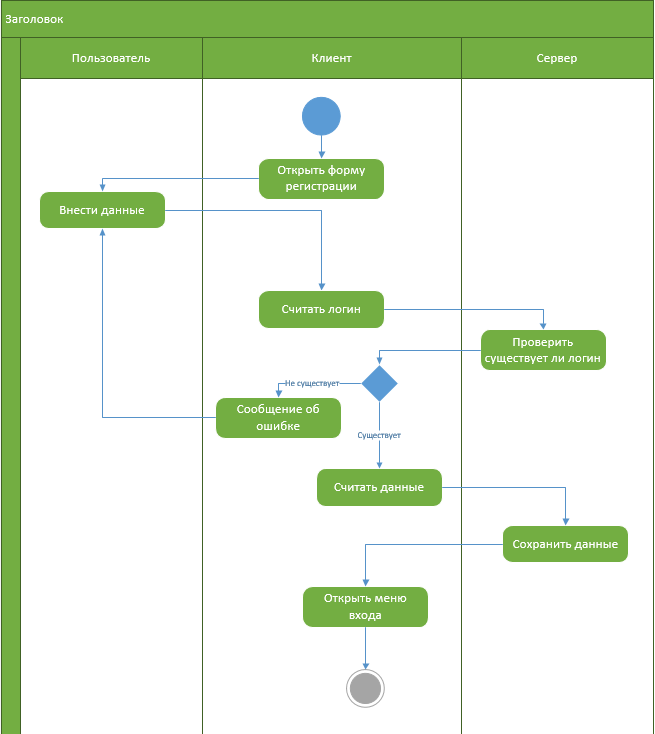


Рисунок 1.9 – Диаграмма деятельности.

1.8 Диаграмма развертывания

Диаграмма развертывания – это тип UML-диаграммы, которая показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их.

Диаграмма развёртывания показывает топологию системы и распределение компонентов системы по ее узлам, а также соединения - маршруты передачи информации между аппаратными узлами. Это единственная диаграмма, на которой применяются “трехмерные” обозначения: узлы системы обозначаются кубиками.

Диаграмма развертывания представлена на рисунке 1.10.

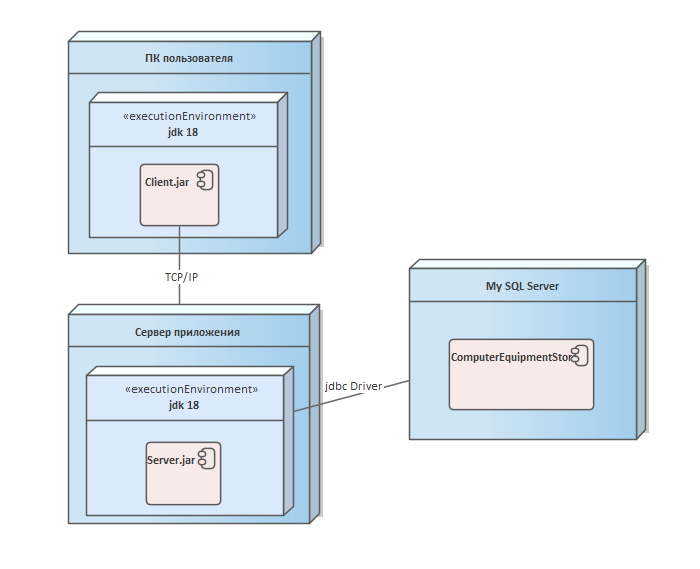


Рисунок 1.10 – Диаграмма развертывания приложения.

В качестве узлов выступает ПК пользователя приложением, сервер и БД.

Для запуска разрабатываемого приложения необходимо наличие исполняемой среды JDK18 на компьютерах пользователей школы иностранных языков.

Связь узлов ПК с сервером осуществляется по протоколу TCP/IP, а связь сервера с БД осуществляется по протоколу JDBC.

# Проектирование и конструирование Программного средства

## Постановка задачи

Для разработки программного средства необходимо определить, для кого оно будет создаваться и что необходимо этому кругу лиц. Далее на основе этих данных определить, что нужно реализовать в программном продукте. Разрабатывается данный проект для всех работающих с учётом товаров в магазине электротоваров. Целью данного проекта является автоматизация процесса учёта товаров. Для реализации поставленных целей данного курсового проекта необходимо выполнить следующие задачи:

* реализовать клиент-серверное приложение;
* связать сервер с базой данных;
* в качестве базы данных использовать MySQL;
* создать оконные формы для удобной работы пользователя;
* сделать приложение простым и понятным в использовании;
* разработать базу данных;
* протестировать полученное приложение.

## Архитектурные решения

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации) [8].

На рисунке 2.1 представлена диаграмма классов пакета DB.

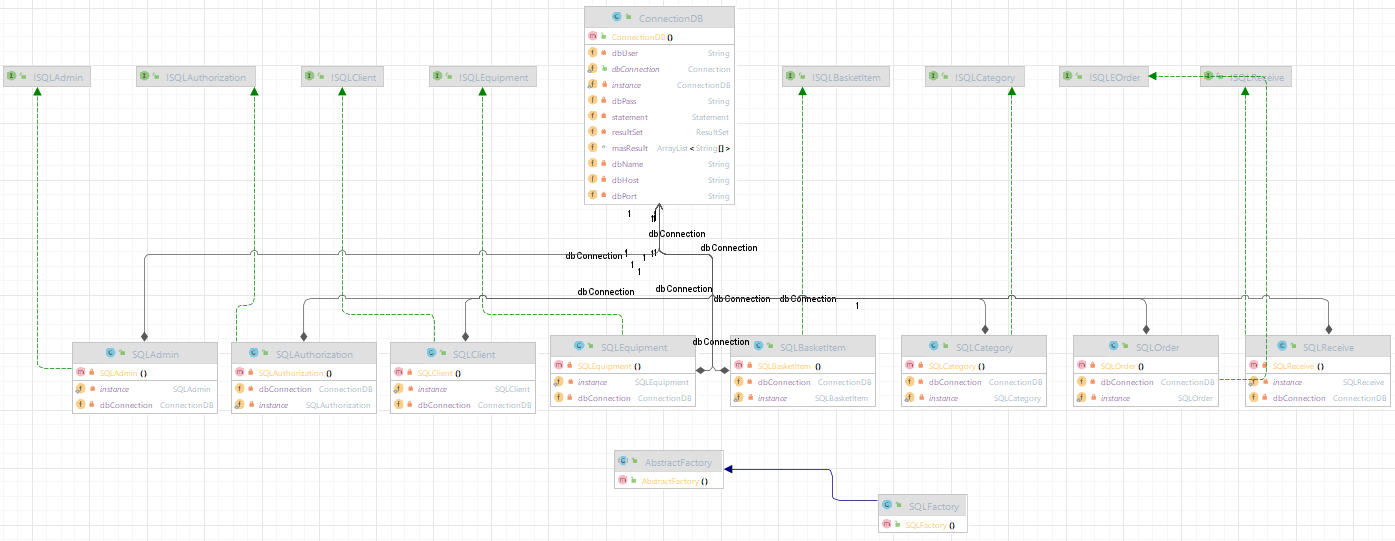


Рисунок 2.1 – Диаграмма классов пакета DB

Данная диаграмма реализует паттерн абстрактная фабрика. **Абстрактная фабрика** — это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов. Диаграммы классов показаны на рисунке 2.2.

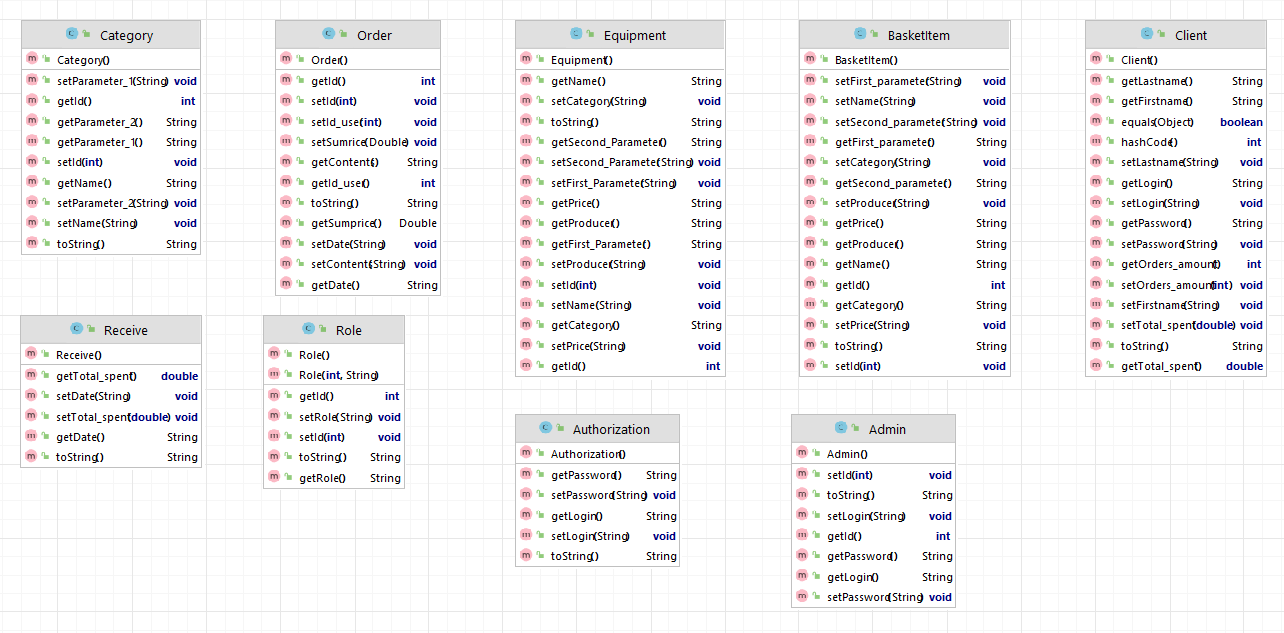


Рисунок 2.2 – Диаграмма классов пакета Computerequipmentstore.

Объекты этого класса наследуются от интерфейса Serializable.

Сериализация представляет процесс записи состояния объекта в поток, соответственно процесс извлечения или восстановления состояния объекта из потока называется десериализацией. Сериализация очень удобна, когда идет работа со сложными объектами.

Таким образом, объекты этих классов служат для их отправления на клиент или наоборот, принятия от клиента.

Реализация паттерна singleton представлена на рисунке 2.3

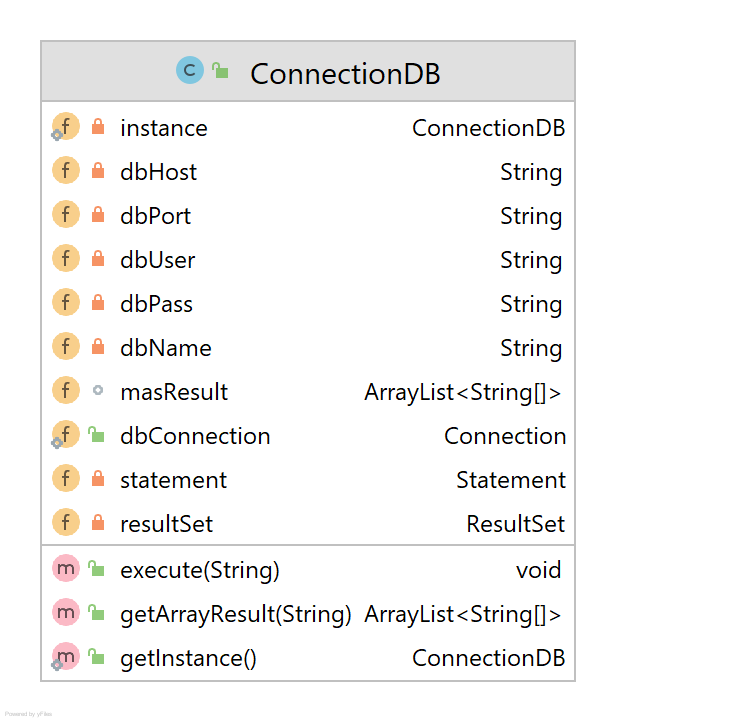


Рисунок 2.3 – Реализация паттерна Singleton.

Singleton относится к порождающим паттернам. Этот паттерн гарантирует, что у класса есть только один объект (один экземпляр класса) и к этому объекту предоставляется глобальная точка доступа.

В курсовом проекте singleton используется для порождения главного контроллера, создание соединения с базой данных.

Разрабатываемый программный продукт представляет собой клиент-серверное приложение для работы с СУБД *MySQL Server* через веб-сервис. Предполагается развертывание в организации, имеющей локальную вычислительную сеть с выделенным сервером. В таком случае на выделенном сервере устанавливается СУБД, на которой развертывается база данных системы, и создается директория для хранения документов. Экземпляры клиентских приложений размещаются на машинах сотрудников организации, на которых предварительно должен быть установлен *JDK* 1.8 и выше. Доступ клиентских приложений к базе данных осуществляется при помощи драйвера *jdbc data provider for MySQL Server*, входящего в состав *JDK*. Данные, необходимые для установки соединения с сервером, хранятся в конфигурационном файле клиентского приложения и доступны для настройки пользователем. Настройки, используемые всеми приложениями системы, хранятся в одной из таблиц базы данных. Это позволяет производить изменения, которые распространяются сразу на все экземпляры приложений, с другой стороны, доступ к настройкам системы закрыт для рядовых пользователей.

Система предусматривает один способ ввода информации – вручную через пользовательский интерфейс приложения и один способ вывода – на экран машины сотрудника.

Экземпляры приложений могут взаимодействовать между собой для пересылки уведомлений сотрудникам. Для этого используются протоколы стека *TCP/IP*. Сетевое взаимодействие обеспечивается средствами архитектуры *java*.*net*.

Диаграмма компонентов описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный и исполняемый код. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними. Диаграмму можно увидеть на рисунке 2.4.

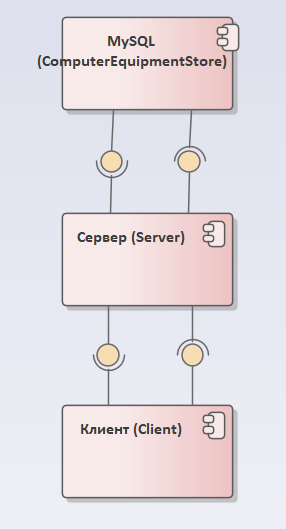


Рисунок 2.4 – Диаграмма компонентов.

Разделение приложения на составные части, каждая из которых реализует определенную часть функциональности, позволяет повысить структурированность системы, позволяет повторно использовать компоненты; обеспечивает большую гибкость продукта и облегчает его изменение.

## Описание алгоритмов, реализующих бизнес-логику разрабатываемого программного средства

Бизнес логика данного курсового проекта - это организация процесса покупки оборудования клиентом, остальной функционал является вспомогательным, поэтому в этой главе рассмотрим два процесса: процесс добавления нового оборудования и процесс заказа товара.

При входе в программу от имени администратора нужно выбрать пункт меню «Работа с оборудованием», после чего выбрать «Добавить оборудование». Для добавления оборудования необходимо заполнить все поля, нужно учесть, что два товара с одинаковым название существовать в базе данных не могут.

Если при добавлении оборудования были допущены ошибки, то на экран выводится соответствующее сообщение и можно попытаться добавить товар снова.

Алгоритм работы программы представлен на рисунке 2.5.

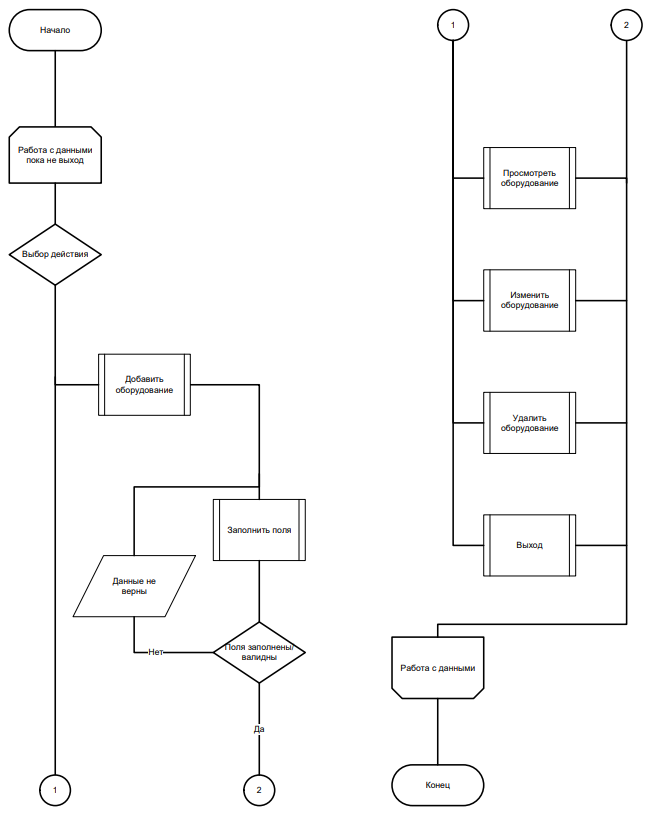


Рисунок 2.5 – Добавление оборудования.

Второй процесс - это добавление товара в корзину и последующие оформление заказа клиентом. После просмотра каталога можно выбрать понравившийся товар и добавить его в корзину, введя название оборудования в поле. Введенные данные отправляются на сервер, который начинает их обрабатывать, подключаясь к базе данных. Если там есть соответствующие данные, то оборудование помещается в корзину, а на клиента отправляется сообщение об успешном добавлении товара, в противном случае сообщение об ошибке.

Алгоритм работы программы представлен на рисунке 2.6.

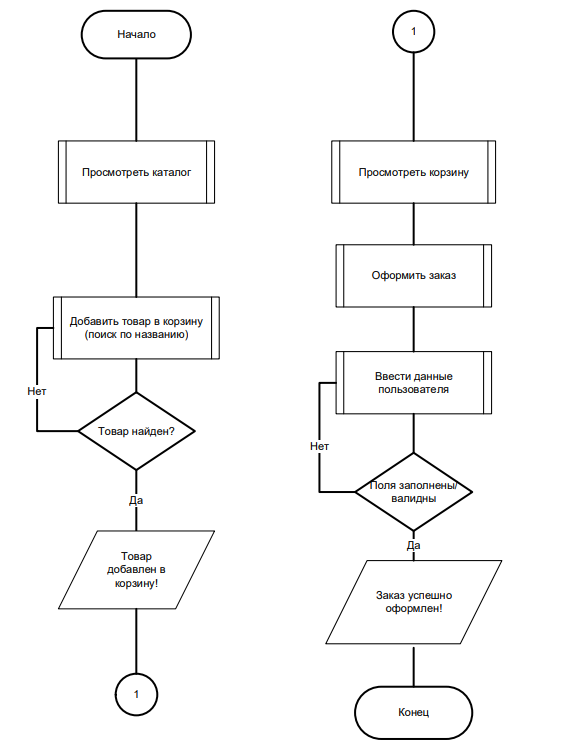


Рисунок 2.6 – Оформление заказа.

## Проектирование пользовательского интерфейса

Пользовательский интерфейс – это система средств для взаимодействия пользователя с компьютером, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т.п.). При этом, в отличие от интерфейса командной строки, пользователь имеет произвольный доступ (с помощью клавиатуры или указательного устройства ввода) ко всем видимым экранным объектам, а на экране реализуется модель мира в соответствии с некоторой метафорой и осуществляется прямое манипулирование.

Основное достоинство хорошего интерфейса пользователя заключается в том, что пользователь всегда чувствует, что он управляет программным обеспечением, а не наоборот.

Для создания у пользователя такого ощущения «внутренней свободы» интерфейс должен обладать рядом свойств:

* Естественность интерфейса (не вынуждает пользователя существенно изменять привычные способы решения задач);
* Согласованность интерфейса (позволяет переносить знания на новые задания, для реализации этого свойства, все формы для создания выкроек должны иметь схожее оформления, а именно поля для ввода разделены на три части: дополнительная информация (название, пряжа, плотность), мерки (прилагается картинка), плотность (значения петельной пробы) (рисунок 4.12). Далее во всех формах идет сама выкройка, при нажатии на кнопку рассчитать, на ней отобразятся значения, а снизу появится текстовое описание (рисунок 4.14);
* Дружественность интерфейса (принцип «Прощения пользователя»), для реализации этого свойства, необходимо выводит сообщения при неправильном вводе, не разрешать выполнение действий, пока данные являются не корректными, так же необходимо подтверждать важные действия (например «Вы точно хотите удалить аккаунт»);
* Принцип обратной связи, на все действия пользователя должен приходить ответ, визуальное текстовое сообщение (например об успешности сохранения данных). Если не удалось выполнить действие, нужно выводить причину почему не удалось.
* Простота интерфейса;
* Гибкость интерфейса – способность учитывать уровень подготовки и производительность труда пользователя. Так как приложение создается для использования всеми слоями населения, в нем не должно быть сложной терминологии и все выкройки (чертежи) должны быть интуитивно понятны, а также иметь проработанное описание;
* Эстетическая привлекательность.

## Обоснование выбора компонентов и технологий для реализации программного средства

Программное приложение должно быть написано на объектно-ориентированном языке *Java* и использовать архитектуру «клиент-сервер». Клиент-серверная архитектура – взаимодействие двух самостоятельных процессов – клиента и сервера, которые могу выполняться как на одном, так и на разных компьютерах, обмениваясь данными по сети (см. рисунок 2.9). Серверы – процессы, которые реализуют определенную службу (например, службу файловой системы или БД), и принимают запросы. Клиенты – процессы, которые запрашивают службы у серверов с помощью отправки запросов и последующего ответа от него. Таким образом, архитектура «клиент-сервер» позволяет эффективно распределить работу между клиентскими и серверными частями системы: приложения, которые работают на стороне клиента не читают записи базы данных «напрямую», а посылают запросы на сервер, где они обрабатываются, результаты обработки отсылаются назад клиенту, что сокращает потоки информации.

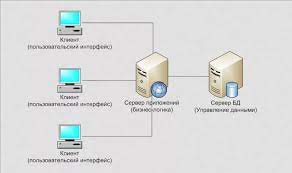


Рисунок 2.7 – Архитектура «клиент-сервер».

Клиент реализует пользовательский интерфейс. Для этого используется библиотека JavaFX. Она предоставляет большие возможности по сравнению с рядом других подобных платформ, в частности, по сравнению со Swing. Это и большой набор элементов управления, и возможности по работе с мультимедиа, двухмерной и трехмерной графикой, декларативный способ описания интерфейса с помощью языка разметки FXML, возможность стилизации интерфейса с помощью CSS, интеграция со Swing и многое другое. Бизнес-логика курсового проекта выполняется на серверной части. Клиент отправляет запросы на сервер, где с помощью *SQL*-запросов будет добавляться, обновляться, удаляться и выбираться вся необходимая информация из БД.

Основной средой разработки был выбран *IDE VSCode,* самым главным преимуществом которого является легкость данной *IDE* в сравнении например с *IntelliJ IDEA*. *MySQL Server* используется в качестве СУБД. *Enterprise Architect* используется для разработки и построения *UML*-диаграмм (*Use Case, Statechart, Sequence diagram*, диаграмма классов, *Component diagram, Deployment diagram*). *CASE-*средство *CA AllFusion Process Modeler r*7 (*BPwin*) используется для проектирование *IDEF*0 модели. *CA AllFusion ERwin Data Modeler r*7 (*ERwin*) используется для проектирования информационной модели. *Microsoft Visio* для проектирования блок схем и чертежей*.*

# Тестирование и проверка работоспособности программного средства

Тестирование – это процесс проверки функционала программы с целью подтверждения того, что она работает в соответствии с определёнными требованиями. Unit-тестирование – это тестирование, которые пишутся, непосредственно, на уровне разработчика (тестирование определённой сущности – метод или класс). Это крайне важный этап разработки ПО, который помогает создавать качественный продукт [9].

Для тестирования методов подсчета параметров для выкроек и методов проверки вводимых значений использовался JUnit.

Для тестирования методов создания объекта класса Equipment класс *TestCreateEquipmrnt.* Для проверки методов, которые проверяют вводимые значения был написан класс *TestValidator,* с двумя методами для проверки почты и чисел с плавающей точкой (рисунок 3.1).

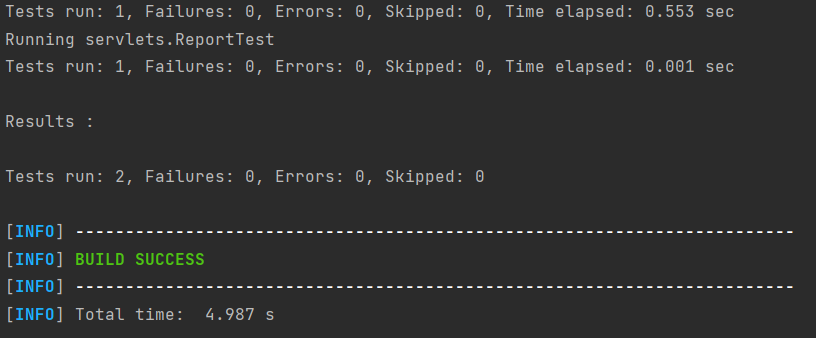


Рисунок 3.1 – Тестирование с помощью maven.

Для того чтобы убедиться, что программное средство может корректно выполнять работу и устойчиво к ошибкам пользователя было проведено ручное тестирование (manual testing), отражающие все возможные исключительные ситуации.

Во-первых, было рассмотрена возможность не корректной регистрации пользователя, а именно:

* попытка регистрации без указания логина;
* попытка регистрации без указания пароля;
* попытка регистрации с указанием уже существующего логина.

Программа реагирует на все три ситуации, выводится сообщение об ошибке (рисунки 3.2, 3.3, 3.4).

Далее была рассмотрена ситуация неправильной авторизации пользователя, а именно:

* попытка входа без указания логина;
* попытка входа без указания пароля;
* ввод неверных данных.

Программа реагирует на все ситуации аналогично реакции в форме для регистрации (рисунки 3.5)

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

Рисунок 3.2 – Ошибки при регистрации.

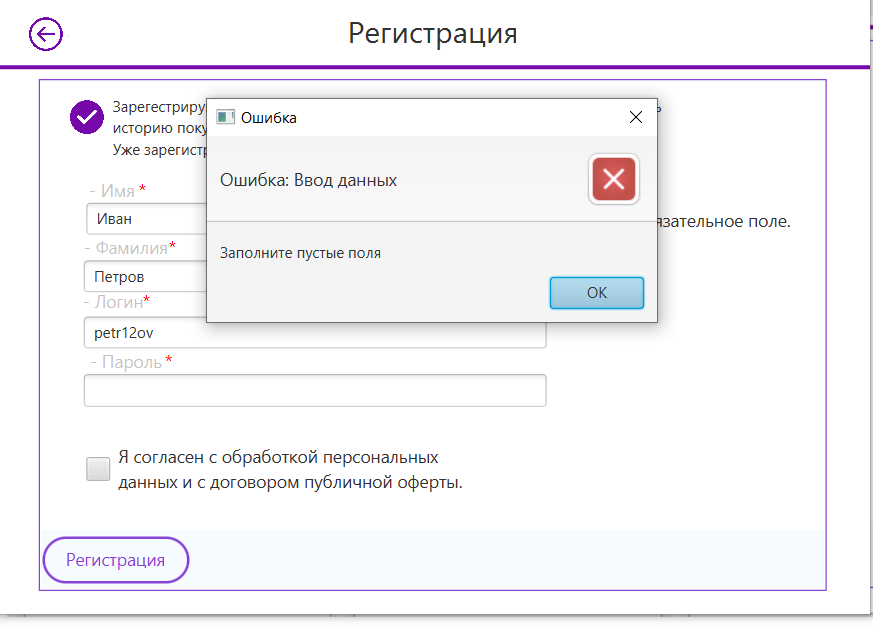


Рисунок 3.3 – Ошибки при регистрации.

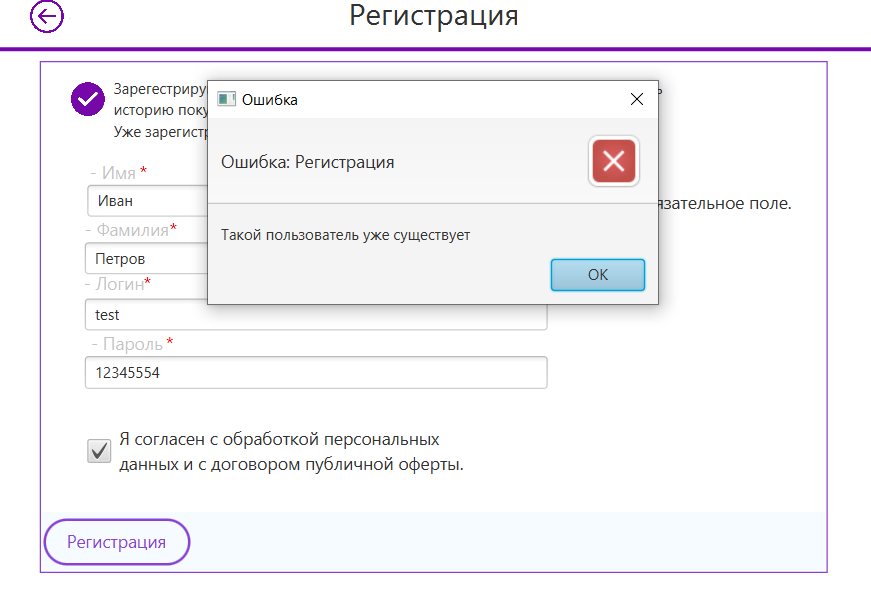


Рисунок 3.4 – Ошибки при регистрации.

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

Рисунок 3.5 – Ошибки при авторизации пользователя.

Далее была протестирован процесс добавления оборудования. Программа реагирует если в поля не вводятся значения или если значения не соответствуют типу запрашиваемых данных. (рисунок 3.7).

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

Рисунок 3.7 – Ошибки при не заполнении формы.

По результатам тестирования можно сделать вывод, что разработанное программное средство удовлетворяет функциональным требованиям и функции выполняются корректно.

# Руководство по развертыванию и использованию программного средства

Приложение является клиент-серверным и функционирует с базой данных MySQL. Для входа предусмотрены 2 роли: администратор и пользователь.

Для запуска серверной части необходимо установить jdk 18. А также обязательным является установка MySQL Workbench. Необходимо последовательно запустить server.jar, а затем client.jar и скрипт для создания базы данных.

Для запуска сервера необходимо в командной строке ввести команду java -jar Server.jar предварительно переместившись в каталог, в котором находится jar-архив. После чего можно запустить графическое приложение для работы клиента. Пример запуска показан на рисунке 4.1. Пример отображения состояния подключений показан на рисунке 4.2.



Рисунок 4.1 – Запуск сервера.

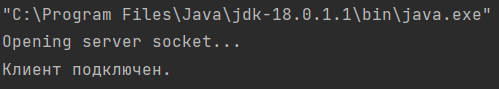


Рисунок 4.2 – Отображение состояний подключения к серверу.

При запуске клиентской части отображается меню входа и регистрации. Соответственно есть возможность пройти процесс авторизации как пользователь или как администратор, или если нет аккаунта пользователя, то зарегистрироваться (рисунок 4.3)

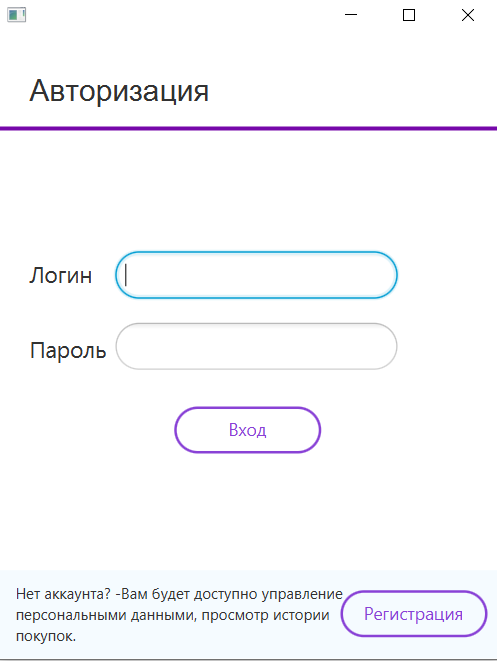


Рисунок 4.3 – Меню входа и регистрации.

Допустим у нас нет аккаунта, нажимаем кнопку “Регистрация”, после чего нам отобразиться форма для регистрации (рисунок 4.4). При успешной регистрации вы попадете в меню клиента.

Во всем приложении нажатие на кнопку «Назад», будет возвращать нас на предыдущую страницу.

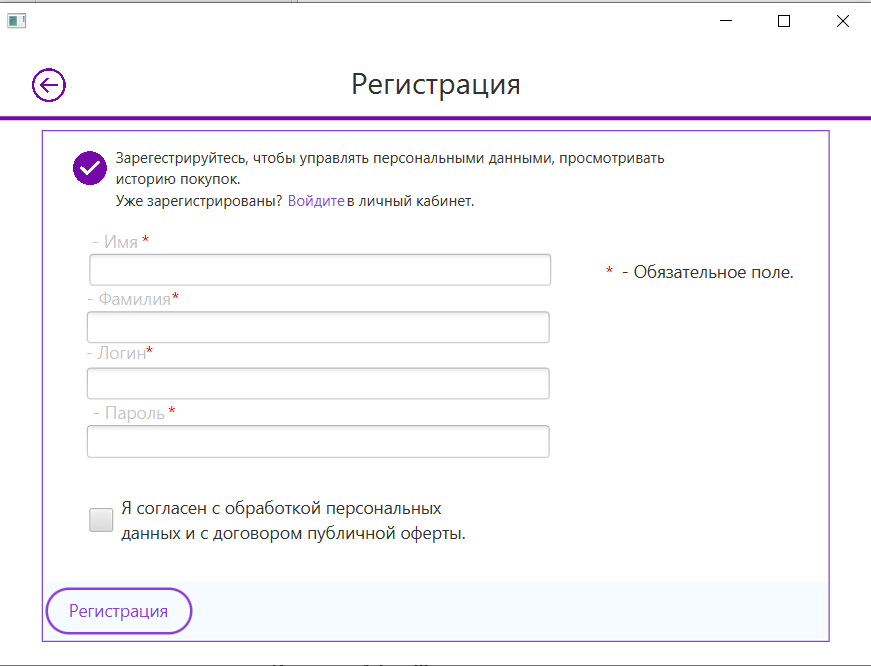


Рисунок 4.4 – Форма для регистрации.

**Действия клиента**

На меню пользователя можно увидеть все возможности пользователя (рисунок 4.5). Он может просмотреть каталог, просмотреть корзину, историю покупок или нажав на иконку пользователя открыть окно со своими данными (рисунок 4.6).

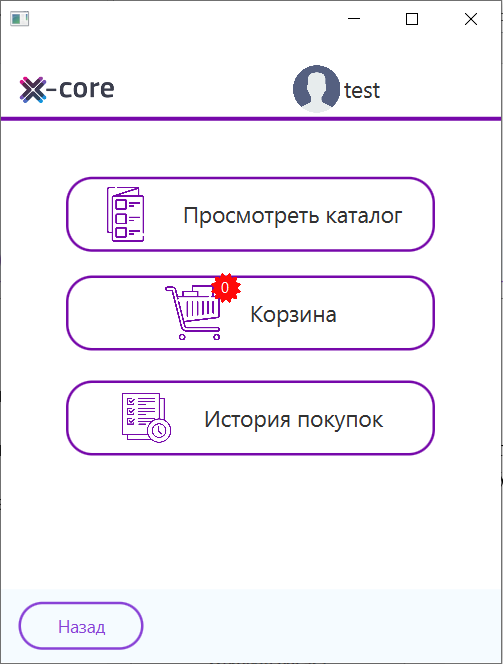


Рисунок 4.5 – Главное меню пользователя.

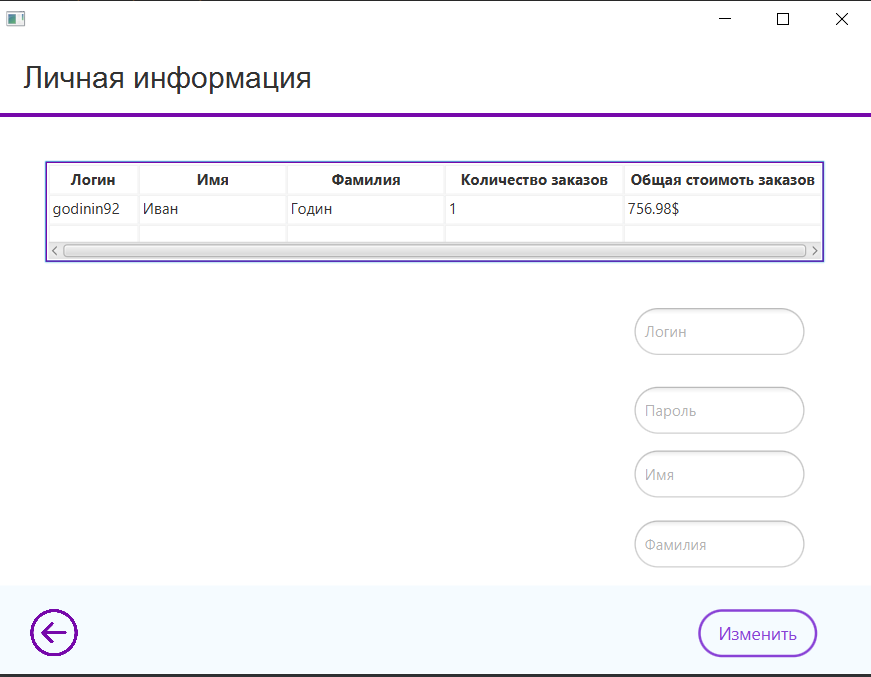


Рисунок 4.6 – Окно с данными пользователя.

Начнем с просмотра каталога, при нажатии на данную кнопку на экране появится каталог оборудования (рисунок 4.7). А также в этом окне можно добавить товар в корзину (рисунок 4.8).

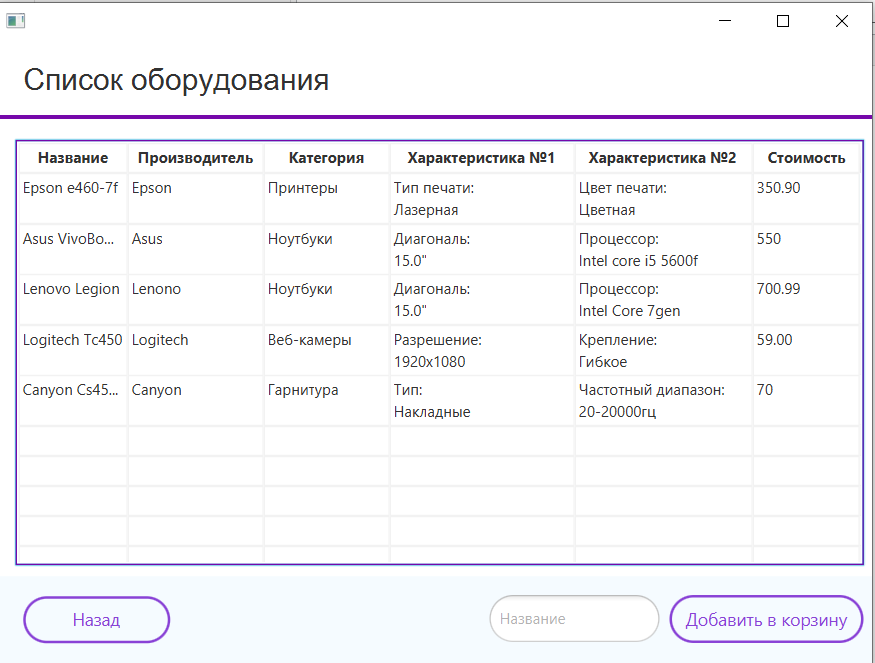


Рисунок 4.7 – Просмотр оборудования.

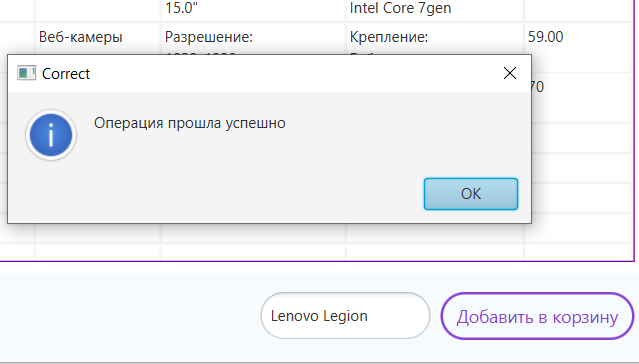


Рисунок 4.8 – Добавления оборудования в корзину.

После добавление товара на иконке корзины в меню пользователя появится число товаров в корзине (рисунок 4.9).

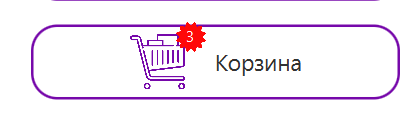


Рисунок 4.9 – Отображение количества товаров в корзине.

Теперь рассмотрим пункт меню «Корзина», в корзине можно просмотреть содержимое корзины и удалить товар из корзины (рисунок 4.10).

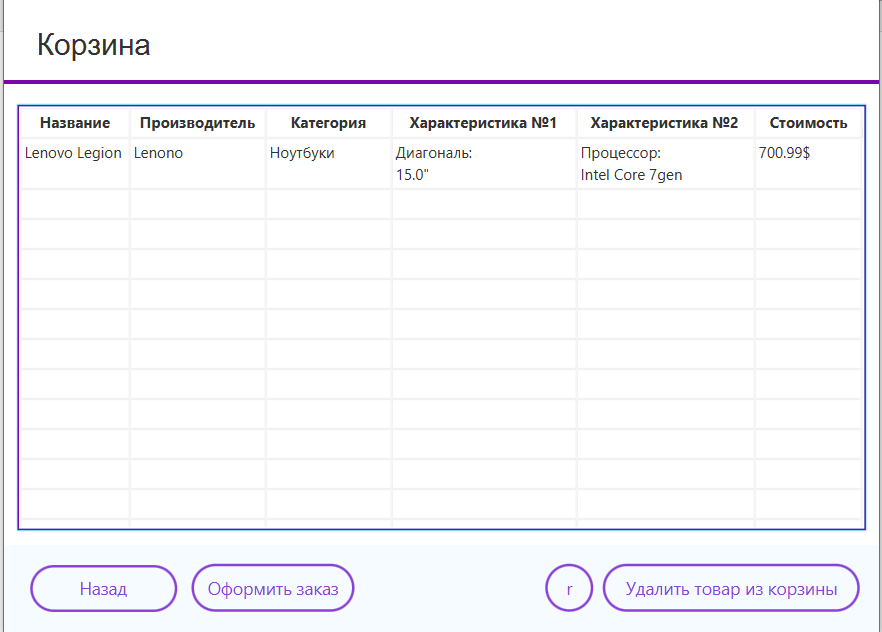


Рисунок 4.10– Просмотр корзины.

Оформим заказ для этого нажмем на кнопку «Оформить заказ» (рисунок 4.11).

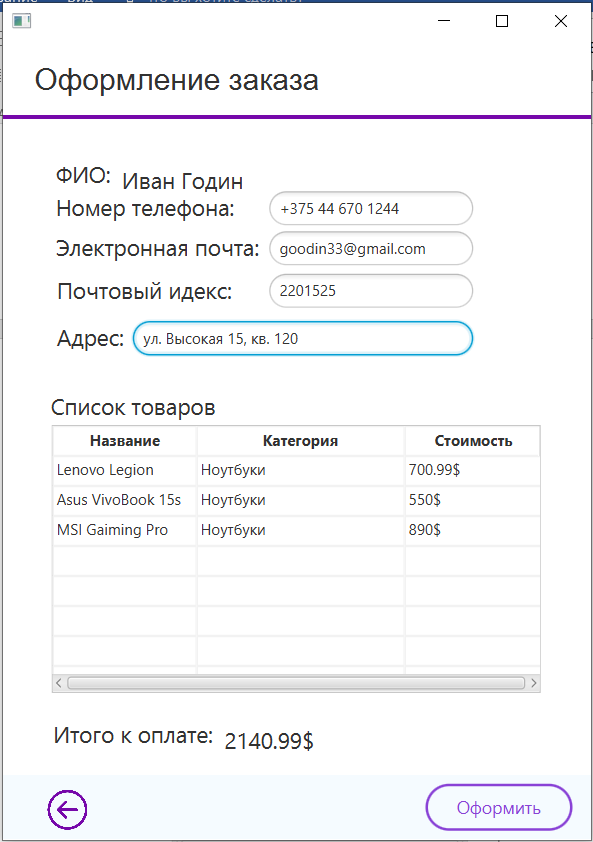


Рисунок 4.11 – Просмотр корзины.

Вернемся в меню пользователя и просмотрим историю покупок клиента пункт меню «История покупок» (рисунок 4.12).

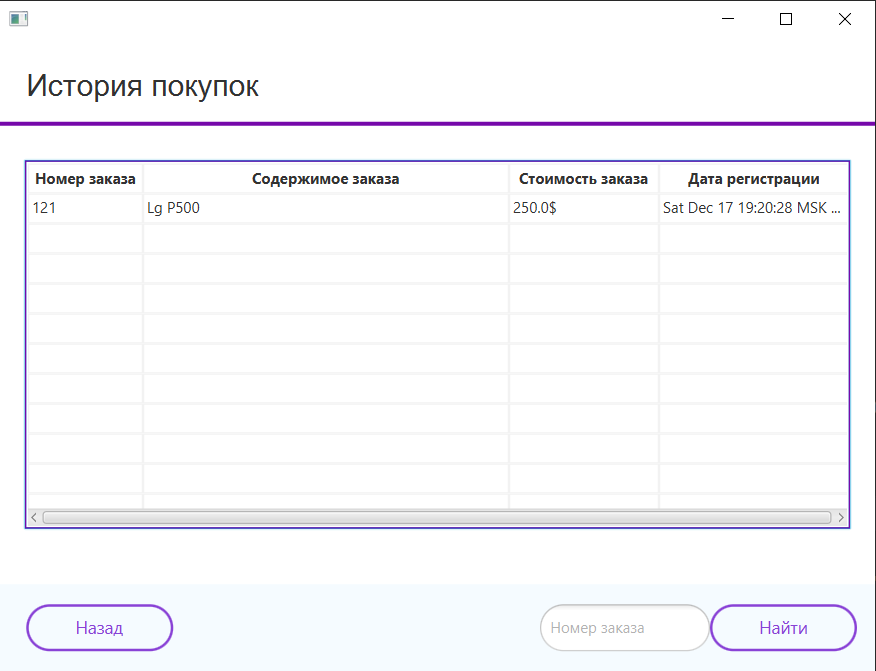


Рисунок 4.12– Просмотр корзины.

После описания всех функций пользователя перейдем к меню админа (рисунок 4.13). Так же, как и в меню пользователя после нажатия на иконку администратора откроется окно с информацией о пользователе. Рассмотрим первый пункт меню «Работа с клиентами» (рисунок 4.14).

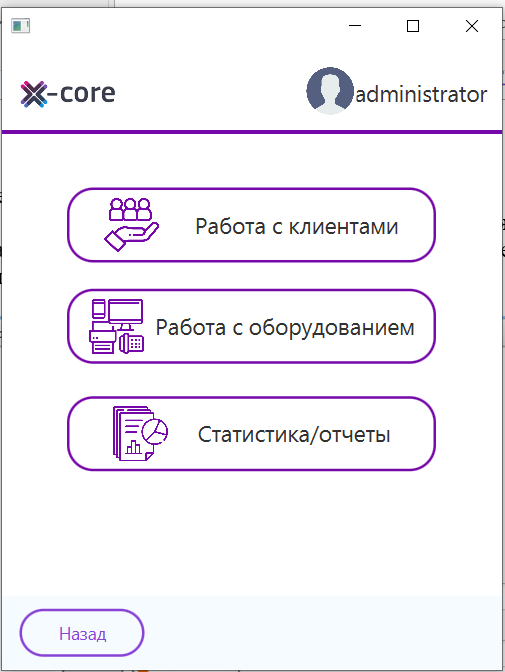


Рисунок 4.13– Меню администратора.

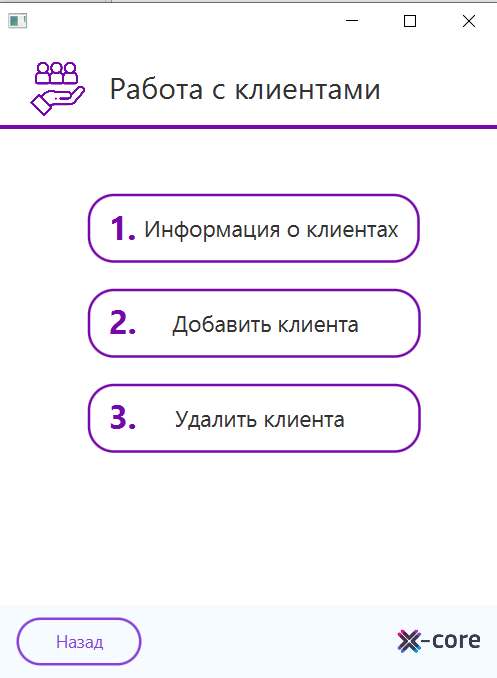


Рисунок 4.14 – Работа с клиентами.

Теперь рассмотрим пункт меню «Информация о клиентах», в открывшемся меню можно просмотреть информацию о клиентах, а также найти клиента по логину (рисунок 4.15).

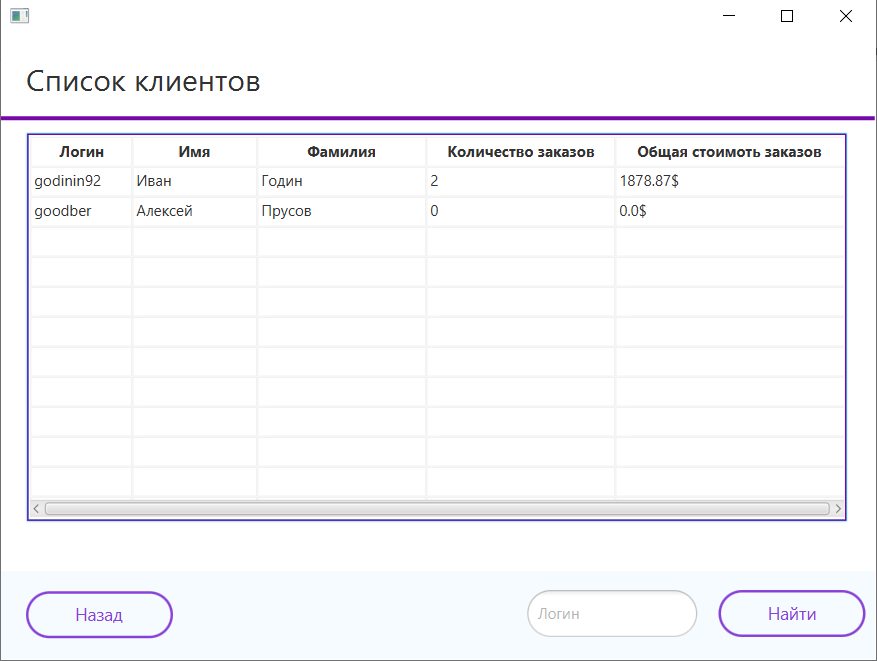


Рисунок 4.15 – Информация о клиентах

Вернемся в меню «Работа с клиентами» при нажатии на пункт меню 2 откроется окно регистрации (рисунок 4.4). Рассмотрим пункт меню 3 (рисунок 4.16).

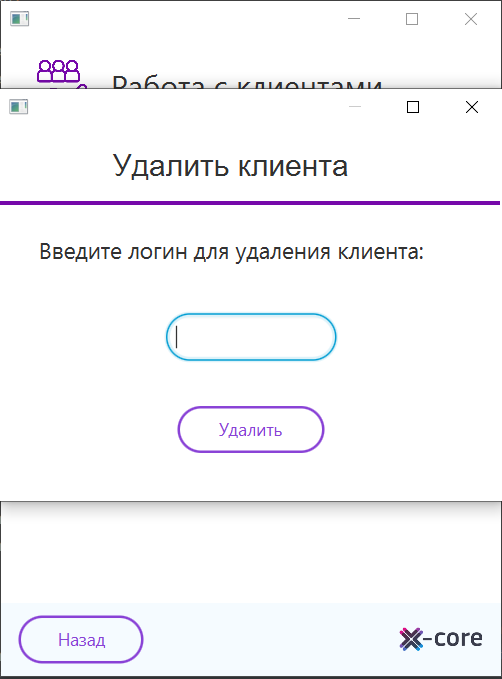


Рисунок 4.16 – Удаление клиента.

Меню администратора, выберем пункт меню «Работа с оборудованием» (рисунок 4.17).

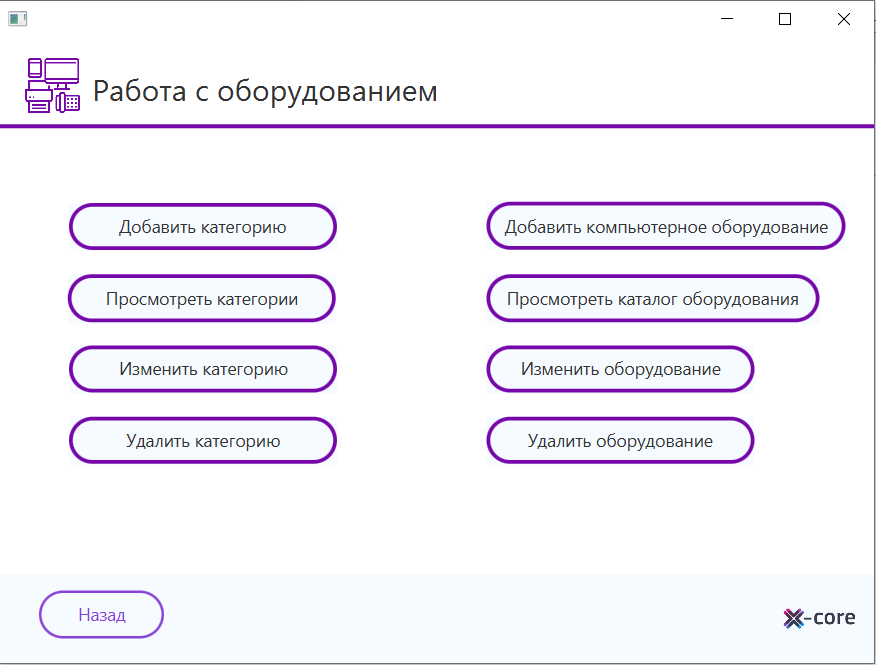


Рисунок 4.17 – Меню «Работа с оборудованием».

Добавим категорию, в появившимся окне необходимо заполнить все поля после нажать кнопку «Добавить категорию» (рисунок 4.18).

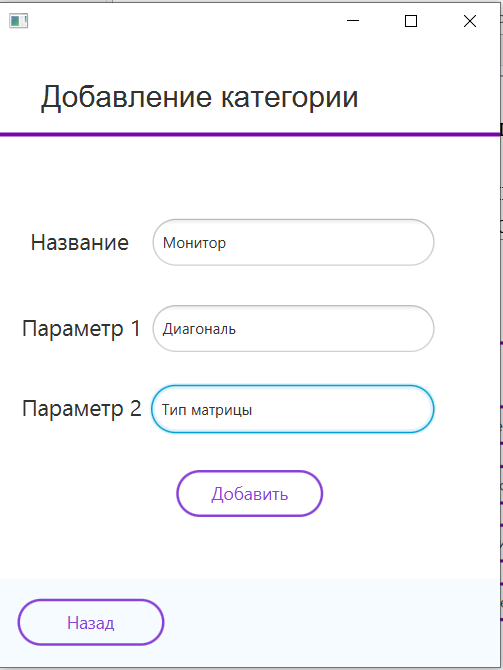


Рисунок 4.18 – Добавление категории.

Добавим оборудование, сперва нужно выбрать категорию (рисунок 4.19) после в появившимся окне необходимо заполнить все поля после нажать кнопку «Добавить» (рисунок 4.20).

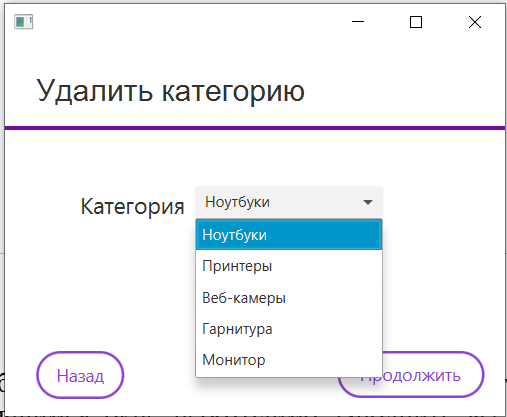


Рисунок 4.19 – Выбор категории.

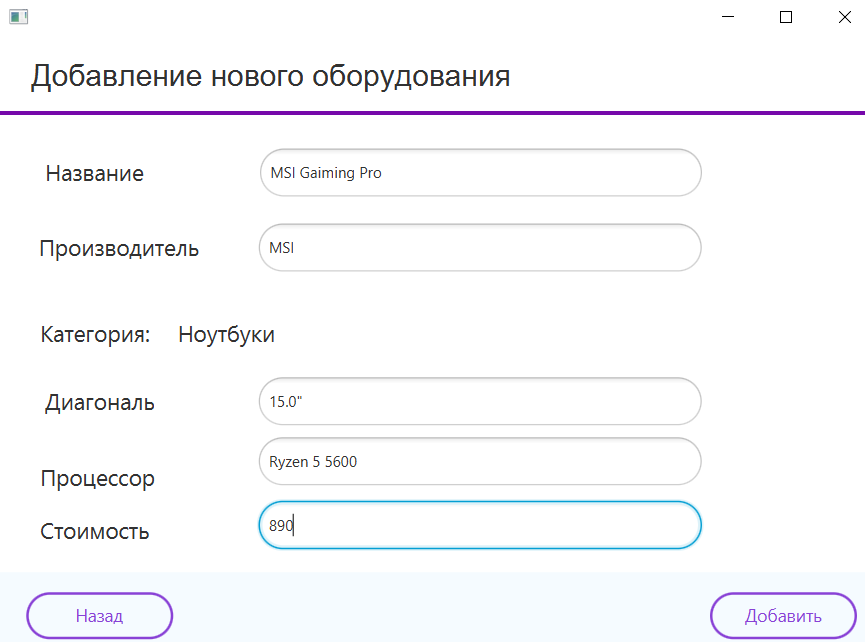


Рисунок 4.20 – Добавление оборудования.

Пункты меню «Изменить категорию», «Изменить оборудование» и «Удалить категорию», «Удалить оборудование» выполняются аналогично пунктам меню «Изменить клиента» (рисунок 4.5) и «Удалить клиента» (рисунок 4.16) соответственно.

Вернемся в меню администратора и выберем пункт меню «Статистика/отчеты (рисунок 4.21).

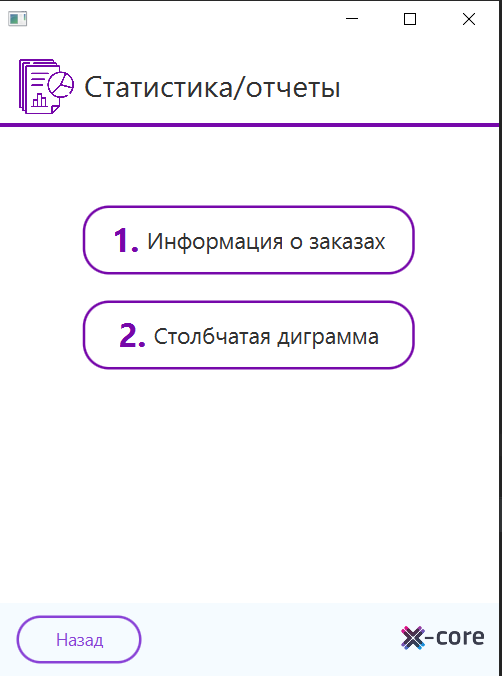


Рисунок 4.21 – Добавление оборудования.

Работа пунктов меню 1,2 продемонстрированы на рисунках 4.22 и 4.23 соответственно.

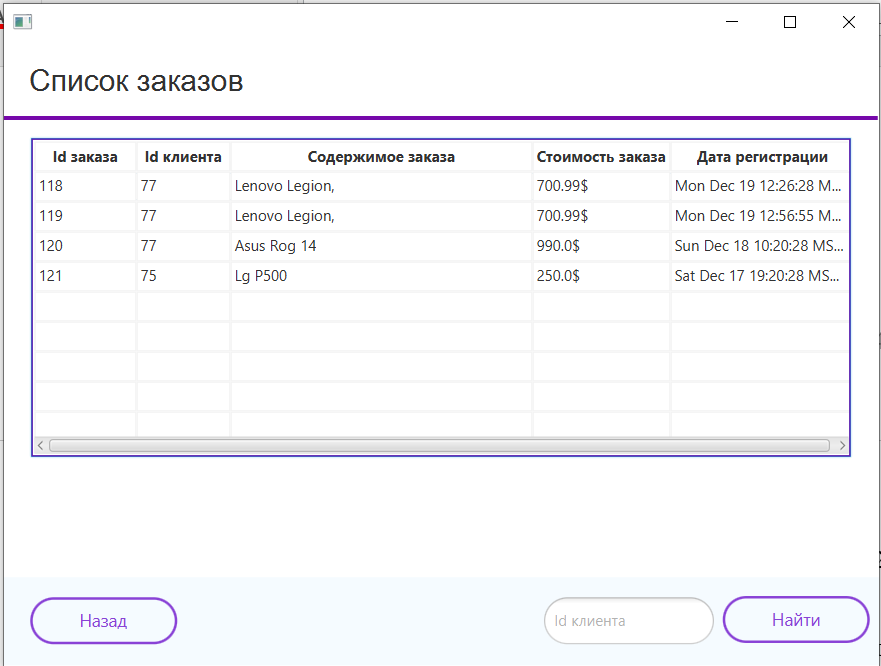


Рисунок 4.22 – Просмотр заказов.

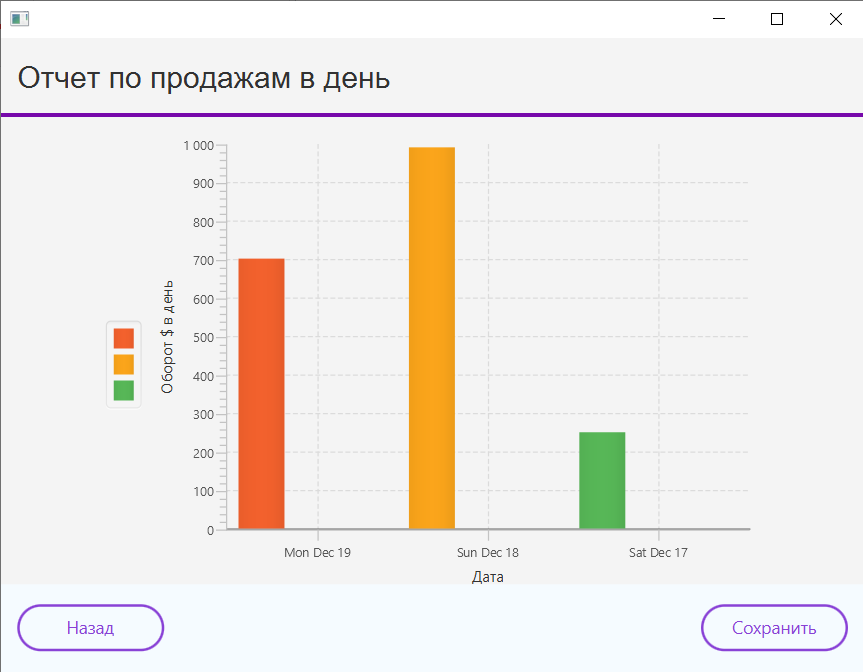


Рисунок 4.23 – Столбчатая диаграмма.

Таким образом, выполнены проектирование и разработка программного средства обеспечение работы магазина компьютерного оборудования. Выполнена постановка задачи и определены основные методы ее решения; в ходе объектного моделирования системы построен ряд *UML*-диаграмм; разработана информационная модель системы, представленная в виде схемы базы данных; описаны основные алгоритмы работы программного средства; разработано руководство пользователя; выполнено тестирование системы.

# Заключение

В результате выполнения данного курсового проекта было разработано программное приложение, позволяющие оптимизировать работу ремесленников и людей занимающихся вязанием как хобби. В нем предусмотрена система администрирования, а также проверка учетных данных при авторизации пользователей.

Приложение удовлетворяет основным характеристикам, которые были заявлены. Оно удобно в эксплуатации, целостно, конкретизировано в рамках заданной предметной области.

При разработке данного программного продукта была учтена логика пользователя, и интерфейс данной программы сделан удобным и понятным.

Достигнута основная цель курсового проекта, разработано программное средство, способное взять на себя все расчеты, составить необходимые выкройки и дать понятное пояснение к ним.

Использование данной программы при производстве вязаных изделий сделает работу автоматизированной, более быстрой и упрощенной, снизит до минимума ошибки в вычислениях (рассчет петельной пробы остается за пользователем). Облегчит способ хранения выкроек, больше не придется утопать в куче листков, определяя какая выкройка является итогоовой. На данный момент в программе реализовано очень мало выкроек, поэтому данная программа может редактироваться и совершенствоваться в соответствии с требованиями предметной области. Подводя итог, можно сказать, что цели и задачи, поставленные перед данной работой, успешно достигнуты и выполнены.

Выполнено проектирование и разработка автоматизированной системы составления выкроек для вязания «Crochet Assistent»: выполнена постановка задачи и определены основные методы ее решения; в ходе объектного моделирования системы построен ряд *UML*-диаграмм; разработана информационная модель системы, представленная в виде схемы базы данных; описаны основные алгоритмы работы программного средства; разработано руководство пользователя; выполнено тестирование системы, показавшее ее соответствие функциональным требованиям, поставленным в задании на разработку.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Технология производства трикотажных изделий [Электронный ресурс] – Режим доступа: http://www.trikotazha.net/stranitsyi-sayta/tehnologiya-proizvodstva-trikotazhnyih-izdeliy.html
2. ГОСТ 17521-72 Типовые фигуры мужчин. Размерные признаки для проектирования одежды [Электронный ресурс] – Режим доступа: <https://files.stroyinf.ru/Index2/1/4294835/4294835250.htm>
3. ГОСТ Р 50713-94 Изделия для новорожденных и детей ясельной группы. Общие технические условия [Электронный ресурс] – Режим доступа: <https://files.stroyinf.ru/Index2/1/4294819/4294819375.htm>
4. ГОСТ 17916-86 Типовые фигуры девочек. Размерные признаки для проектирования одежды [Электронный ресурс] – Режим доступа: <https://files.stroyinf.ru/Data/75/7514.pdf>

Приложение А  
(обязательное)  
Отчет о проверке на заимствование в системе «Антиплагиат»

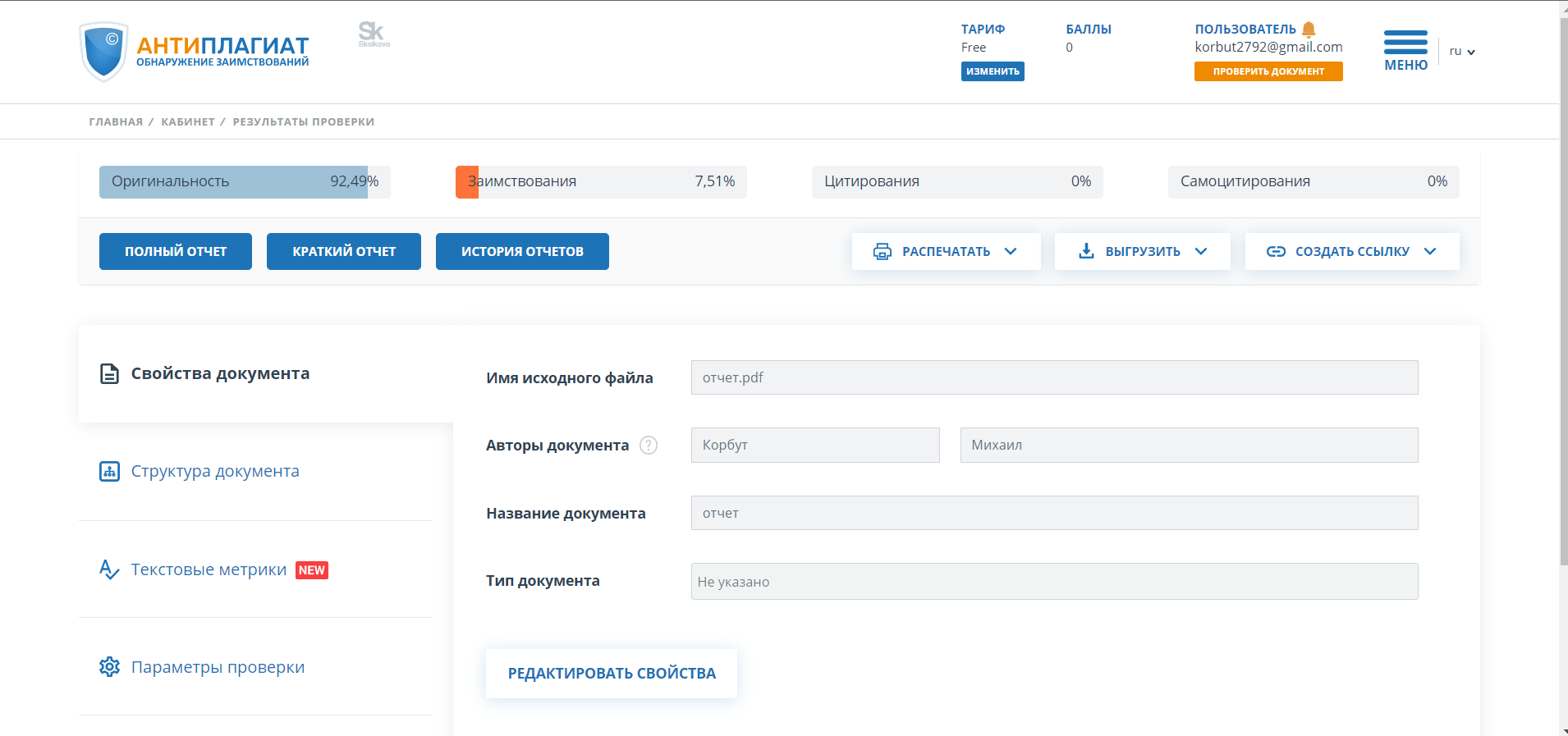


Рисунок А.1 – Отчет о проверке на заимствования в системе «Антиплагиат»

**Приложение Б**(обязательное)  
Листинг кода алгоритмов, реализующих основную бизнес-логику

// Класс ConnectionDB. Отвечает за подключение к БД

package DB;

import java.sql.\*;

import java.util.ArrayList;

public class ConnectionDB {

private static ConnectionDB instance;

private String dbHost = "localhost";

private String dbPort = "3306";

private String dbUser = "root";

private String dbPass = "12345";

private String dbName = "computerequipmentstore";

ArrayList<String[]> masResult;

public static Connection dbConnection;

private Statement statement;

private ResultSet resultSet;

public ConnectionDB()

throws ClassNotFoundException, SQLException {

String connectionString = "jdbc:mysql://" + dbHost + ":" + dbPort + "/" + dbName + "?verifyServerCertificate=false"+

"&useSSL=false"+

"&requireSSL=false"+

"&useLegacyDatetimeCode=false"+

"&amp"+

"&serverTimezone=UTC";

Class.forName("com.mysql.cj.jdbc.Driver");

dbConnection = DriverManager.getConnection(connectionString, dbUser, dbPass);

statement = dbConnection.createStatement();

}

public void execute(String query) {

try {

statement.execute(query);

} catch (SQLException e) {

e.printStackTrace();

}

}

public static synchronized ConnectionDB getInstance() throws SQLException, ClassNotFoundException {

if (instance == null) {

instance = new ConnectionDB();

}

return instance;

}

public ArrayList<String[]> getArrayResult(String str) {

masResult = new ArrayList<String[]>();

try {

resultSet = statement.executeQuery(str);

int count = resultSet.getMetaData().getColumnCount();

while (resultSet.next()) {

String[] arrayString = new String[count];

for (int i = 1; i <= count; i++)

arrayString[i - 1] = resultSet.getString(i);

masResult.add(arrayString);

}

}catch(SQLException e){

e.printStackTrace();

}

return masResult;

}

}

// Класс Worker и несколько кейсов отвечающих за получение команд от клиента и запуска обработки запроса.

protected Socket clientSocket = null;

ObjectInputStream sois;

ObjectOutputStream soos;

public Worker(Socket clientSocket) {

this.clientSocket = clientSocket;

}

@Override

public void run() {

try {

sois = new ObjectInputStream(clientSocket.getInputStream());

soos = new ObjectOutputStream(clientSocket.getOutputStream());

while (true) {

System.out.println("Получение команды от клиента...");

String choice = sois.readObject().toString();

System.out.println(choice);

System.out.println("Команда получена");

switch (choice) {

case "authorization" -> {

System.out.println("Выполняется авторизация пользователя....");

Authorization auth = (Authorization) sois.readObject();

System.out.println(auth.toString());

SQLFactory sqlFactory = new SQLFactory();

Role r = sqlFactory.getRole().getRole(auth);

System.out.println(r.toString());

if (r.getId() != 0 && r.getRole() != "" && sqlFactory.deleteBasket()) {

soos.writeObject("OK");

soos.writeObject(r);

} else

soos.writeObject("There is no data!");

}

case "adminInf" -> {

System.out.println("Запрос к БД на получение информации об администраторе: " + clientSocket.getInetAddress().toString());

SQLFactory sqlFactory = new SQLFactory();

ArrayList<Admin> infList = sqlFactory.getAdmin().get();

System.out.println(infList.toString());

soos.writeObject(infList);

}

// Класс EquipmentSQL. C листингом кода основных методов таких, как добавление в бд, вызов процедур и т.д.

import ComputerEquipmentStore.BasketItem;

import ComputerEquipmentStore.Category;

import ComputerEquipmentStore.Equipment;

import ComputerEquipmentStore.Role;

import java.sql.CallableStatement;

import java.sql.SQLException;

import java.sql.SQLIntegrityConstraintViolationException;

import java.util.ArrayList;

import java.util.Date;

public class SQLEquipment implements ISQLEquipment {

private static SQLEquipment instance;

private ConnectionDB dbConnection;

private SQLEquipment() throws SQLException, ClassNotFoundException {

dbConnection = ConnectionDB.getInstance();

}

public static synchronized SQLEquipment getInstance() throws SQLException, ClassNotFoundException {

if (instance == null) {

instance = new SQLEquipment();

}

return instance;

}

@Override

public ArrayList<Equipment> findEquipment(Equipment obj) {

String str = "select name, producer, category, firstParameter, secondParameter, price\n" +

"from equipment"+ " where `equipment`.name = \"" + obj.getName() + "\";";

ArrayList<String[]> result = dbConnection.getArrayResult(str);

ArrayList<Equipment> equipmentist = new ArrayList<>();

for (String[] items: result){

Equipment equipment = new Equipment();

equipment.setName(items[0]);

equipment.setProducer(items[1]);

equipment.setCategory(items[2]);

equipment.setFirst\_Parameter(items[3]);

equipment.setSecond\_Parameter(items[4]);

equipment.setPrice(items[5]);

equipmentist.add(equipment);

}

return equipmentist;

}

@Override

public ArrayList<BasketItem> findEquipmentForBasket(Equipment obj) {

String str = "select name, category, price\n" +

"from equipment" + " where `equipment`.name = \"" + obj.getName() + "\";";

ArrayList<String[]> result = dbConnection.getArrayResult(str);

ArrayList<BasketItem> equipmentist = new ArrayList<>();

for (String[] items: result){

BasketItem equipment = new BasketItem();

equipment.setName(items[0]);

equipment.setCategory(items[1]);

equipment.setPrice(items[2]);

equipmentist.add(equipment);

}

return equipmentist;

}

@Override

public boolean deleteEquipment(Equipment obj) {

String proc = "{call delete\_equipment(?)}";

try (CallableStatement callableStatement = ConnectionDB.dbConnection.prepareCall(proc)) {

callableStatement.setString(1, obj.getName());

callableStatement.execute();

} catch (SQLIntegrityConstraintViolationException e) {

System.out.println("ошибка");

return false;

} catch (Exception e) {

e.printStackTrace();

return false;

}

return true;

}

@Override

public boolean deleteEquipmentFromBasket(Equipment obj) {

String proc = "{call delete\_EquipmnetFromBasket(?)}";

try (CallableStatement callableStatement = ConnectionDB.dbConnection.prepareCall(proc)) {

callableStatement.setString(1, obj.getName());

callableStatement.execute();

} catch (SQLIntegrityConstraintViolationException e) {

System.out.println("ошибка");

return false;

} catch (Exception e) {

e.printStackTrace();

return false;

}

return true;

}

@Override

public ArrayList<Equipment> get() {

String str = "select name, producer, category, firstParameter, secondParameter, price\n" +

"from equipment;";

ArrayList<String[]> result = dbConnection.getArrayResult(str);

ArrayList<Equipment> infList = new ArrayList<>();

for (String[] items: result){

Equipment equipment = new Equipment();

equipment.setName(items[0]);

equipment.setProducer(items[1]);

equipment.setCategory(items[2]);

equipment.setFirst\_Parameter(items[3]);

equipment.setSecond\_Parameter(items[4]);

equipment.setPrice(items[5]);

infList.add(equipment);

}

return infList;

}

@Override

public ArrayList<Equipment> getBasket() {

String str = "select name, producer, category, parameter1, parameter2, price\n" +

"from basket;";

ArrayList<String[]> result = dbConnection.getArrayResult(str);

ArrayList<Equipment> infList = new ArrayList<>();

for (String[] items: result){

Equipment equipment = new Equipment();

equipment.setName(items[0]);

equipment.setProducer(items[1]);

equipment.setCategory(items[2]);

equipment.setFirst\_Parameter(items[3]);

equipment.setSecond\_Parameter(items[4]);

equipment.setPrice(items[5]);

infList.add(equipment);

}

return infList;

}

@Override

public Equipment getEquipment(Role r) {

String str = "select name, category, price\n" +

"from equipment;";

ArrayList<String[]> result = dbConnection.getArrayResult(str);

Equipment equipment = new Equipment();

for (String[] items: result){

equipment.setName(items[0]);

equipment.setCategory(items[1]);

equipment.setPrice(items[2]);

}

return equipment;

}

@Override

public boolean insert(Equipment obj, Category ct) {

String proc = "{call insert\_equipment(?,?,?, ?, ?, ?)}";

try (CallableStatement callableStatement = ConnectionDB.dbConnection.prepareCall(proc)) {

System.out.println(obj.toString());

callableStatement.setString(1, obj.getName());

callableStatement.setString(2, obj.getProducer());

callableStatement.setString(3, obj.getCategory());

callableStatement.setString(4, ct.getParameter\_1() + ":\n" + obj.getFirst\_Parameter());

callableStatement.setString(5, ct.getParameter\_2() + ":\n" + obj.getSecond\_Parameter());

callableStatement.setString(6, obj.getPrice());

callableStatement.execute();

} catch (SQLIntegrityConstraintViolationException e) {

System.out.println("ошибка");

return false;

} catch (Exception e) {

e.printStackTrace();

return false;

}

return true;

}

@Override

public boolean insertBasket(Equipment obj) {

String proc = "{call insert\_equipment\_to\_basket(?,?,?,?,?,?)}";

try (CallableStatement callableStatement = ConnectionDB.dbConnection.prepareCall(proc)) {

callableStatement.setString(1, obj.getName());

callableStatement.setString(2, obj.getProducer());

callableStatement.setString(3, obj.getCategory());

callableStatement.setString(4, obj.getFirst\_Parameter());

callableStatement.setString(5, obj.getSecond\_Parameter());

callableStatement.setString(6, obj.getPrice());

callableStatement.execute();

} catch (SQLIntegrityConstraintViolationException e) {

System.out.println("ошибка");

return false;

} catch (Exception e) {

e.printStackTrace();

return false;

}

return true;

}

@Override

public boolean addEquipmentsToOrder(int id\_user) {

String proc = "{call make\_order(?, ?, ?, ?)}";

String str = "select name, category, price\n" +

"from basket;";

ArrayList<String[]> result = dbConnection.getArrayResult(str);

String names = "";

double sumprice = 0;

for (String[] items: result){

double i3 = 0;

names += items[0] + ", ";

try {

i3 = Double.parseDouble(items[2]);

sumprice += i3;

} catch (NumberFormatException e) {

System.err.println("Неправильный формат строки!");

}

}

Date date = new Date();

try (CallableStatement callableStatement = ConnectionDB.dbConnection.prepareCall(proc)) {

System.out.println("Order{equipments='" + names + "', sumprice='100'}");

callableStatement.setInt(1, id\_user);

callableStatement.setString(2, names);

callableStatement.setDouble(3, sumprice);

callableStatement.setString(4, date.toString());

callableStatement.execute();

} catch (SQLIntegrityConstraintViolationException e) {

System.out.println("ошибка");

return false;

} catch (Exception e) {

e.printStackTrace();

return false;

}

return true;

}

}

// Класс MenuAdminController показывает основные принципы работы контроллера.

package controllers;

public class MenuAdminController {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private Button backButton;

@FXML

private Button dbWorkButton;

@FXML

private Button clientWorkButton;

@FXML

private Button statisticWorkButton;

@FXML

private Button personalInfButton;

@FXML

void statisticWork(ActionEvent event) throws IOException {

statisticWorkButton.getScene().getWindow().hide();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("/statisticWork.fxml"));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

Stage stage = new Stage();

stage.setScene(new Scene((root)));

stage.show();

}

@FXML

void persInf(ActionEvent event) throws IOException {

Connect.client.sendMessage("adminInf");

WindowChanger.changeWindow(getClass(), personalInfButton, "adminInformation.fxml", "", false);

}

@FXML

void backToMain(ActionEvent event) {

backButton.getScene().getWindow().hide();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("/main.fxml"));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

Stage stage = new Stage();

stage.setScene(new Scene((root)));

stage.show();

}

}

// Методы initialize и getCategory() отвечающие за добавление записей в xml таблицу. Метод initialize переопределяет метод интерфеся Initializable.

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

nameColumn.setCellValueFactory(field -> new SimpleObjectProperty<>(field.getValue().getName()));

parameter1Column.setCellValueFactory(field -> new SimpleObjectProperty<>(field.getValue().getParameter\_1()));

parameter2Column.setCellValueFactory(field -> new SimpleObjectProperty<>(field.getValue().getParameter\_2()));

categoryTable.setItems(getCategory());

}

private ObservableList<Category> getCategory() {

ObservableList<Category> categoryList = FXCollections.observableArrayList();

ArrayList<Category> category = (ArrayList<Category>) Connect.client.readObject();

System.out.println(category);

categoryList.addAll(category);

categoryTable.setItems(categoryList);

return categoryList;

}

// Метод реализующий смену окон.

public static void changeWindow(Сlass className, Button button, String fname, String title, boolean ismodal) throws IOException {

FXMLLoader fxmlLoader = new FXMLLoader();

fxmlLoader.setLocation(className.getResource("/" + fname));

fxmlLoader.load();

Parent root = fxmlLoader.getRoot();

Stage stage = new Stage();

stage.setTitle(title);

stage.setScene(new Scene(root));

if (ismodal) {

stage.initModality(Modality.APPLICATION\_MODAL);

}

else {

button.getScene().getWindow().hide();

}

stage.show();

}

// Класс Check, методы этого класса проверяют вводимые значения на соответствие запрашиваемому типу данных.

public class Check {

public static boolean checkInt(String str) {

Pattern r = Pattern.compile("[\\\\d]+");

Matcher m = r.matcher(str);

return m.matches();

}

public static boolean checkDouble(String str) {

Pattern r = Pattern.compile("[+-]?([0-9]\*[.])?[0-9]+");

Matcher m = r.matcher(str);

return m.matches();

}

public static boolean checkString(String str) {

Pattern r = Pattern.compile("[a-zA-Z]+");

Matcher m = r.matcher(str);

return m.matches();

}

}

// Класс Client.

public class Client {

private Socket clientSocket;

private ObjectOutputStream outStream;

private ObjectInputStream inStream;

private String message;

public Client(String ipAddress, String port){

try {

clientSocket = new Socket(ipAddress, Integer.parseInt(port));

outStream = new ObjectOutputStream(clientSocket.getOutputStream());

inStream = new ObjectInputStream(clientSocket.getInputStream());

} catch (IOException e) {

System.out.println("Server not found: " + e.getMessage());

System.exit(0);

}

}

public void sendMessage(String message){

try {

outStream.writeObject(message);

} catch (IOException e) {

e.printStackTrace();

}

}

public void sendObject(Object object){

try {

outStream.writeObject(object);

} catch (IOException e) {

e.printStackTrace();

}

}

public String readMessage() throws IOException {

try {

message = (String) inStream.readObject();

} catch (ClassNotFoundException | IOException e) {

e.printStackTrace();

}

return message;

}

public Object readObject(){

Object object = new Object();

try {

object = inStream.readObject();

} catch (ClassNotFoundException | IOException e) {

e.printStackTrace();

}

return object;

}

public void close() {

try {

clientSocket.close();

//outStream.flush();

inStream.close();

outStream.close();

} catch (EOFException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Приложение В**  
(обязательное)  
Листинг скрипта генерации базы данных

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema computerequipmentstore

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `foreignlanguageschool` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci ;

USE `fcomputerequipmentstore` ;

CREATE TABLE `admins` (

`idadmins` int NOT NULL AUTO\_INCREMENT,

`id\_keys` int NOT NULL,

PRIMARY KEY (`idadmins`),

UNIQUE KEY `id\_keys\_UNIQUE` (`id\_keys`),

CONSTRAINT `fk\_admins\_keys` FOREIGN KEY (`id\_keys`) REFERENCES `keys` (`id\_keys`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `basket` (

`idequipment` int NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`producer` varchar(45) NOT NULL,

`category` varchar(45) NOT NULL,

`parameter1` varchar(45) NOT NULL,

`parameter2` varchar(45) NOT NULL,

`price` varchar(45) NOT NULL,

PRIMARY KEY (`idequipment`),

UNIQUE KEY `name\_UNIQUE` (`name`)

) ENGINE=InnoDB AUTO\_INCREMENT=146 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `categories` (

`idcategory` int NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`parameter1` varchar(45) NOT NULL,

`parameter2` varchar(45) NOT NULL,

PRIMARY KEY (`idcategory`),

UNIQUE KEY `name\_UNIQUE` (`name`)

) ENGINE=InnoDB AUTO\_INCREMENT=93 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `client` (

`idclient` int NOT NULL AUTO\_INCREMENT,

`firstname` varchar(45) NOT NULL,

`lastname` varchar(45) NOT NULL,

`id\_keys` int NOT NULL,

`orders\_amount` int DEFAULT '0',

`total\_spent` double DEFAULT '0',

PRIMARY KEY (`idclient`),

UNIQUE KEY `id\_keys\_UNIQUE` (`id\_keys`),

KEY `fk\_clients\_keys\_idx` (`id\_keys`),

CONSTRAINT `fk\_clients\_keys` FOREIGN KEY (`id\_keys`) REFERENCES `keys` (`id\_keys`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=47 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `equipment` (

`idequipment` int NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`producer` varchar(45) NOT NULL,

`category` varchar(45) NOT NULL,

`firstParameter` varchar(45) NOT NULL,

`secondParameter` varchar(45) NOT NULL,

`price` varchar(45) NOT NULL,

PRIMARY KEY (`idequipment`),

UNIQUE KEY `name\_UNIQUE` (`name`)

) ENGINE=InnoDB AUTO\_INCREMENT=101 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `keys` (

`id\_keys` int NOT NULL AUTO\_INCREMENT,

`login` varchar(45) NOT NULL,

`password` varchar(45) NOT NULL,

PRIMARY KEY (`id\_keys`),

UNIQUE KEY `login\_UNIQUE` (`login`)

) ENGINE=InnoDB AUTO\_INCREMENT=78 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `orders` (

`idorder` int NOT NULL AUTO\_INCREMENT,

`iduser` int NOT NULL,

`contents` varchar(45) NOT NULL,

`sumprice` double DEFAULT NULL,

`date` varchar(45) DEFAULT NULL,

PRIMARY KEY (`idorder`)

) ENGINE=InnoDB AUTO\_INCREMENT=122 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `delete\_basket`()

BEGIN

DELETE FROM basket;

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `delete\_category`(name\_ varchar(45))

BEGIN

delete from `categories`

where name = name\_;

delete from `equipment`

where category = name\_;

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `delete\_client`(login\_ varchar(45))

BEGIN

select `id\_keys` into @id\_k from `keys` where `login` = login\_;

delete from `keys`

where id\_keys = @id\_k;

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `delete\_equipment`(name\_ varchar(45))

BEGIN

delete from `equipment`

where name = name\_;

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `delete\_EquipmnetFromBasket`(name\_ varchar(45))

BEGIN

delete from `basket`

where name = name\_;

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `find\_login`(uslogin VARCHAR(45), uspass VARCHAR(45), OUT id\_user INT, OUT usrole VARCHAR(10))

BEGIN

SET id\_user = 0;

SET usrole = "";

SELECT id\_keys INTO id\_user FROM `keys`

WHERE `login` = uslogin AND `password` = uspass;

SELECT COALESCE(ur, "") into usrole

FROM ( select "client" as ur from `client` where id\_keys = id\_user

union

select "admin" as ur from `admins` where id\_keys = id\_user

) as T;

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `get\_admin`()

BEGIN

select `keys`.login, `keys`.`password` from admins

join `keys` on `keys`.id\_keys = admins.id\_keys;

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `insert\_admin`(login varchar(45), pass varchar(45))

BEGIN

CALL insert\_keys(login, pass, @id\_keys);

INSERT INTO `computerequipmentstore`.`admins` (`id\_keys`)

VALUES (@id\_keys);

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `insert\_equipment`(name varchar(45), producer varchar(45), category varchar(45), parameter1 varchar(45), parameter2 varchar(45), price varchar(45))

BEGIN

insert into `computerequipmentstore`.`equipment` (`name`, `producer`, `category`, `firstParameter`, `secondParameter`, price)

VALUES (name, producer, category, parameter1, parameter2, price);

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `make\_order`(iduser int, contents varchar(45), sumprice double, dat varchar(45))

BEGIN

insert into `computerequipmentstore`.`orders` (iduser, contents, sumprice, date)

VALUES (iduser, contents, sumprice, dat);

UPDATE `computerequipmentstore`.`client`

SET `total\_spent` = sumprice + total\_spent

WHERE id\_keys = iduser;

UPDATE `computerequipmentstore`.`client`

SET `orders\_amount` = orders\_amount + 1

WHERE id\_keys = iduser;

END$$

DELIMITER ;

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `insert\_keys`(login varchar(45), pass varchar(45), out id\_keys int)

BEGIN

insert into `computerequipmentstore`.`keys`(`login`, `password`)

values (login, pass);

select last\_insert\_id() into id\_keys;

END$$

DELIMITER ;