СОДЕРЖАНИЕ

[Введение 8](#_Toc121580518)

[1 Анализ и моделирование предметной области программного средства 9](#_Toc121580519)

[1.1 Описание предметной области 9](#_Toc121580520)

[1.2 Разработка функциональной модели предметной области 12](#_Toc121580521)

[1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 17](#_Toc121580522)

[1.4 Разработка информационной модели предметной области 19](#_Toc121580523)

[1.5 Модели представления программного средства и их описание 21](#_Toc121580524)

[1.5.1 Диаграмма последовательности 21](#_Toc121580525)

[1.5.2 Диаграмма деятельности 23](#_Toc121580526)

[1.5.3 Диаграмма развертывания 24](#_Toc121580527)

[2 Проектирование и конструирование программного средства 26](#_Toc121580528)

[2.1 Постановка задачи 26](#_Toc121580529)

[2.2 Архитектурные решения 27](#_Toc121580530)

[2.3 Описание алгоритмов, реализующих бизнес-логику разрабатываемого программного средства 30](#_Toc121580531)

[2.4 Проектирование пользовательского интерфейса 32](#_Toc121580532)

[2.5 Обоснование выбора компонентов и технологий для реализации программного средства 33](#_Toc121580533)

[3 Тестирование и проверка работоспособности программного средства 35](#_Toc121580534)

[4 Руководство по развертыванию и использованию программного средства ………………………………………………………………………………….39](#_Toc121580535)

[Заключение 56](#_Toc121580536)

[Список использованных источников 57](#_Toc121580537)

[Приложение А (обязательное) Отчет о проверке на заимствование в системе «Антиплагиат» 58](#_Toc121580538)

[Приложение Б (обязательное) Листинг кода алгоритмов, реализующих основную бизнес-логику 59](#_Toc121580539)

[Приложение В (обязательное) Листинг скрипта генерации базы данных 70](#_Toc121580540)

Ведомость документов курсового проекта…………………………………….72

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**

Данный процесс делится на несколько подпроцессов:

* выбор компьютерной техники;
* оформление покупки;
* оплата товаров;
* выдача товаров.

Изготовление швейных изделий предполагает раскрой ткани по лекалам и последующий пошив. В трикотажном производстве выбор способов изготовления более широк.

Получить трикотажное изделие заданной формы можно четырьмя способами: кроеным, регулярным, полурегулярным и цельновязанным. В первом случае трикотажное полотно раскраивают по лекалам так же, как ткань, учитывая направление петельных столбиков, расположение рисунка и т. д. В основном этот способ используется для производства тонких изделий (футболки). При регулярном способе детали вывязываются по контуру аналогично ручному вязанию. Полурегулярный способ - нечто среднее между кроеным и регулярным: вяжутся купоны простейших геометрических форм с последующим подкроем, этот способ в основном используется при производстве дешевой массовой продукции и спецодежды. Большинство трикотажных производств России, Турции и Китая производят трикотаж именно полурегулярным методом. Цельновязанный способ позволяет вывязывать на вязальной машине полностью готовое изделие, что минимизирует ручной труд на производстве, но имеет очень высокую стоимость разработки изделия и большие ограничения по используемым переплетениям и формам. Каждый способ имеет свои особенности и, следовательно, каждому соответствуют определенные методы швейной обработки.

Необходимо отметить, что каждая из используемых технологий требует разного оснащения трикотажного производства, например для полурегулярного метода обязательно наличие раскройного оборудования, машины для регулярного способа должны быть оснащены гребенкой и системой обрезки нити, для цельновязанных изделий используются специальные широкие машины, оснащенные иглами с увеличенным размером головки для вязания через иглу. Кроме того производство по технологии Fully Fashion требует высокой квалификации программистов-дессинаторов и модельера-конструктора.

**Регулярный способ производства трикотажа (fully fashion)**

Этим способом можно, как в ручном вязании, вывязать детали изделия точно по лекалу, убавляя и прибавляя петли по контуру. Изделие не требует подкроя, детали соединяются на машинах цепного стежка (кеттельные, стачивающие машины, швейные машины цепного стежка) без обрезания краев, сырье используется очень экономно (отходы составляют 2-5%).

Недостатки этого способа - низкая производительность при относительной сложности и довольно высокой стоимости, а достоинства - высокое качество изделий, изготовленных таким способом. Поэтому при производстве сложных изделий средней и верхней ценовой категории из высококачественной пряжи используют именно его.

Детали изделия могут быть получены не только плоской формы, но и объемной путем изменения количества петель или рядов внутри контура детали. Так, групповой перенос петель позволяет получить вертикальную вытачку (например, талиевую), а вязание неполных петельных рядов с выключением группы игл - горизонтально расположенную (например, нагрудную) вытачку. Оборудование, которым оснащено наше производство позволяет выполнять такие операции в автоматическом режиме.

**Сбавки, прибавки и кеттлевка**

Чтобы сузить или расширить вязаную деталь и получить заданную форму, в процессе вязания на вязальной машине изменяют количество петель, размещающихся но ширине детали. Соответственно, для расширения количество петель увеличивают, а для сужения — уменьшают.

В случае сужения крайние, а иногда и соседние с ними петли в середине изделия переносятся с выключаемых игл на другие иглы, остающиеся в работе. Боковые края деталей вязаных изделий, полученных таким способом, имеют красивую плавную линию и очень удобны для стачивания эластичными цепными швами.

Исключение составляет лишь групповая сбавка петель, когда из работы выключают одновременно 5-10 и более игл. Так делают, к примеру, у пройм жакетов и джемперов или при вывязывании оката рукава. При групповой сбавке петли со всех выключаемых игл переносятся поочередно одновременно с провязыванием двух краевых петель (кеттлевка). В результате на краях деталей образуются выраженные прямоугольные уступы с закрытыми петлями, величина которых зависит от числа выключаемых игл и от того, через сколько петельных рядов делают сбавку.

В процессе прибавки простым добавлением петель на боковых краях деталей образуются мелкие уступы, образованные на включаемых иглах. Это значит, что припуск на шов в данном случае будет неровным. Когда нужно получить плавный контур края вывязанной детали, используется прибавка переносом краевых петель на включаемые иглы с заполнением освободившихся игл петлями предыдущего ряда. Данный способ не всегда оправдан, так как увеличивает время и стоимость вязания, но позволяет получить идеальный контур детали.

Используя сбавки и прибавки путем перенесения петель с иглы на иглу, можно получить не только деталь нужной формы, но и интересный рисунчатый эффект по кромке изделия. Иногда переносят по несколько петель сразу. Наиболее заметен этот своеобразный рисунок при получении линии реглана. Иногда таким образом вывязывают планку по борту изделия, перенося одновременно несколько петель, из которых и образуется планка.

**Пошив регулярных изделий**

Вязаные детали, полученные с вязальных машин, по минимуму подвергаются швейной обработке. Это делает качество изделий значительно выше, и они более удобны в носке по сравнению с выкроенными из полотен. Трикотажные изделия высокого качества, как правило, предполагают регулярный способ изготовления и минимальное количество швейных операций.

Оверлочный шов с обрезкой детали, используемый при производстве трикотажа кроеным и полурегулярным способом имеет большую ширину и толщину шва, что отрицательно сказывается на потребительских свойствах. Кроме того, эластичность оверлочной и челночной строчки не соотвествует эластичности трикотажного полотна, что может привести к разрывам швов при резких движениях. Поэтому для обработки высококачественных изделий, изготовленных регулярным способом мы используем только машины цепного стежка с небольшими припусками на шов, так как край заработан и не распускается. «Высший пилотаж» при пошиве трикотажных изделий - это максимальное использование кеттельных швов. Кеттлевка — это соединение деталей трикотажных изделий цепной строчкой «петля в петлю». Создается впечатление, что шва как бы и нет. Обычно прикеттлевывают бейки горловины, воротники, борта, накладные карманы. Но можно также этим методом соединять горизонтальные швы, допустим, плечевые. Получается максимально тонкое, эластичное, практически незаметное соединение. На оборудовании для пошива трикотажа нет челнока, вместо него используется петлитель. С лицевой стороны такая строчка состоит из стежков, а с изнаночной создается цепочка из петель. Поэтому такие строчки называются цепными. [1]

Возможность получения на вязальных машинах деталей изделия заданной формы является очень важным преимуществом современного трикотажного производства, так как позволяет получить изделия высокого качества в промышленных масштабах. Отсутствие отходов при раскрое значительно сокращает расход сырья на каждое изделие, что способствует уменьшению его себестоимости.

## Разработка функциональной модели предметной области

Основным процессом предметной области данного курсового проекта является осуществление продаж в интернет-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, web-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.

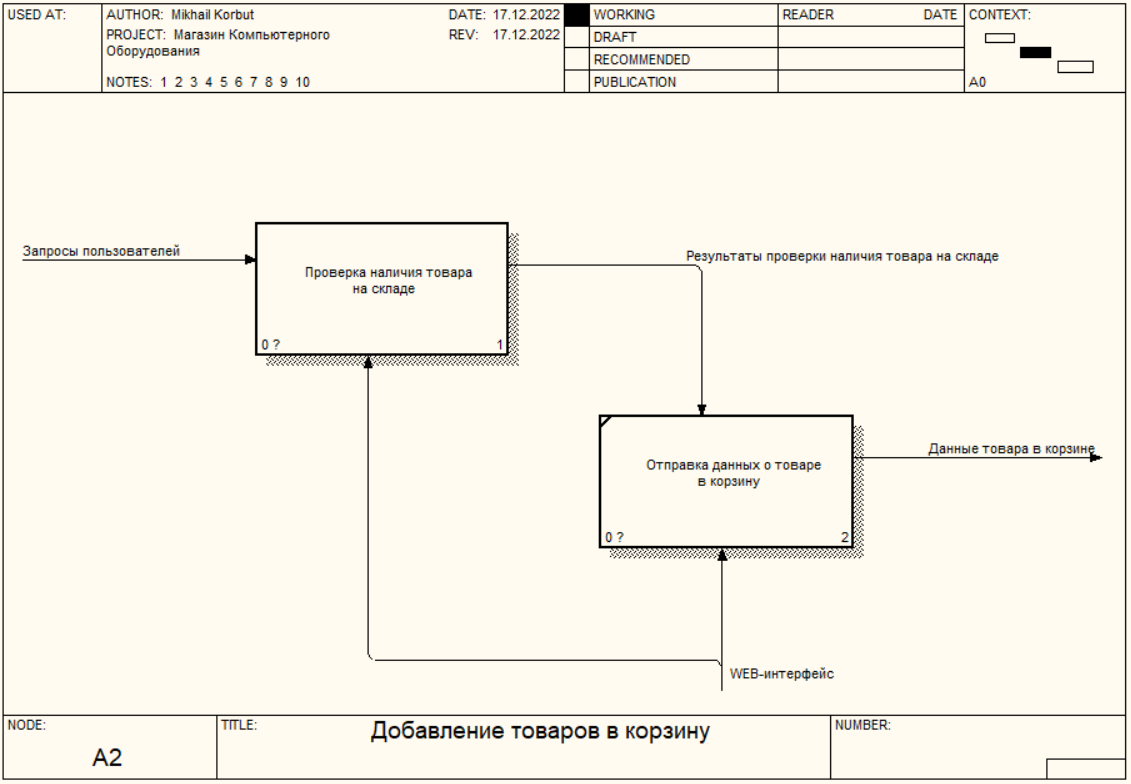


Рисунок 1.3 – Декомпозиция блока «Добавление товаров в корзину»

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.

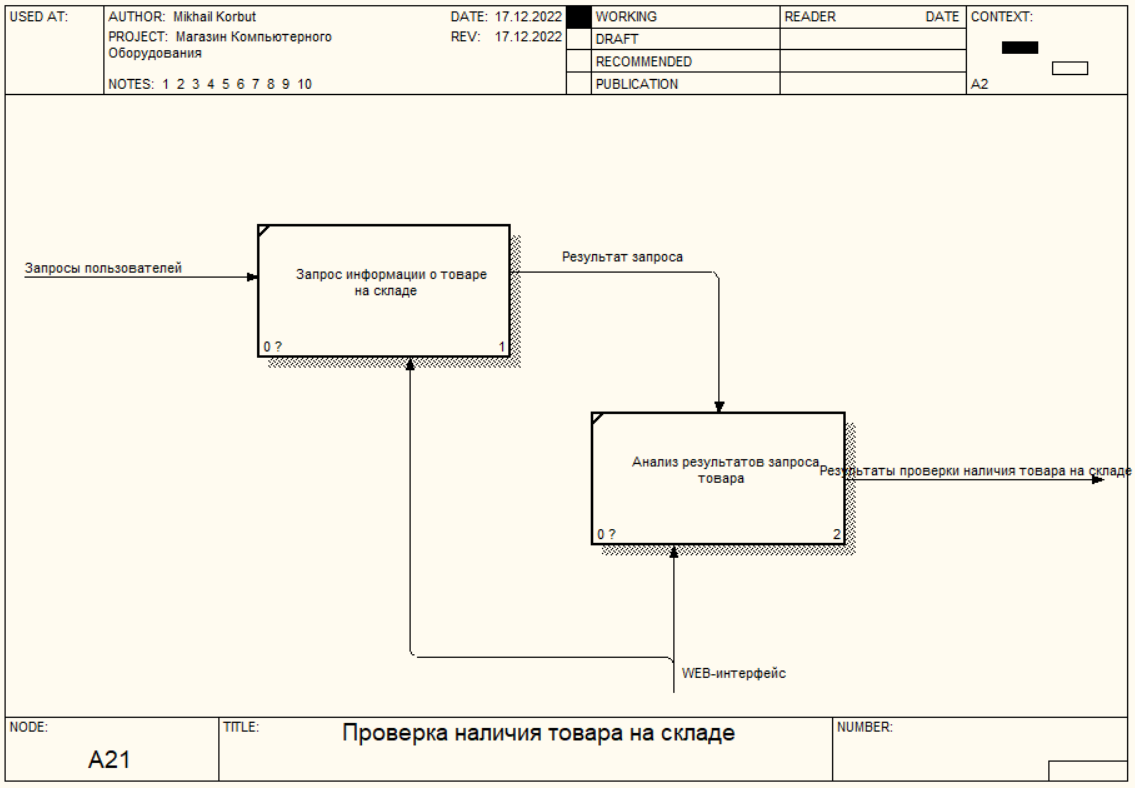


Рисунок 1.4 – Декомпозиция блока «Проверка наличия товара на складе»

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.

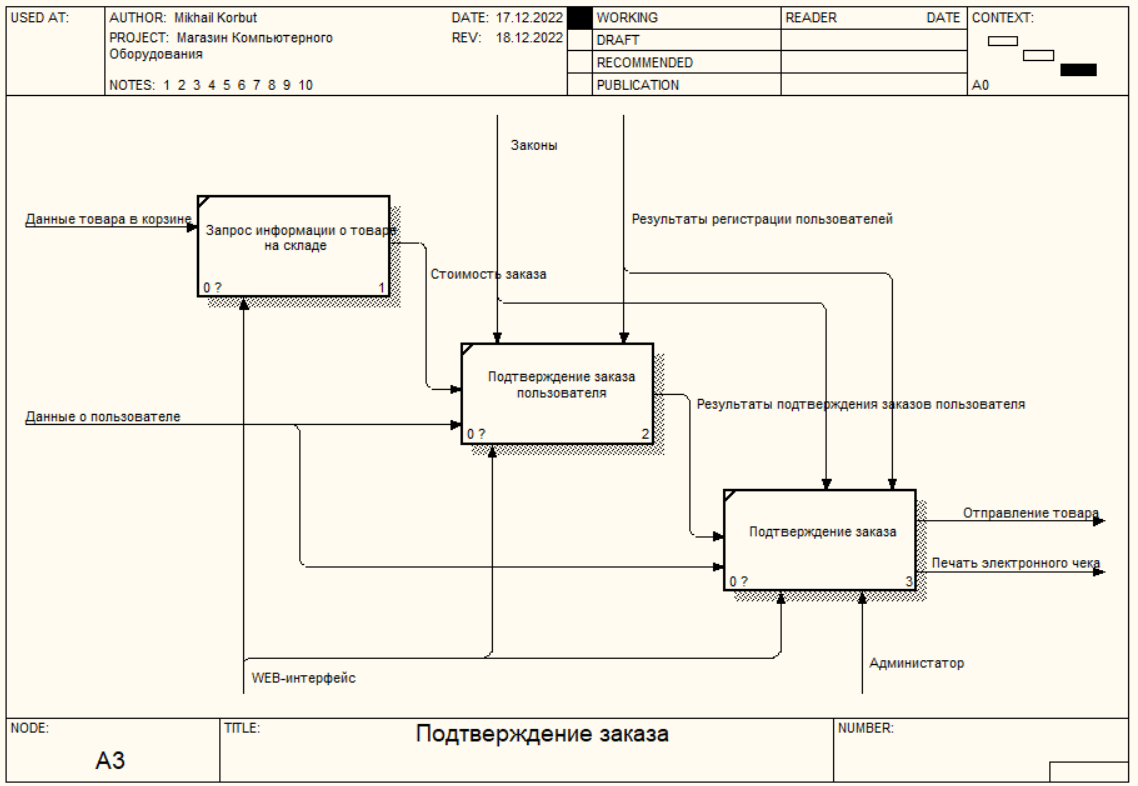


Рисунок 1.5 – Декомпозиция блока «Подтверждение заказа»

## 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).

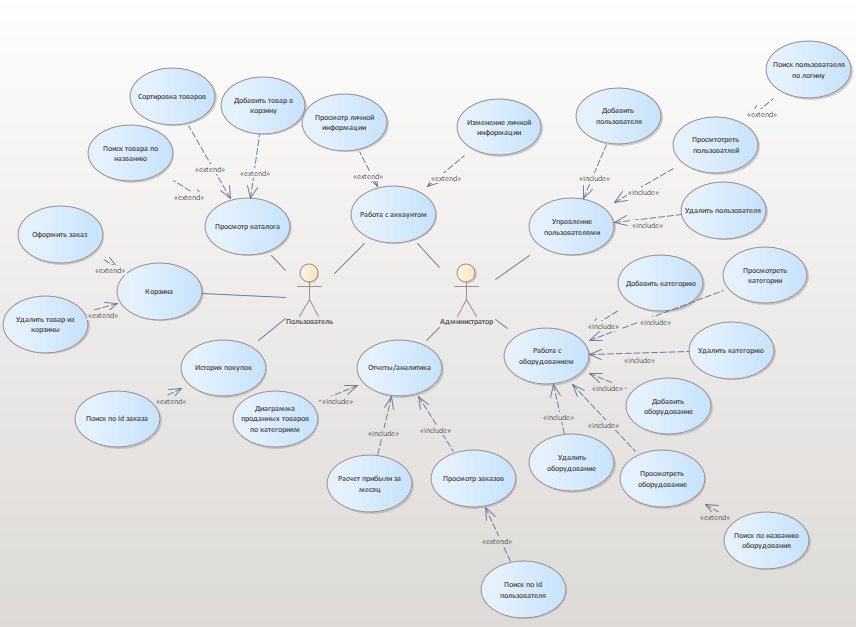


Рисунок 1.6 – Диаграмма вариантов использования

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.

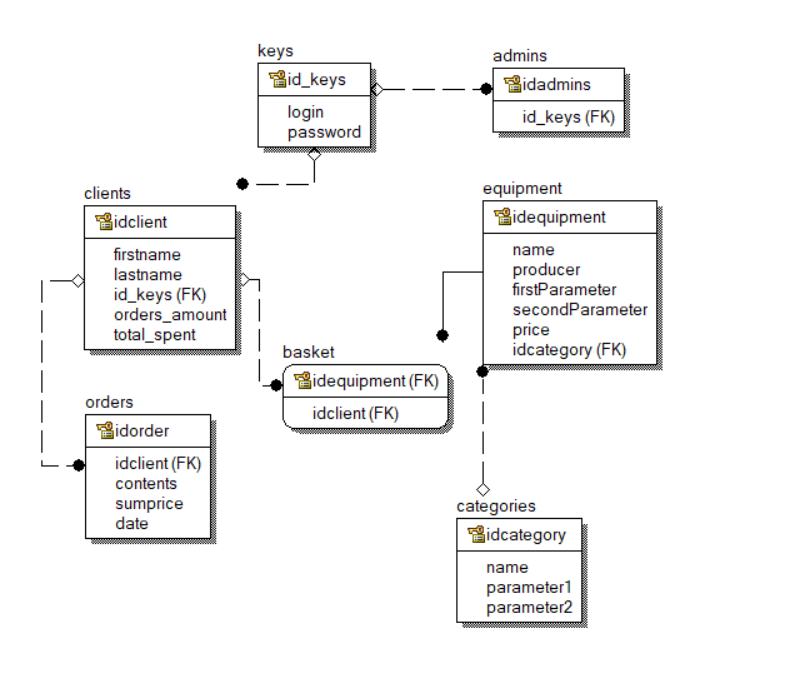


Рисунок 1.7 - Информационная модель системы

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. 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

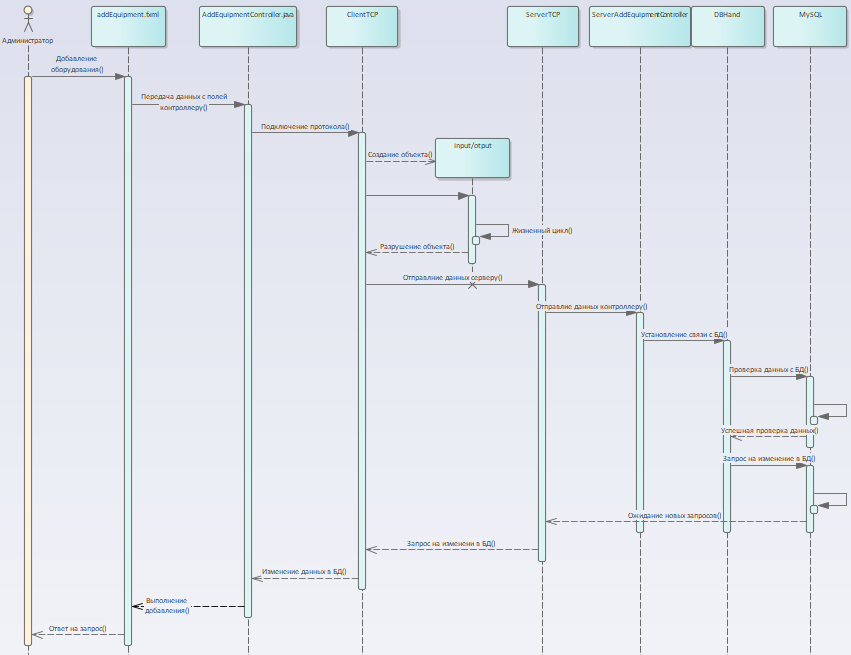


Рисунок 1.8 – Диаграмма последовательности процесса добавления оборудования

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. 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).

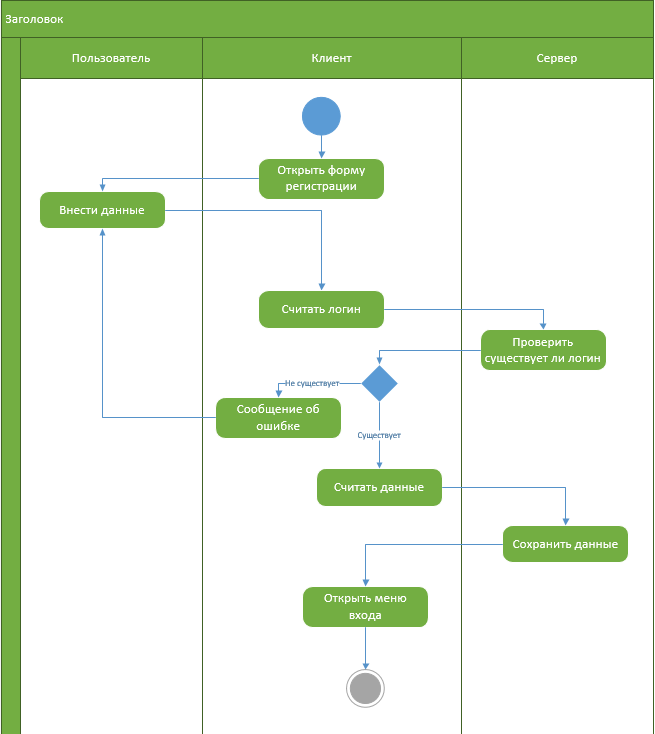


Рисунок 1.9 – Диаграмма деятельности

* + 1. 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.

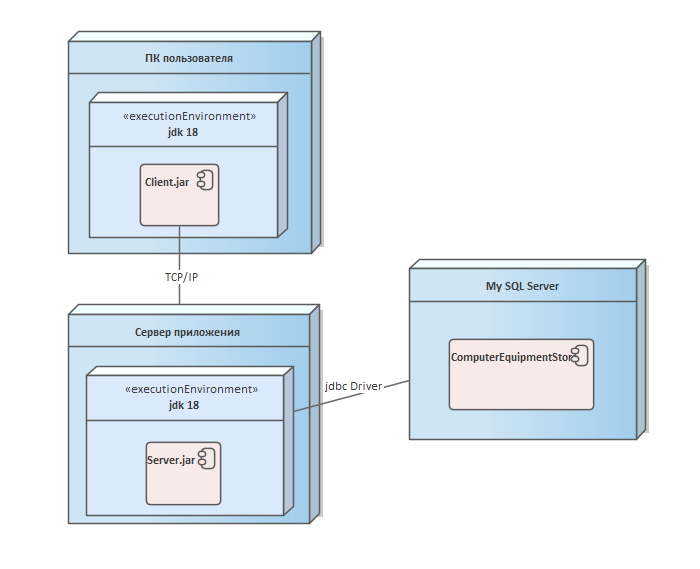


Рисунок 1.10 – Диаграмма развертывания приложения

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.

Для больших компаний создающих вязанный трикотаж уже существует множество программных продуктов высокого класса. Но для обычных людей, занимающихся вязанием как хобби или ремесленников для которых производство изделий на заказ является источником заработка, такие программы могут быть очень сложными, дорогими, а так же им может быть не нужен весь функционал, а только его часть. Данное приложение будет разрабатываться для второй группы лиц.

Самое сложное в производстве вязаного трикотажа это расчеты. Необходимо снять мерки, сделать чертеж изделия и вычислить все необходимые размеры, но не в сантиметрах, а в рядах и петлях, для чего необходимо иметь петельную пробу. Процент ошибок при вычислении вручную большой, поэтому именно расчеты при составлении выкроек нужно автоматизировать.

Так как программное средство направлено на составление выкроек для вязания, необходимо спроектировать базу данных для хранения всех необходимых значений для составления выкроек, удобный интерфейс для ввода всей информации. Необходимо автоматически (по желанию пользователя) подсчитывать необходимые параметры для выкройки, основываясь на введенных значениях мерок и петельной пробы, предоставлять картинку выкройки с этими параметрами и текстовое описание к ней. Пользователь должен иметь возможность обратиться к созданным выкройкам в любой момент, запросить расчет или изменить данные. Так же к выкройкам, которые создал пользователь, должен иметь доступ только он, поэтому нужно разработать систему аккаунтов.

Для надежной работы приложения нужно следить за действиями пользователей, например чтобы предотвратить атаку «Отказ в обслуживании». Для этого нужно реализовать журнал событий, в который будут заноситься все важные действия в приложении. Доступ к журналу имеет только администратор и только он может его очистить. Но самого журнала не хватит, чтобы предотвратить атаку, необходимо также реализовать возможность заблокировать пользователя (пока статус не изменится, данный пользователь не сможет пользоваться приложением).

## 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 5.4.1 представлена диаграмма классов пакета DB.

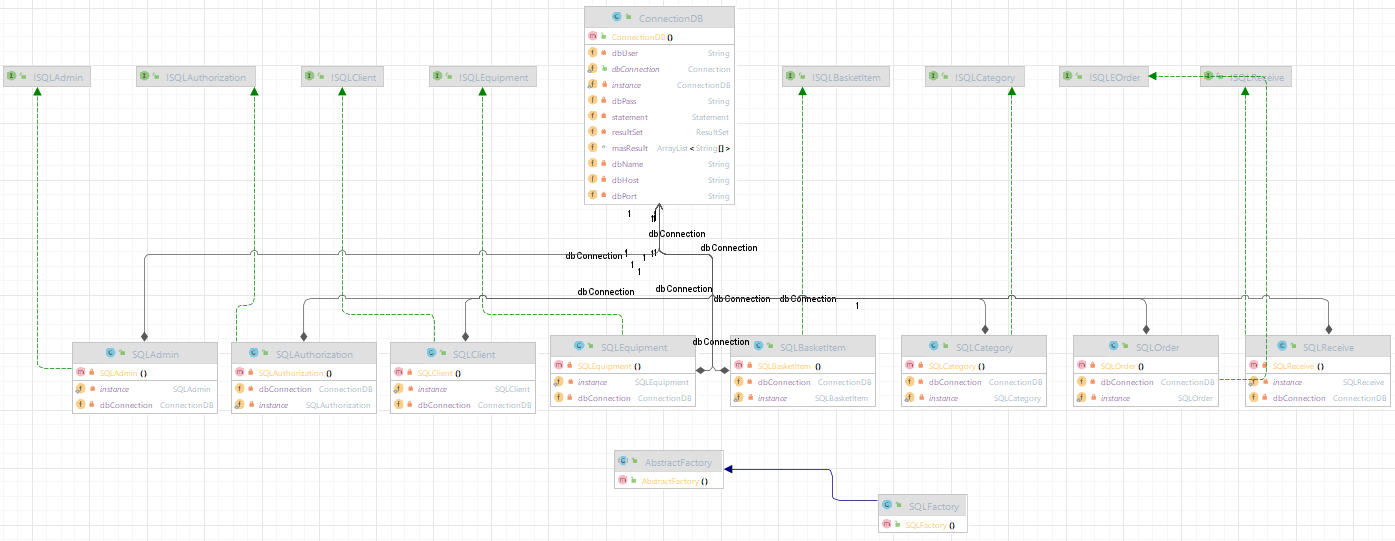


Рисунок 5.4.1 – Диаграмма классов пакета DB

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.

На рисунке 5.4.2 представлена диаграмма классов пакета SchoolOrg.

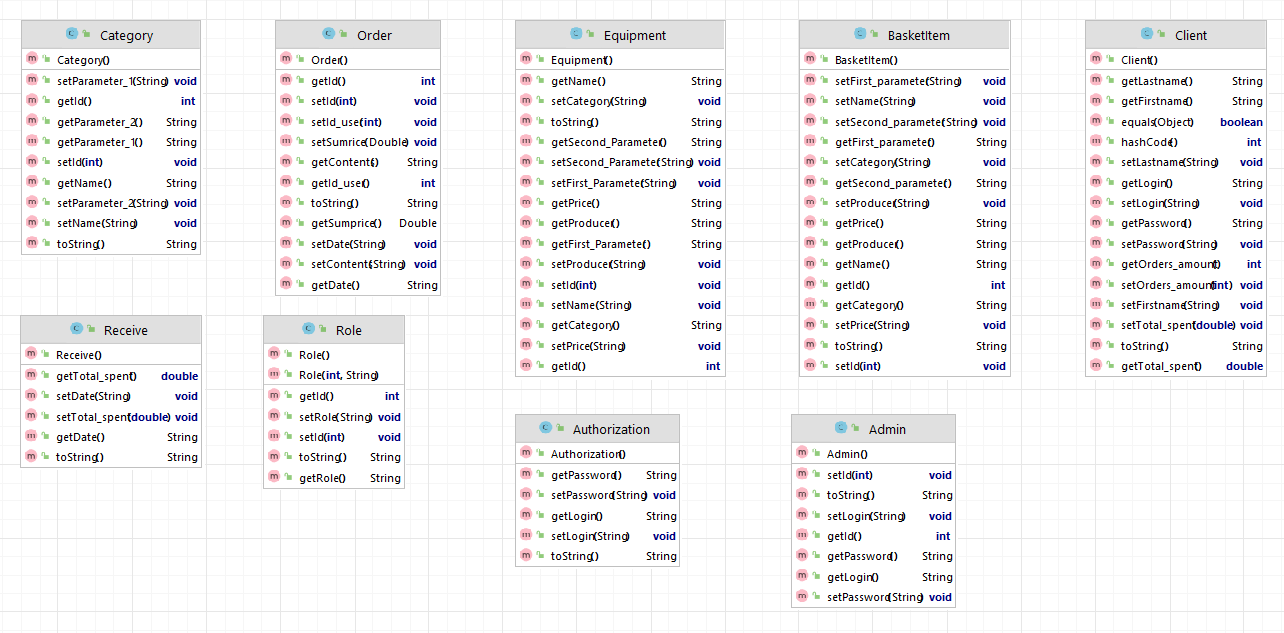


Рисунок 5.4.2 – Диаграмма классов пакета Computerequipmentstore

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 5.4.3.

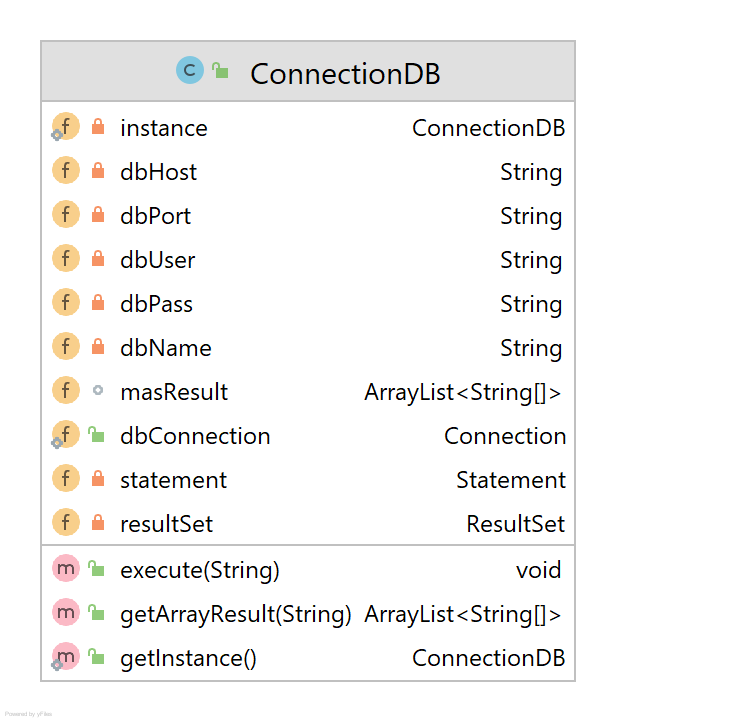


Рисунок 5.4.3 – Реализация паттерна Singleton

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 2.3.

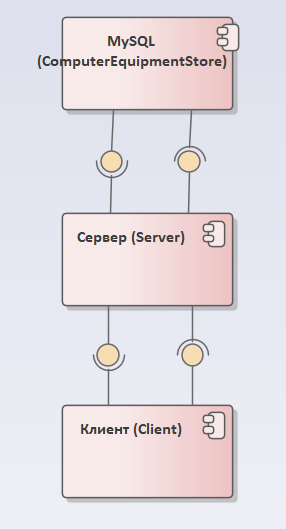


Рисунок 2.3 – Диаграмма компонентов

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 три процесса: процесс авторизации как пользователя, так и администратора, организация процесса покупки оборудования клиентом, составление отчета по продажам.

При входе в программу требуется ввести логин и пароль и в зависимости от введенной информации, в базе данных находится соответствующая роль и открывается нужное меню.

Если авторизация не прошла успешно, то на экран выводится соответствующее сообщение и можно попытаться авторизоваться снова.

У каждой роли есть свой индивидуальный функционал.

У администратора он более широкий. Помимо стандартных операций он может просмотреть и сформировать отчет о деятельности магазина, также все записи, которые хранятся в таблицах базы данных можно добавлять, изменять, удалять.

Пользователь помимо просмотра каталога может добавить понравившийся товар в корзину, после чего оформить покупку.

Алгоритм работы программы представлен на рисунке 2.4.

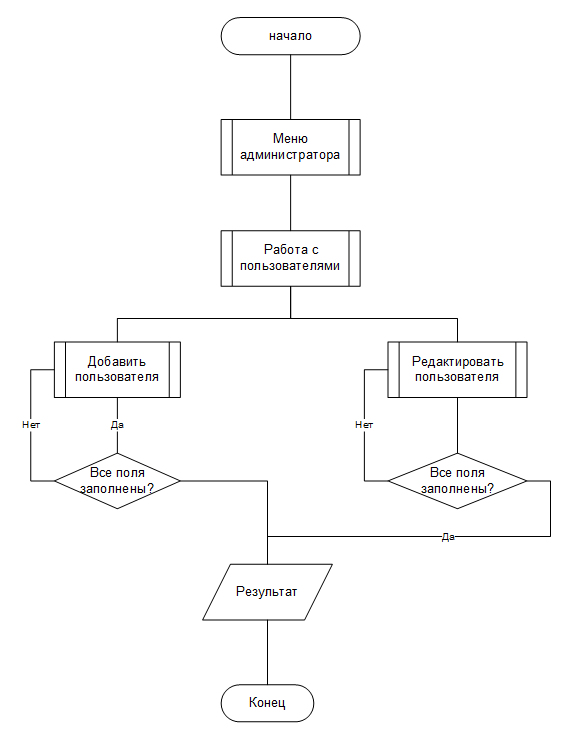


Рисунок 2.4 – Алгоритм работы программы

Второй процесс - это добавление товара в корзину клиентом функция, вызывающаяся при нажатии на кнопку «Добавить в корзину». После просмотра оборудования можно выбрать понравившийся 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 2.5.

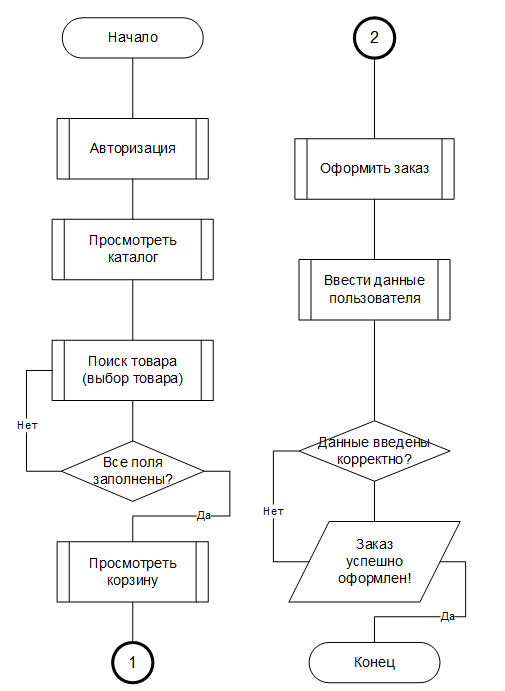


Рисунок 2.5 – Расчет выкройки

Третий процесс – это добавление производителя функция, вызывающаяся при нажатии на кнопку «Добавить в производителя». Введенные данные отправляются на сервер, который начинает их обрабатывать, подключаясь к базе данных. Если там нет такай записи, то

## 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.

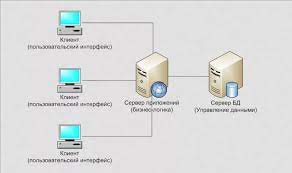


Рисунок 2.9 – Архитектура «клиент-сервер»

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 – 3.2).

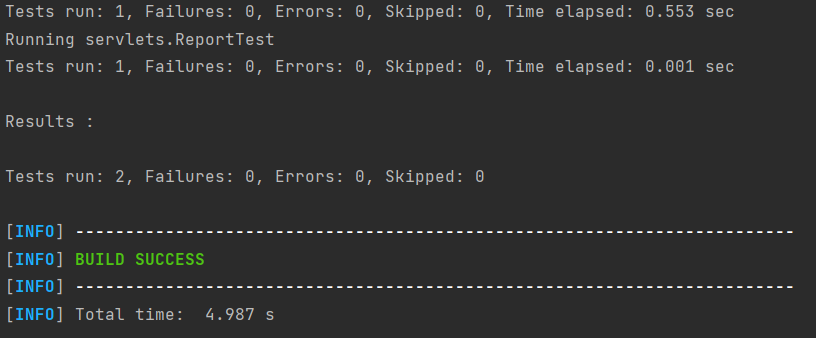


Рисунок 3.1 – Тестирование с помощью maven

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 3.3, 3.4, 3.5).

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 3.6)

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

Рисунок 3.3 – Ошибки при регистрации

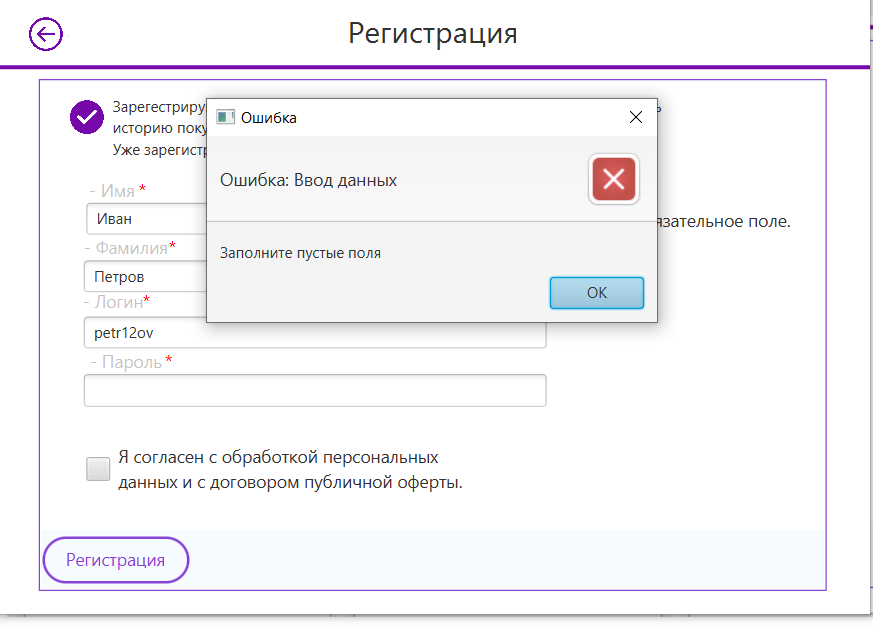


Рисунок 3.4 – Ошибки при регистрации

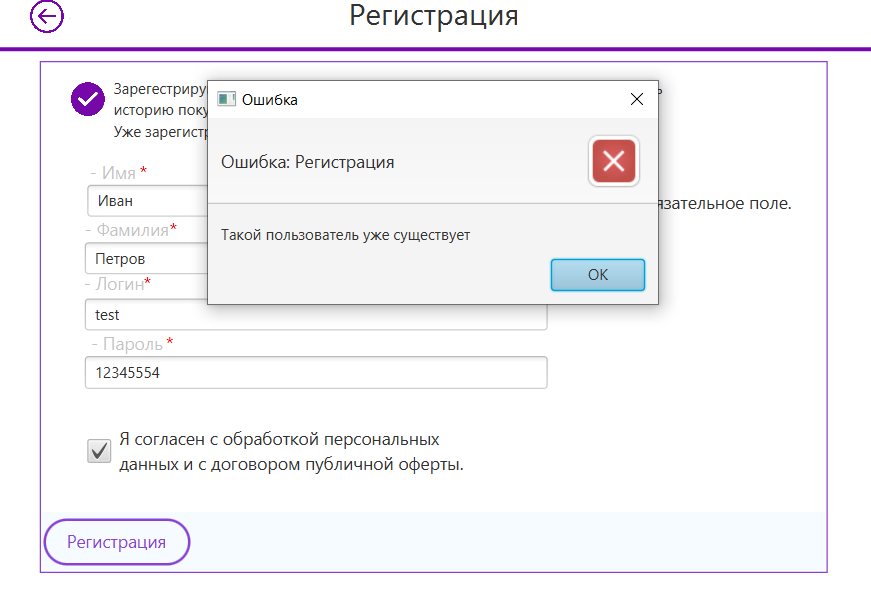


Рисунок 3.5 – Ошибки при регистрации

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

Рисунок 3.6 – 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

Для начала нужно сгенерировать базу данных, для этого запускаем утилиту MySQL Workbench, открываем скрипт генерации базы данных (Приложение В) и нажимаем значок выполнить (молния ).

Далее открыть файл ConnectionDB.java на сервере и прописать данные о порте пароле и название базы данных. (рисунок 4.1).

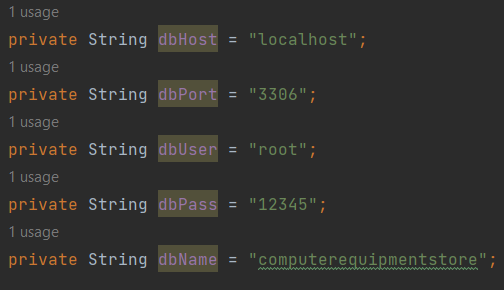


Рисунок 4.1 – Заполнение полей в ConnectionDB.java

Далее необходимо запустить сервер проекта. Для этого пользователь должен запустить ServerStart.java файл, после чего Main.java на клиенте. Пример запуска показан на рисунке 4.2. Пример отображения состояния подключений показан на рисунке 4.3.



Рисунок 4.2 – Запуск сервера

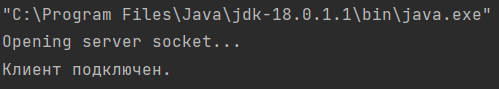


Рисунок 4.3 – Отображение состояний подключения к серверу

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 4.4)

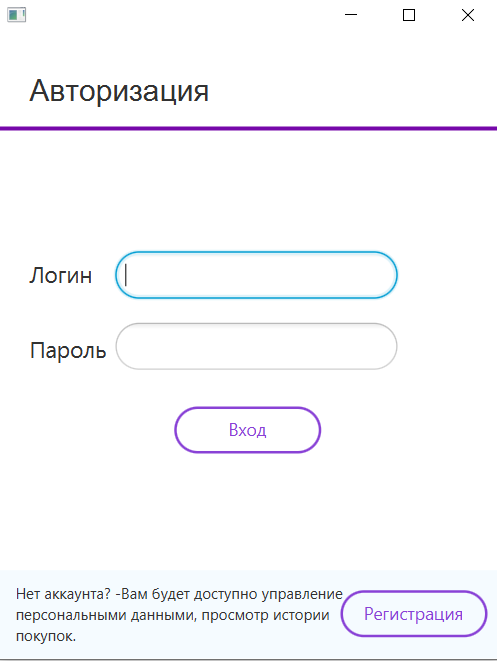


Рисунок 4.4 – Меню входа и регистрации

Допустим у нас нет аккаунта, нажимаем кнопку “Регистрация”, после чего нам отобразиться форма для регистрации (рисунок 4.5). При успешной регистрации вы попадете в меню клиента.

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

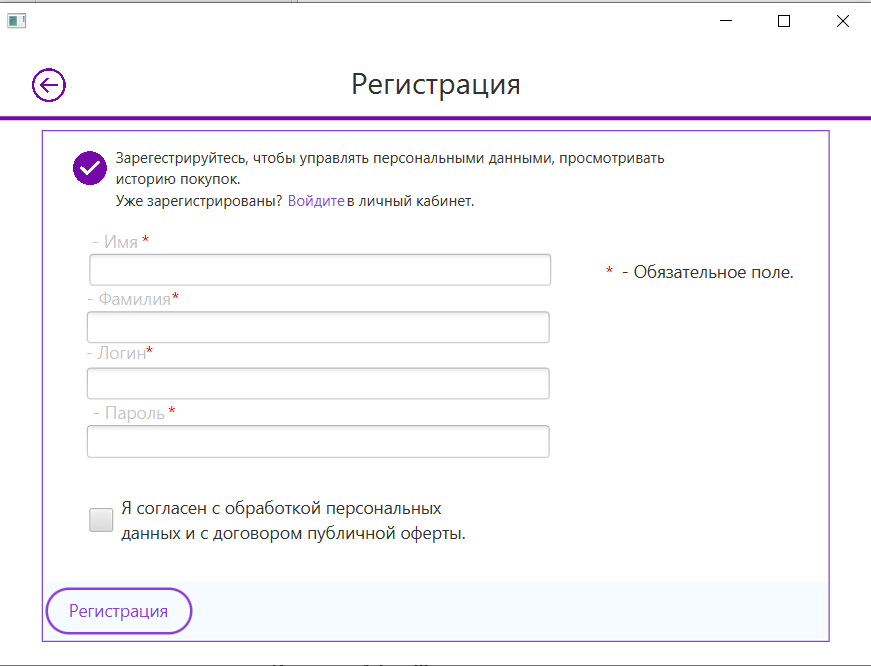


Рисунок 4.5 – Форма для регистрации

**Действия клиента**

На меню пользователя можно увидеть все возможности пользователя (рисунок 4.6). Он может просмотреть каталог, просмотреть корзину, историю покупок или нажав на иконку пользователя открыть окно со своими данными (рисунок 4.7).

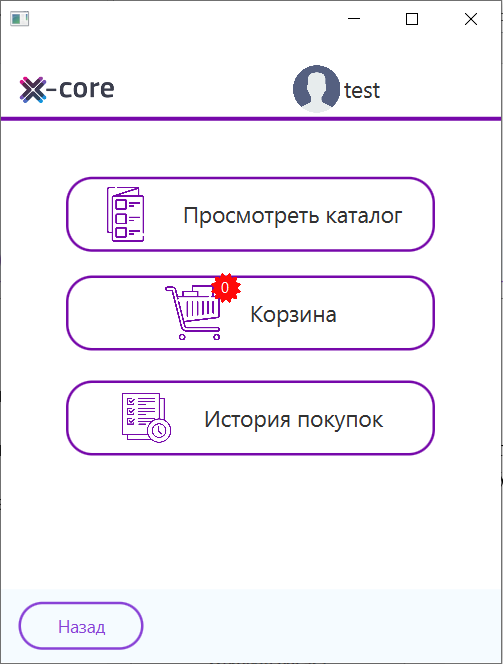


Рисунок 4.6 – Главное меню пользователя

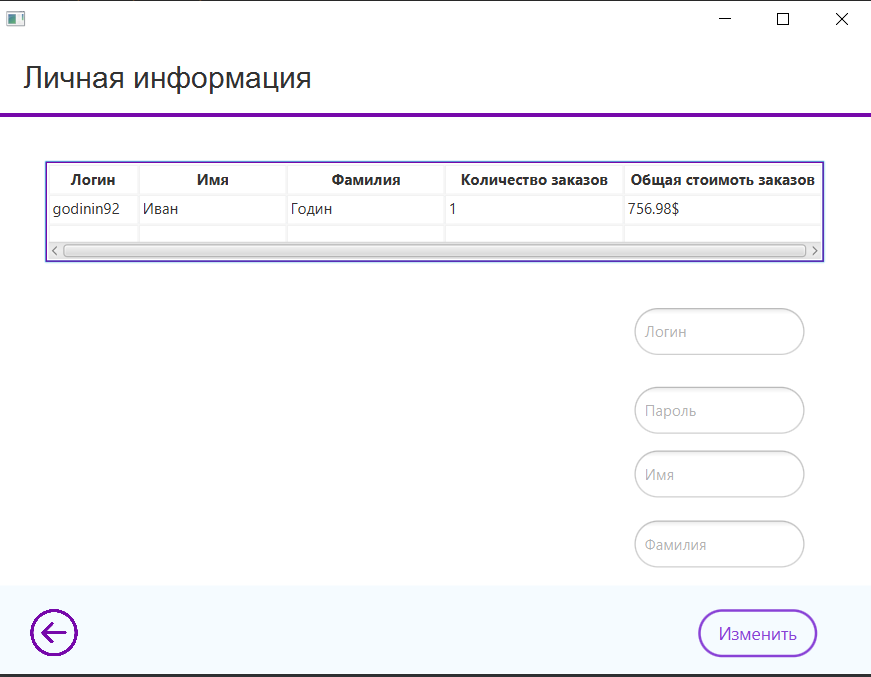


Рисунок 4.6 – Окно с данными пользователя

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).

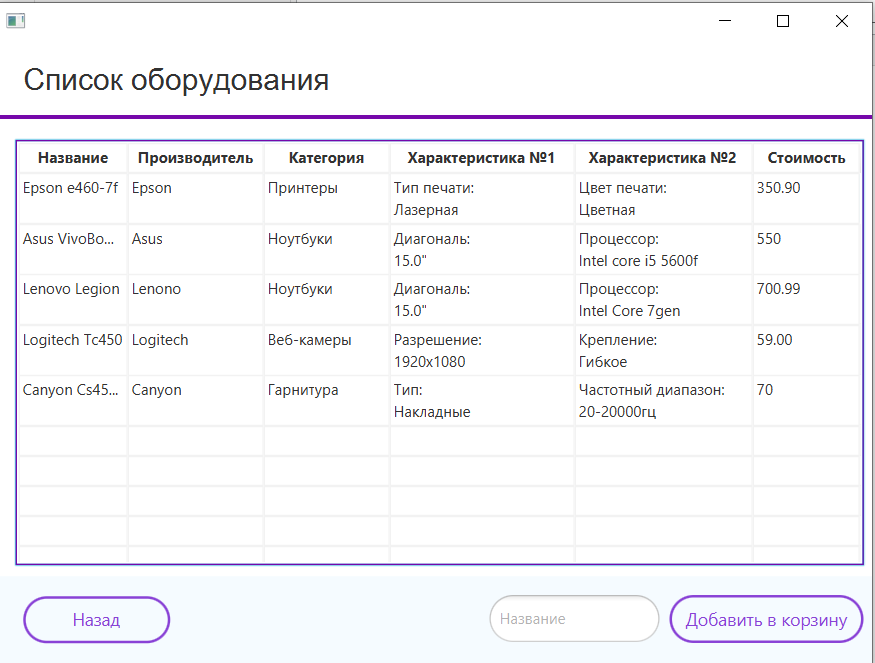


Рисунок 4.9 – Просмотр оборудования

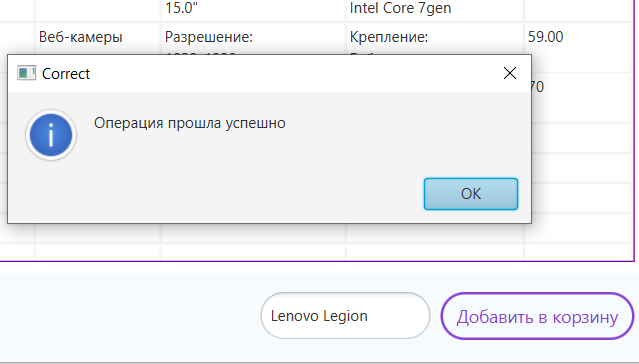


Рисунок 4.9 – Добавления оборудования в корзину

После добавление товара на иконке корзины в меню пользователя появится число товаров в корзине (рисунок 4.10).

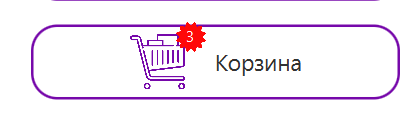


Рисунок 4.10 – Отображение количества товаров в корзине

Теперь рассмотрим пункт меню «Корзина», в корзине можно просмотреть содержимое корзины и удалить товар из корзины (рисунок 4.11).

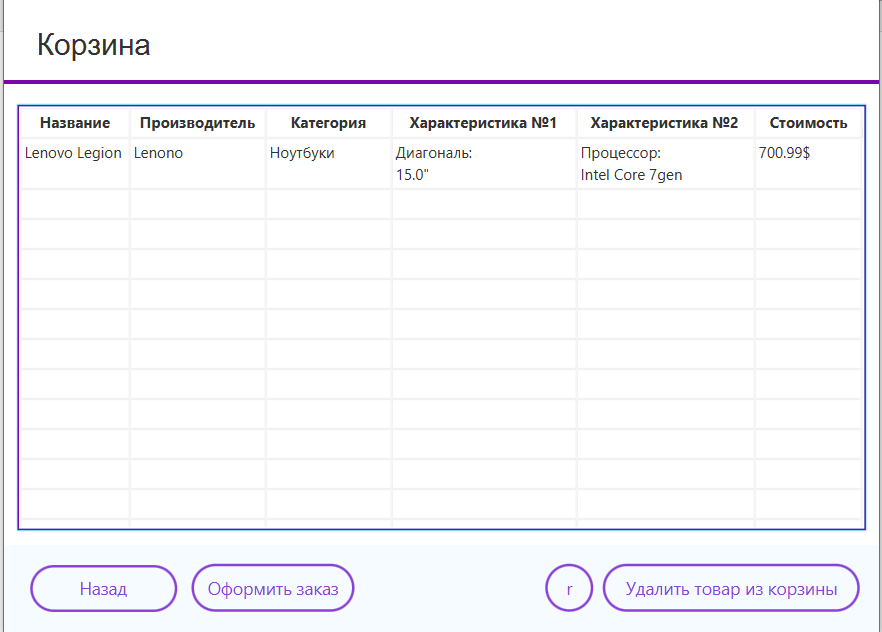


Рисунок 4.11– Просмотр корзины

Оформим заказ для этого нажмем на кнопку «Оформить заказ» (рисунок 4.12).

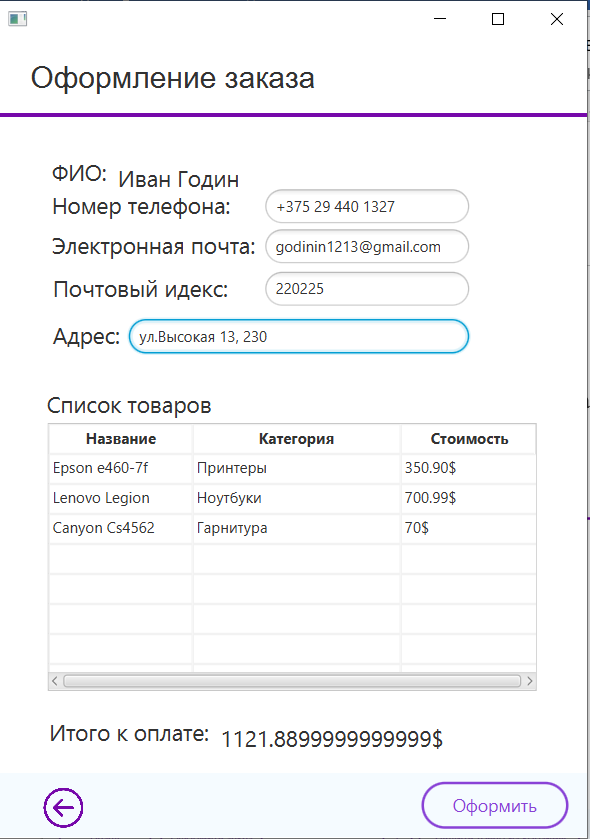


Рисунок 4.12 – Просмотр корзины

Вернемся в меню пользователя и просмотрим историю покупок клиента пункт меню «История покупок» (рисунок 4.13).

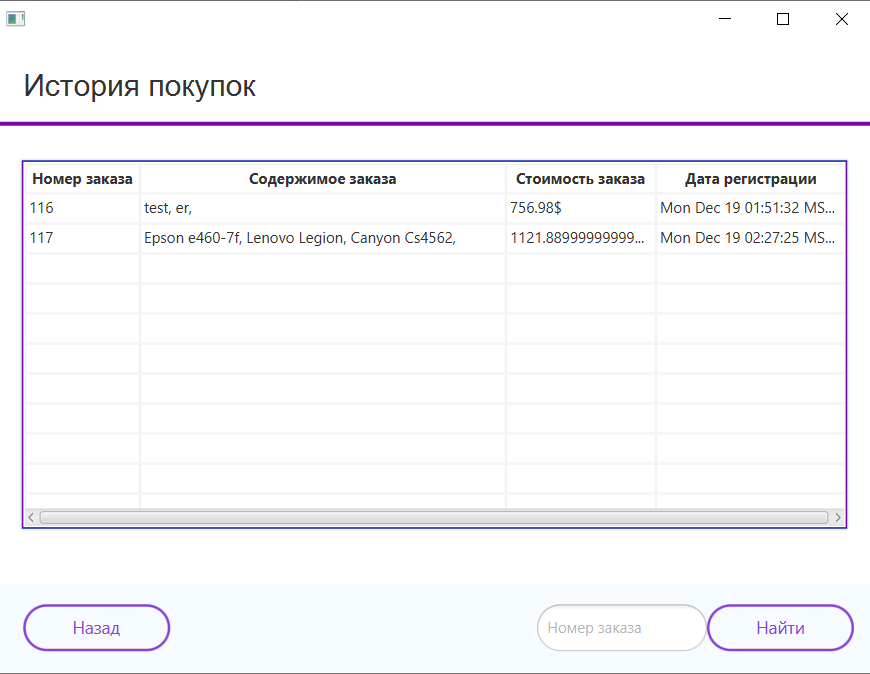


Рисунок 4.13– Просмотр корзины

После описания всех функций пользователя перейдем к меню админа (рисунок 4.14). 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 4.15).

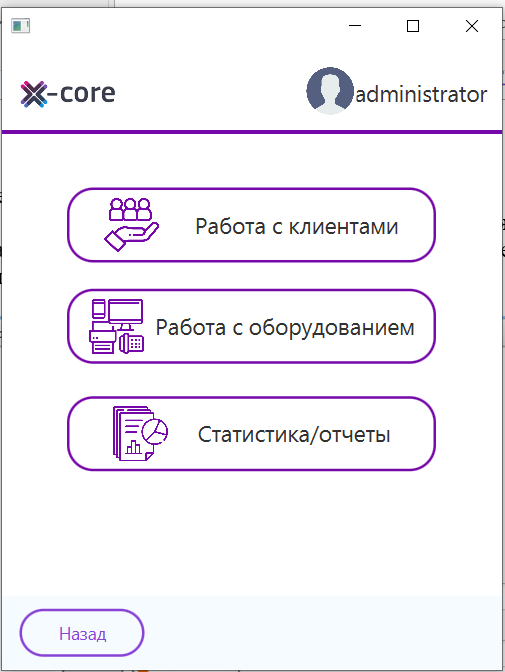


Рисунок 4.14 – Меню администратора

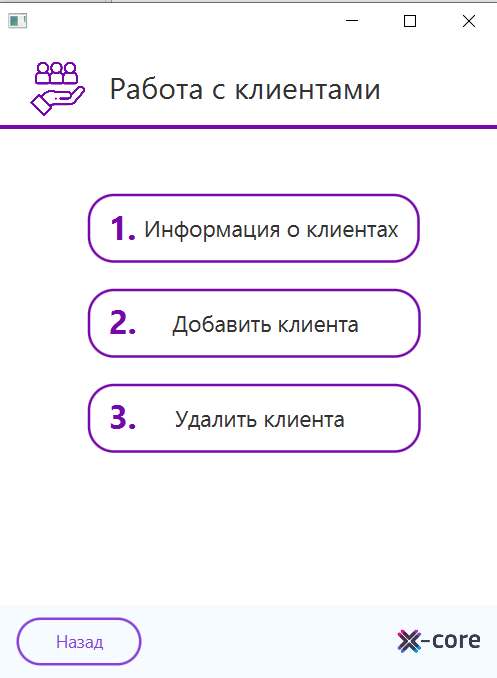


Рисунок 4.15 – Работа с клиентами

Теперь рассмотрим пункт меню «Информация о клиентах», в открывшемся меню можно просмотреть информацию о клиентах а также найти клиента по логину (рисунок 4.16).

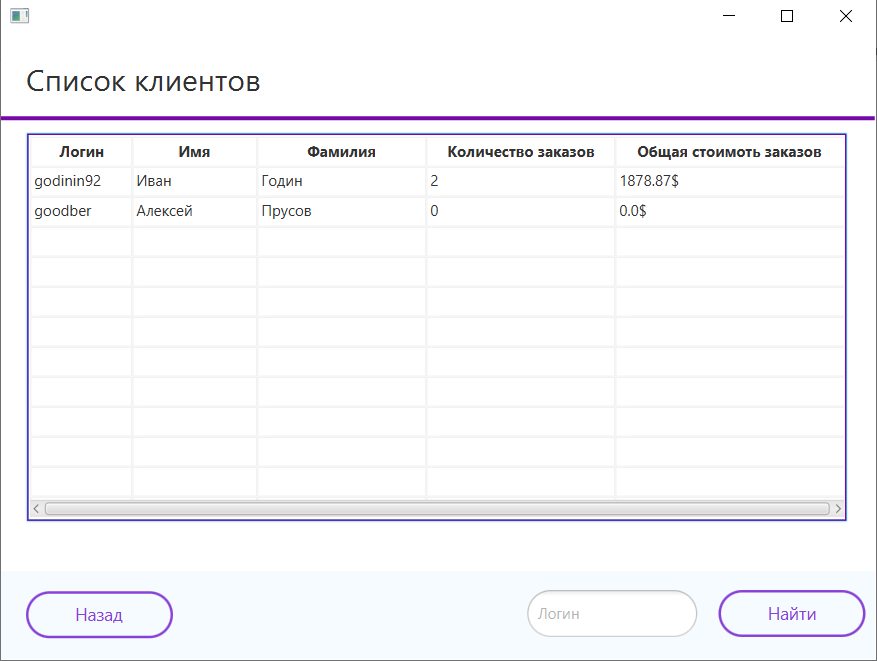


Рисунок 4.16 – Информация о клиентах

Вернемся в меню «Работа с клиентами» при нажатии на пункт меню 2 откроется окно регистрации (рисунок 4.5). Рассмотрим пункт меню 3 (рисунок 4.17).

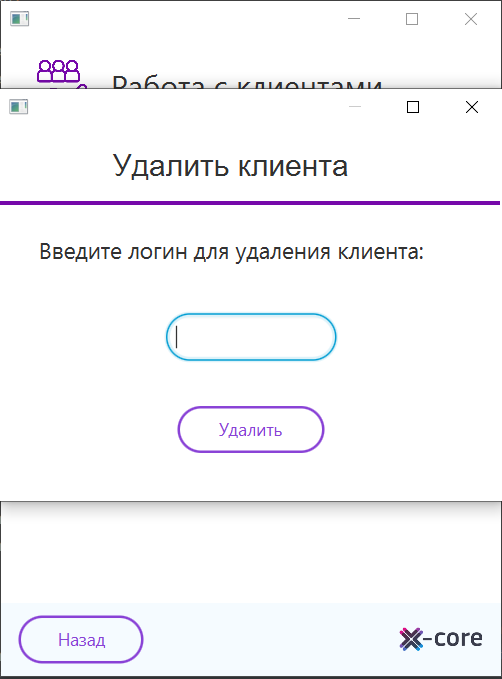


Рисунок 4.17 – Выкройка шапки и текстовое пояснение

Меню администратора, выберем пункт меню «Работа с оборудованием» (рисунок 4.18).

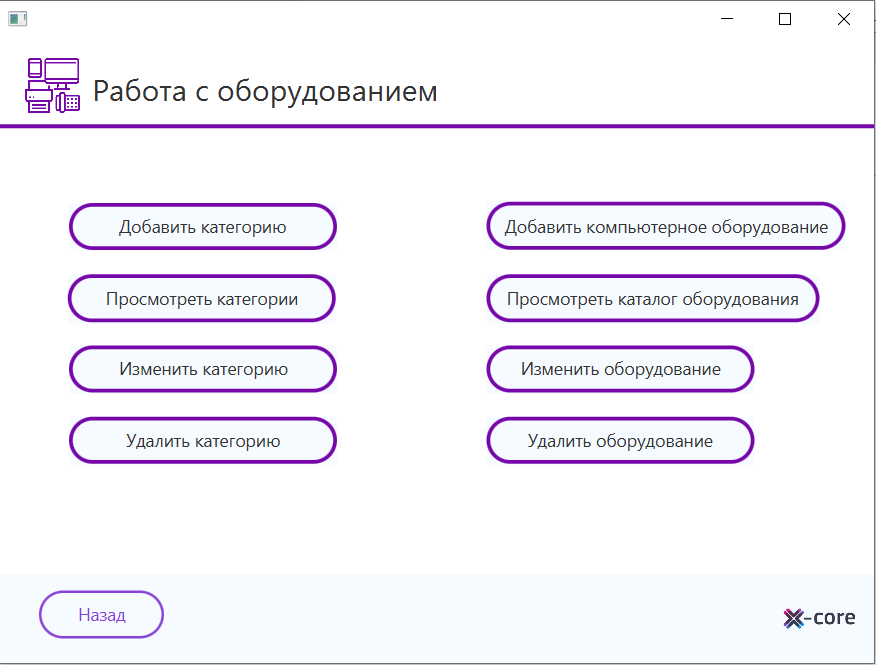


Рисунок 4.18 – Выкройка шапки и текстовое пояснение

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

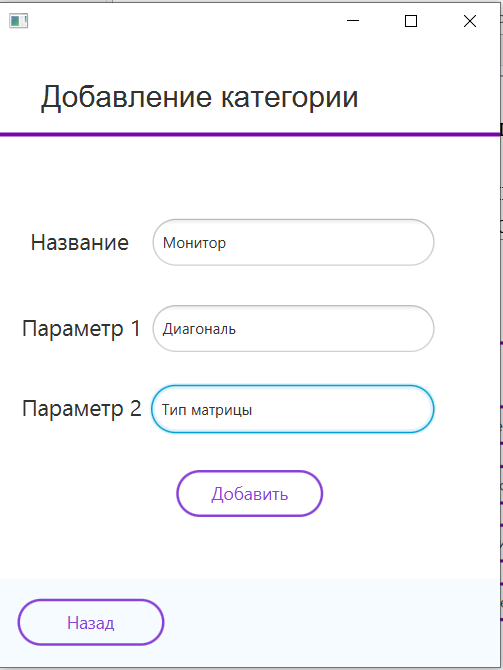


Рисунок 4.19 – Добавление категории

Добавим оборудование, сперва нужно выбрать категорию (рисунок 4.20) после в появившимся окне необходимо заполнить все поля после нажать кнопку «Добавить» (рисунок 4.21).

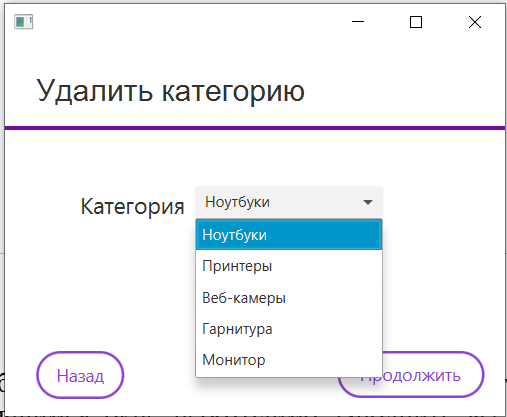


Рисунок 4.20 – Выбор категории

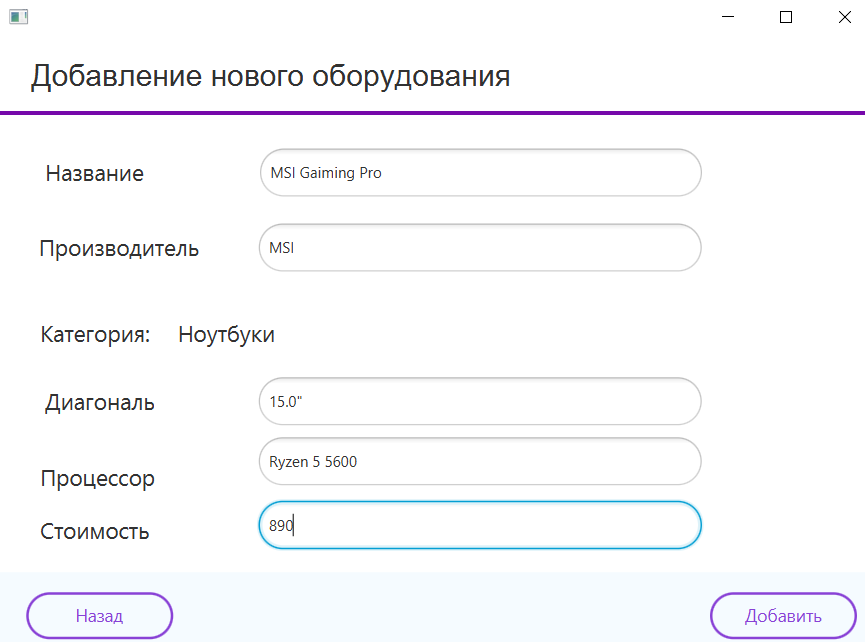


Рисунок 4.21 – Добавление оборудования

Пункты меню «Изменить категорию», «Изменить оборудование» и «Удалить категорию», «Удалить оборудование» выполняются аналогично пунктам меню «Изменить клиента» (рисунок 4.6) и «Удалить клиента» (рисунок 4.17) соответственно.

Вернемся в меню администратора и выберем пункт меню «Статистика/отчеты (рисунок 4.22).

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

// Методы класса Patern. Подсчет параметров для выкроек

public List<Integer> calculateHat(Hat hat){

return hat.calculate(densityHorizontal, densityVertical);

}

public List<Integer> calculateSocks(Socks socks){

return socks.calculate(densityHorizontal, densityVertical);

}

public List<Integer> calculateMittens(Mittens mittens){

return mittens.calculate(densityHorizontal, densityVertical);

}

// Метод класса Hat. Подсчет параметров для выкройки шапки

public List<Integer> calculate(float densityHorizontal, float densityVertical) {

// 6 число-количество стягиваемых перель на макушке

List<Integer> patternValues = new ArrayList<Integer>();

int hatW, hBeforeTopHat, hatH, petalW, topHatH, countTighten;

// Ширина одного лепестка макушки

petalW = (int)((densityHorizontal\*headGirth\*0.85+2)/4);

// Ширина всей шапки

hatW = petalW\*4;

// Высота шапки

hatH = (int)(((hatDepth/2)+lapelLen)\*densityVertical);

// Высота мукушки и количества петель для стягивания

if(petalW % 2 == 0){

topHatH = petalW - 1;

countTighten = 8;

}

else {

topHatH = petalW - 2;

countTighten = 12;

}

// Высота до макушка

hBeforeTopHat = hatH - topHatH;

patternValues.add(topHatH); // topHatH

patternValues.add(petalW);

patternValues.add( hatH);

patternValues.add(hBeforeTopHat);

patternValues.add(hatW);

patternValues.add(countTighten);

return patternValues;

}

// Метод класса Mittens. Подсчет параметров для выкройки варежки

public List<Integer> calculate(float densityHorizontal, float densityVertical) {

List<Integer> patternValues = new ArrayList<Integer>();

Продолжение приложения Б

int countCollect,heelH, tipH, sidesW, tipW, halfMittensW, bFH, bFW, beforeTipH, betweenCuffsBFH, cuffsH, mittensW;

// ширина половины варежки

halfMittensW = (int)(densityHorizontal\*palmGirth/2);

// ширина варежки

mittensW = halfMittensW\*2;

// длина монжета

cuffsH = (int)(cuffsLen\*densityVertical);

// длина до большого пальца

betweenCuffsBFH = (int)(cuffsThumbLen\*densityVertical);

int buf = (int)halfMittensW%3;

if(buf == 1){

tipW = (int)halfMittensW/3 + 1;

sidesW = (int)halfMittensW/3;

}

else if(buf == 2){

tipW = (int)halfMittensW/3;

sidesW = (int)halfMittensW/3 + 1 ;

}

else{

tipW = (int)halfMittensW/3;

sidesW = tipW;

}

// Длина носка

heelH = sidesW\*2 - 2;

// Длина до мизинца

beforeTipH = (int)(endLen \* densityVertical - (betweenCuffsBFH + heelH));

// Ширина большого пальца

bFW = (int)(thumbGirch \* densityHorizontal);

bFW += (int)(bFW/5) + 2;

// Длина большого пальца

bFH = (int)(thumbLen \* densityVertical);

tipH = 0;

countCollect = bFW;

while(countCollect > 4){

if(countCollect%2 == 1){

countCollect = countCollect/2+1;

}else{

countCollect = countCollect/2;

}

tipH++;

}

patternValues.add(countCollect);

patternValues.add(heelH);

patternValues.add(tipH);

patternValues.add(sidesW);

patternValues.add(tipW);

patternValues.add(halfMittensW);

patternValues.add(bFH);

patternValues.add(bFW);

Продолжение приложения Б

patternValues.add(beforeTipH);

patternValues.add(betweenCuffsBFH);

patternValues.add(cuffsH);

patternValues.add(mittensW);

return patternValues;

}

// Метод класса Socks. Подсчет параметров для выкройки носков

public List<Integer> calculate(float densityHorizontal, float densityVertical) {

List<Integer> patternValues = new ArrayList<Integer>();

int sidesW, tipW, halfSocksW, socksLength, heelH, bHeelH, beforeHeelH, cuffsH, socksW;

// ширина половины носка

halfSocksW =(int)(densityHorizontal\*footGirth/2);

// ширина носка

socksW = halfSocksW\*2;

// высота носка

beforeHeelH = (int)((cuffsLen+3)\*densityVertical);

// Высота манжета

cuffsH = (int)(cuffsLen\*densityVertical);

// Кол рядов между манжетой и пяткой

bHeelH = beforeHeelH - cuffsH;

int buf = (int)halfSocksW%3;

if(buf == 1){

tipW = (int)halfSocksW/3 + 1;

sidesW = (int)halfSocksW/3;

}

else if(buf == 2){

tipW = (int)halfSocksW/3;

sidesW = (int)halfSocksW/3 + 1 ;

}

else{

tipW = (int)halfSocksW/3;

sidesW = tipW;

}

// Длина пятки и носка

heelH = sidesW\*2 - 2;

// Длина носка без пятки и носка

socksLength = (int)(densityVertical\*footLen - 4\*heelH);

patternValues.add(sidesW);

patternValues.add(tipW);

patternValues.add(halfSocksW);

patternValues.add(socksLength);

patternValues.add(heelH);

patternValues.add(bHeelH);

patternValues.add(beforeHeelH);

patternValues.add(cuffsH);

patternValues.add(socksW);

return patternValues;

}

// Метод класса контроллера

Продолжение приложения Б

// Вычисление выкройки

@FXML

private void calculatePattern() throws IOException {

if (checkAll()) {

pattern.setPatternName(patternName.getText());

pattern.setDensity(dansity.getText());

pattern.setYarn(yarn.getText());

pattern.setDensityVertical(Float.parseFloat(densityVertical.getText()));

pattern.setDensityHorizontal(Float.parseFloat(densityHorizontal.getText()));

pattern.setType("Шапка");

if (lapelLen.getText() != "") {

hat.setHeadGirth(Float.parseFloat(headGirth.getText()));

hat.setHatDepth(Float.parseFloat(hatDepth.getText()));

hat.setLapelLen(Float.parseFloat(lapelLen.getText()));

} else {

hat.setHeadGirth(Float.parseFloat(headGirth.getText()));

hat.setHatDepth(Float.parseFloat(hatDepth.getText()));

}

List<Integer> patternValues = pattern.calculateHat(hat);

topHatH.setText(Integer.toString(patternValues.get(0)));

petalW.setText(Integer.toString(patternValues.get(1)));

hatH.setText(Integer.toString(patternValues.get(2)));

hBeforeTopHat.setText(Integer.toString(patternValues.get(3)));

hatW.setText(Integer.toString(patternValues.get(4)));

String expl = "Для создания шапки необходимо:\n";

expl += "1) Набрать "+hatW.getText()+" петель\n";

expl += "2) Провязать "+ hBeforeTopHat.getText() + " рядов до макушки\n";

expl += "3) Разделить изделие на 4 равные части по "+ petalW.getText() +" петель\n";

expl += "4) К двум крайним петлям идут убавки, убавлять нужно в"+

"каждом втором\n ряду по 2 петли пока не останется ";

if (patternValues.get(5) == 8){

expl += "2 петли\n";

}

else{

expl += "3 петли\n";

}

expl += "5) Стянуть оставшиеся петли ("+patternValues.get(5)+" петель)\n";

expl += "6) Сшить изделие верикальным трикотажным швом и постирать, изделие готово\n";

explanations.setStyle("-fx-text-fill: black");

explanations.setText(expl);

}

}

// Метод класса контроллера PatternMittensController

// Вычисление выкройки

@FXML

Продолжение приложения Б

private void calculatePattern() throws IOException {

if (checkAll()) {

pattern.setPatternName(patternName.getText());

pattern.setDensity(dansity.getText());

pattern.setYarn(yarn.getText());

pattern.setDensityVertical(Float.parseFloat(densityVertical.getText()));

pattern.setDensityHorizontal(Float.parseFloat(densityHorizontal.getText()));

pattern.setType("Варежки");

if (cuffsLen.getText() != "") {

mittens.setCuffsLen(Float.parseFloat(cuffsLen.getText()));

mittens.setCuffsThumbLen(Float.parseFloat(cuffsThumbLen.getText()));

mittens.setEndLen(Float.parseFloat(endLen.getText()));

mittens.setPalmGirth(Float.parseFloat(palmGirth.getText()));

mittens.setPinkyLen(Float.parseFloat(pinkyLen.getText()));

mittens.setThumbGirch(Float.parseFloat(thumbGirch.getText()));

mittens.setThumbLen(Float.parseFloat(thumbLen.getText()));

} else {

mittens.setCuffsThumbLen(Float.parseFloat(cuffsThumbLen.getText()));

mittens.setEndLen(Float.parseFloat(endLen.getText()));

mittens.setPalmGirth(Float.parseFloat(palmGirth.getText()));

mittens.setPinkyLen(Float.parseFloat(pinkyLen.getText()));

mittens.setThumbGirch(Float.parseFloat(thumbGirch.getText()));

mittens.setThumbLen(Float.parseFloat(thumbLen.getText()));

}

List<Integer> patternValues = pattern.calculateMittens(mittens);

countCollect.setText(Integer.toString(patternValues.get(0)));

heelH.setText(Integer.toString(patternValues.get(1)));

tipH.setText(Integer.toString(patternValues.get(2)));

sidesW.setText(Integer.toString(patternValues.get(3)));

tipW.setText(Integer.toString(patternValues.get(4)));

halfMittensW.setText(Integer.toString(patternValues.get(5)));

bFH.setText(Integer.toString(patternValues.get(6)));

bFW.setText(Integer.toString(patternValues.get(7)));

beforeTipH.setText(Integer.toString(patternValues.get(8)));

betweenCuffsBFH.setText(Integer.toString(patternValues.get(9)));

cuffsH.setText(Integer.toString(patternValues.get(10)));

mittensW.setText(Integer.toString(patternValues.get(11)));

String expl = "Для создания левой варежки необходимо:\n";

expl += "1) Набрать "+mittensW.getText()+" петель\n";

expl += "2) Провязать "+ cuffsH.getText() + " рядов резинки(или лицевой глади)\n";

expl += "3) Провязать "+ betweenCuffsBFH.getText() + " рядов лицевой гладью до большого пальца\n";

expl += "4) Выделить"+(int)(patternValues.get(7)/2)+"\n крайних петель слева, "+

"набрать петель чтобы получилось \n"+bFW.getText()+" петель \n";

Продолжение приложения Б

expl += "5) Провязать "+bFH.getText()+"рядов, потом убавлять каждые две петли объединить,\n повторять пока не останется"+

countCollect.getText()+ " петель, т.е. провязать с убавками \n"+heelH.getText()+" ряд(ов)\n";

expl += "6) Петли которые были набраны навесить на место крайних слева\n";

expl += "7) Провязать "+ beforeTipH.getText() + " рядов до мыска\n";

expl += "4) Разделить изделие на 2 равные части по "+ halfMittensW.getText() +" петель\n";

expl += "5) Левую половину мысленно разделить на три часть соответственно по "

+sidesW.getText()+",\n "+tipW.getText()+", "+sidesW.getText()+" петель\n";

expl += "6) Работаем с правой стороной и вывязываем мысок частичным вязанием \n(укороченными рядами): "+

"Допустим каретка(нить) слева. Выводить из работы \n поочередно (справа потом слева) с каждой стороны по 2 петли \n"+

"в первых двух рядах далее по 1 петле в каждом ряду пока не останестя \n"+tipW.getText()+" петель в работе."+

"Далее поочередно вводим иглы\n в работу по 1(последние 2 ряда по 2) петле с каждой стороны пока \nневернем в работу все петли\n";

expl += "7) Сшиваем левую половину с правой (открытые части) горизонтальным \nтрикотажным швом,"+

" боковые части сшиваем вертикальным трикотажным швом,\n постирать, изделие готово\n";

expl += "Чтобы связать правую варежку на шаге 4 выделить петли справа\n";

explanations.setStyle("-fx-text-fill: black");

explanations.setText(expl);

}

}

// Метод класса контроллера PatternMittensController

// Вычисление выкройки

@FXML

private void calculatePattern() throws IOException {

if (checkAll()) {

pattern.setPatternName(patternName.getText());

pattern.setDensity(dansity.getText());

pattern.setYarn(yarn.getText());

pattern.setDensityVertical(Float.parseFloat(densityVertical.getText()));

pattern.setDensityHorizontal(Float.parseFloat(densityHorizontal.getText()));

pattern.setType("Носки");

if (cuffsLen.getText() != "") {

socks.setFootGirth(Float.parseFloat(footGirth.getText()));

socks.setFootLen(Float.parseFloat(footLen.getText()));

socks.setCuffsLen(Float.parseFloat(cuffsLen.getText()));

} else {

socks.setFootGirth(Float.parseFloat(footGirth.getText()));

socks.setFootLen(Float.parseFloat(footLen.getText()));

}

Продолжение приложения Б

List<Integer> patternValues = pattern.calculateSocks(socks);

sidesW.setText(Integer.toString(patternValues.get(0)));

tipW.setText(Integer.toString(patternValues.get(1)));

halfSocksW.setText(Integer.toString(patternValues.get(2)));

socksLength.setText(Integer.toString(patternValues.get(3)));

heelH.setText(Integer.toString(patternValues.get(4)));

bHeelH.setText(Integer.toString(patternValues.get(5)));

beforeHeelH.setText(Integer.toString(patternValues.get(6)));

cuffsH.setText(Integer.toString(patternValues.get(7)));

socksW.setText(Integer.toString(patternValues.get(8)));

String expl = "Для создания левого носка необходимо:\n";

expl += "1) Набрать "+socksW.getText()+" петель\n";

expl += "2) Провязать "+ cuffsH.getText() + " рядов резинки(или лицевой глади)\n";

expl += "3) Провязать "+ beforeHeelH.getText() + " рядов лицевой гладью до пятки\n";

expl += "4) Разделить изделие на 2 равные части по "+ halfSocksW.getText() +" петель\n";

expl += "5) Левую половину мысленно разделить на три часть соответственно по "

+sidesW.getText()+",\n "+tipW.getText()+", "+sidesW.getText()+" петель\n";

expl += "6) Работаем с правой стороной и вывязываем пятку(мысок) частичным вязанием\n (укороченными рядами): "+

"Допустим каретка(нить) слева. Выводить из работы поочередно\n(справа потом слева) с каждой стороны по 2 петли "+

"в первых двух рядах далее по 1 петле в каждом\n ряду пока не останестя "+tipW.getText()+" петель в работе."+

"Далее поочередно вводим иглы \nв работу по 1(последние 2 ряда по 2) петле с каждой стороны пока не вернем в работу все петли\n";

expl += "7) Когда довязали пятку, вводим в pаботу правую половину петель и вяжем \n"+

socksLength.getText()+" рядов до мыска\n";

expl += "8) Мысок провязываем аналогично пятке(5-6)\n";

expl += "9) Сшиваем левую половину с правой (открытые части) горизонтальным\n трикотажным швом,"+

" боковые части сшиваем вертикальным трикотажным швом,\n постирать, изделие готово\n";

expl += "Чтобы связать правый носок пятку и мысок вывязоваем справа\n";

explanations.setStyle("-fx-text-fill: black");

explanations.setText(expl);

}

}

// Класс реализующий соединение с базой данных

public class DBConnector {

private static DBConnector dbconnector = null;

private static Connection connection = null;

private static final String JDBC\_DRIVER = "com.mysql.cj.jdbc.Driver";

private static final String DATABASE\_URL = "jdbc:mysql://localhost:3306/crochet\_assistent";

Продолжение приложения Б

private static final String USER = "root";

private static final String PASSWORD = "MySQLPassword";

private DBConnector(){

try {

connection = DriverManager.getConnection(DATABASE\_URL, USER, PASSWORD);

} catch (SQLException e) {

System.out.println("Проблемы при подключении к базе данных");

e.printStackTrace();

}

}

public static Connection getInstanse()

{

if(dbconnector == null)

{

dbconnector = new DBConnector();

}

return connection;

}

}

// Методы класса Pattern для сохранения данных в базу и чтения их

// добавление новой выкройки

public boolean insertnew(String userLogin) {

Connection connection = DBConnector.getInstanse();

PreparedStatement preparedStatement = null;

try {

String SQL = "INSERT INTO pattern(typeClothes, patternName, density, yarn, densityVertical, densityHorizontal, userLogin) VALUES (?, ?, ?, ?, ?, ?, ?); ";

preparedStatement = connection.prepareStatement(SQL);

preparedStatement.setString(1, type);

preparedStatement.setString(2, patternName);

preparedStatement.setString(3, density);

preparedStatement.setString(4, yarn);

preparedStatement.setFloat(5, densityVertical);

preparedStatement.setFloat(6, densityHorizontal);

preparedStatement.setString(7, userLogin);

if (preparedStatement.executeUpdate() == 1) {

SQL = "SELECT max(id) as lastpat FROM pattern";

preparedStatement = connection.prepareStatement(SQL);

ResultSet rs = preparedStatement.executeQuery();

if (rs.next()) {

idPat = rs.getInt("lastpat");

}

return true;

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

if (preparedStatement != null) {

Продолжение приложения Б

try {

preparedStatement.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

return false;

}

// Поиск выкройки по id

public boolean findPattern(int id) {

Connection connection = DBConnector.getInstanse();

PreparedStatement preparedStatement = null;

String SQL = "SELECT \* FROM crochet\_assistent.pattern WHERE id = " + id + ";";

try {

preparedStatement = connection.prepareStatement(SQL);

ResultSet resultSet = preparedStatement.executeQuery(SQL);

if (resultSet.next()) {

idPat = id;

type = resultSet.getString("typeClothes");

patternName = resultSet.getString("patternName");

idPat = resultSet.getInt("id");

density = resultSet.getString("density");

yarn = resultSet.getString("yarn");

densityVertical = resultSet.getFloat("densityVertical");

densityHorizontal = resultSet.getFloat("densityHorizontal");

return true;

} else {

return false;

}

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

// Изменение данных в базе

public boolean chengePattern() {

Connection connection = DBConnector.getInstanse();

PreparedStatement preparedStatement = null;

String SQL = "UPDATE crochet\_assistent.pattern SET typeClothes = '" + type + "', patternName = '" + patternName

+ "', density = '" + density + "', yarn = '" + yarn + "', densityVertical = " + densityVertical

+ ", densityHorizontal = " + densityHorizontal + " WHERE id = " + idPat + " ";

try {

preparedStatement = connection.prepareStatement(SQL);

if (preparedStatement.executeUpdate(SQL) != 0) {

return true;

} else {

return false;

Продолжение приложения Б

}

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

// Удаление выкройки

public static boolean deletePattern(int id) {

Connection connection = DBConnector.getInstanse();

PreparedStatement preparedStatement = null;

String SQL = "DELETE FROM crochet\_assistent.pattern WHERE id = " + id + " ";

try {

preparedStatement = connection.prepareStatement(SQL);

if (preparedStatement.executeUpdate(SQL) != 0) {

System.out.println("Пользователь удалил выкройку");

return true;

} else {

return false;

}

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

// Обработка событий, класс Action

// Новое событие

public static boolean newAction(String nameact, String userlogin) {

Connection connection = DBConnector.getInstanse();

PreparedStatement preparedStatement = null;

try {

String SQL = "INSERT INTO actions(nameact, userlogin) VALUES (?, ?); ";

preparedStatement = connection.prepareStatement(SQL);

preparedStatement.setString(1, nameact);

preparedStatement.setString(2, userlogin);

if (preparedStatement.executeUpdate() == 1) {

return true;

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

if (preparedStatement != null) {

try {

preparedStatement.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

Продолжение приложения Б

}

return false;

}

// Возможные запросы

// "SELECT \* FROM crochet\_assistent.actions;"

// "SELECT \* FROM crochet\_assistent.actions WHERE userlogin = '"+userlogin+"';"

// "SELECT \* FROM crochet\_assistent.actions where dateact > NOW() - interval 1 month;"

// "SELECT \* FROM crochet\_assistent.actions where dateact > NOW() - interval 7 day;"

// "SELECT \* FROM crochet\_assistent.actions where dateact > NOW() - interval 1 day;"

// "SELECT \* FROM crochet\_assistent.actions WHERE nameact like '%" + enterInf + "%' OR userlogin like '%"+enterInf+"%' ;";

// Поиск данных

public static List<Action> find(String SQL) {

Connection connection = DBConnector.getInstanse();

PreparedStatement preparedStatement = null;

List<Action> pats = new ArrayList<Action>();

try {

preparedStatement = connection.prepareStatement(SQL);

ResultSet resultSet = preparedStatement.executeQuery(SQL);

while (resultSet.next()) {

pats.add(new Action(resultSet.getDate("dateact").toString(),resultSet.getTime("timeact").toString(),

resultSet.getString("nameact"), resultSet.getString("userlogin")));

}

} catch (SQLException e) {

e.printStackTrace();

}

return pats;

}

// Очистка журнала событий

public static boolean delete() {

Connection connection = DBConnector.getInstanse();

PreparedStatement preparedStatement = null;

String SQL = "DELETE FROM crochet\_assistent.actions ";

try {

preparedStatement = connection.prepareStatement(SQL);

if (preparedStatement.executeUpdate(SQL) != 0) {

return true;

} else {

return false;

}

} catch (SQLException e) {

e.printStackTrace();

}

return false;

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

-- MySQL Workbench Forward Engineering

CREATE DATABASE if not exists crochet\_assistent;

USE crochet\_assistent;

-- Пользователи

CREATE TABLE if not exists crochet\_assistent.users(

login VARCHAR(50) NOT NULL ,

pasword VARCHAR(50) NOT NULL,

email VARCHAR(50) ,

username VARCHAR(50) ,

blocked INT NOT NULL default 0,

PRIMARY KEY (login)

);

-- События, действия пользователей

CREATE TABLE if not exists crochet\_assistent.actions (

id INT NOT NULL AUTO\_INCREMENT,

dateact date NOT NULL default(CURRENT\_DATE()),

timeact time NOT NULL default(CURRENT\_TIME()),

nameact VARCHAR(255) ,

userlogin VARCHAR(50) NOT NULL,

PRIMARY KEY (id),

CONSTRAINT fk\_action\_user

FOREIGN KEY (userlogin)

REFERENCES crochet\_assistent.users (login)

ON UPDATE CASCADE);

-- Выкройки, общая информация

CREATE TABLE if not exists crochet\_assistent.pattern (

id INT NOT NULL AUTO\_INCREMENT,

typeClothes VARCHAR(50) NOT NULL,

patternName VARCHAR(255) ,

density VARCHAR(255) ,

yarn VARCHAR(255) ,

densityVertical FLOAT NOT NULL,

densityHorizontal FLOAT NOT NULL,

createDate DATE NOT NULL DEFAULT (CURRENT\_DATE()),

userlogin VARCHAR(50) NOT NULL,

PRIMARY KEY (id),

CONSTRAINT fk\_patt\_user

FOREIGN KEY (userlogin)

REFERENCES crochet\_assistent.users (login)

ON DELETE CASCADE

ON UPDATE CASCADE);

-- Мерки для варежек

CREATE TABLE if not exists crochet\_assistent.mittens (

idpat INT NOT NULL,

palmGirth FLOAT NOT NULL,

thumbGirch FLOAT NOT NULL,

cuffsLen FLOAT NOT NULL,

cuffsThumbLen FLOAT NOT NULL,

pinkyLen FLOAT NOT NULL,

endLen FLOAT ,

thumbLen FLOAT NOT NULL,

PRIMARY KEY (idpat),

Продолжение приложения В

CONSTRAINT fk\_mit\_patt

FOREIGN KEY (idpat)

REFERENCES crochet\_assistent.pattern (id)

ON DELETE CASCADE

ON UPDATE CASCADE);

-- Мерки для носков

CREATE TABLE if not exists crochet\_assistent.socks (

idpat INT NOT NULL,

footGirth FLOAT NOT NULL,

footLen FLOAT NOT NULL,

cuffsLen FLOAT NOT NULL,

PRIMARY KEY (idpat),

CONSTRAINT fk\_sock\_patt

FOREIGN KEY (idpat)

REFERENCES crochet\_assistent.pattern (id)

ON DELETE CASCADE

ON UPDATE CASCADE);

-- Мерки для шапки

CREATE TABLE if not exists crochet\_assistent.hat (

idpat INT NOT NULL,

headGirth FLOAT NOT NULL,

hatDepth FLOAT NOT NULL,

lapelLen FLOAT NOT NULL,

PRIMARY KEY (idpat),

CONSTRAINT fk\_hat\_patt

FOREIGN KEY (idpat)

REFERENCES crochet\_assistent.pattern (id)

ON DELETE CASCADE

ON UPDATE CASCADE);