МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Инженерно-экономический факультет

Кафедра экономической информатики

Дисциплина: Программирование сетевых приложений

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Система расчета стоимости изделия предприятия

Выполнил: Мойсеенко Иван Андреевич

Проверил: Петрович Никита Олегович

Минск 2018

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc532246226)

[1 описание систем расчета стоимости изделий 6](#_Toc532246227)

[2 постановка задачи и обзор методов её решения 9](#_Toc532246228)

[2.1 Обзор методов решения поставленной задачи 9](#_Toc532246229)

[2.2 Описание используемых паттернов проектирования 10](#_Toc532246230)

[3 Функциональное моделирование процесса расчета стоимости изделия 13](#_Toc532246231)

[4 Описание информационной системы расчета стоимости изделия 21](#_Toc532246232)

[5 Диаграммы представления и их описание 23](#_Toc532246233)

[5.1 Диаграмма вариантов использования 23](#_Toc532246234)

[5.2 Диаграмма последовательности 23](#_Toc532246235)

[5.3 Диаграмма состояний 23](#_Toc532246236)

[5.4 Диаграмма компонентов 24](#_Toc532246237)

[5.5 Диаграмма развертывания 24](#_Toc532246238)

[5.6 Диаграмма классов 24](#_Toc532246239)

[6 Описание алгоритмов реализующих бизнес-логику системы расчета стоимости изделия 25](#_Toc532246240)

[6.1 Описание алгоритма работы всей программы 25](#_Toc532246241)

[6.2 Описание алгоритма расчета итоговой стоимости 25](#_Toc532246242)

[7 Руководство пользователя 26](#_Toc532246243)

[7.1 Руководство серверной части приложения 26](#_Toc532246244)

[7.2 Руководство клиентской части приложения 27](#_Toc532246245)

[8 Результаты тестирования системы расчета стоимости изделия 34](#_Toc532246246)

[Заключение 35](#_Toc532246247)

[Список использованных источников 36](#_Toc532246248)

[Приложение А (обязательное) 37](#_Toc532246249)

[Приложение Б (обязательное) 38](#_Toc532246250)

[Приложение В (обязательное) 45](#_Toc532246251)

[Приложение Г (обязательное) 47](#_Toc532246252)

[Приложение Д (обязательное) 50](#_Toc532246253)

ВВЕДЕНИЕ

На сегодняшний день информационные технологии – одна из ведущих отраслей деятельности, которая прочно вошла во все сферы человеческой деятельности. Современный мир требует повсеместное внедрение и использование любых технологичных решений, мобильных телефонов, планшетов, ноутбуков и т.д. С ростом использования подобных устройств и появление новых возможностей возникает необходимость все более широкого применение современных технологий.

Главное неудобство в любой работе представляет необходимость ручных расчетов любого вида и ведение сопутствующей документации. Разнообразные бумаги сопровождают основные процессы на предприятии. Ведение документации призвано упорядочить и систематизировать деятельность компании. Несмотря на широкое внедрение технологий, многие компании до сих пор используют в работе бумажную документацию, справочники и т.д. Недостатками такой документации являются большое её количество, неудобство использования (затрудненный поиск, сортировка, любого рода выборка), увеличение временных затрат на складской учет, сложность передачи, хранения, копирования и т.д. Современная система управления данными призвана решить многие вышеназванные проблемы. Эффективная система предполагает рациональное, оптимизированное управление данными, эффективные и быстрые методы расчетов, методы различные методы работы с базами данных. Такая система предусматривает полную или частичную автоматизацию основных процессов. Применение автоматизированной системы для расчета стоимости изделия позволяет сократить временные и финансовые затраты на ведение документации, минимизирует ошибки в расчетах и предоставляет многие возможности для работы с результатами расчетов: вывод в удобной форме, поиск, фильтрация, автоматическое создание отчетов. Именно поэтому разработку программного обеспечения такого плана актуальной на сегодняшний день.

Предполагается использование архитектуры клиент-сервер, которая предъявляет специфические требования как к клиенту, так и к серверу. Программа, удовлетворяющая этим требованиям, может считаться клиент-серверным приложением, выполняющим распределенную обработку данных. Под распределенной обработкой понимается выполнение серверной частью программы запросов клиентской части. Серверная часть приложения обеспечивает хранение данных и их обработку, а клиентская часть лишь передает серверу соответствующие запросы.

Преимущества клиент-серверных систем:

1. Клиент-серверный подход — модульный, причем серверные программные компоненты компактны и автономны.
2. Поскольку каждый компонент выполняется в отдельном защищенном процессе пользовательского режима, сбой сервера не повлияет на остальные компоненты операционной системы.
3. Автономность компонентов делает возможным их выполнение на нескольких процессорах на одном компьютере (симметричная многопроцессорная обработка) или на нескольких компьютерах сети (распределенные вычисления).
4. Обязанность клиента, как правило, — предоставлять пользовательские сервисы и, прежде всего, пользовательский интерфейс, то есть средства для приема, отображения и редактирования данных, введенных пользователем, которые служат основой для запроса серверу. Кроме того, клиент можно настроить на обработку части данных, чтобы уменьшить нагрузку на ресурсы сервера.

Клиент-серверный подход позволяет распределять функции вычислительной системы между несколькими компьютерами в сети, которые не зависят друг от друга. Это в свою очередь снижает требования к клиентским компьютерам, требуя больших вычислительных мощностей только от сервера, что значительно снижает общую стоимость всей сети. При этом все клиенты могут общаются с одним сервером, что позволяет наиболее удобно работать с общими данными, не заботясь о проблемах общения между различными клиентскими компьютерами.

*Цель данного курсового проекта* – оптимизация процесса расчета стоимости изделия, обеспечение мгновенной обработки и систематизированного хранения данных, создание отчетов работы, что в свою очередь приведет к значительному сокращению временных затрат. Также основополагающим фактором при разработке данного приложения является необходимость создание интерфейса, понятного даже для неподготовленного пользователя. Так как данное приложение предполагает замену привычной документации на электронную во многих областях, то логично предположить, что потенциальные пользователи данного приложения не являются продвинутыми пользователями ПК. В виду чего простой и понятный интерфейс не будет требовать больших затрат на переобучение сотрудников, что экономит деньги предприятия и, в свою очередь, повышает привлекательность данного приложения.

Для правильного подхода к разработке приложения для автоматизации расчетов по стоимости изделия следует поставить и решить *следующие задачи:*

1. изучить предметную область работы экономических отделов предприятий;
2. разработать логическую и физическую модель представления данных;
3. разработать функциональную модель расчета стоимости изделия;
4. разработать базу данных, содержащую необходимую информацию для расчетов;
5. реализовать клиент-серверное взаимодействие, позволяющее управлять базой данных и выполнять определённый ряд функций только на серверной части;
6. разработать приложение с удобным и понятным пользовательским интерфейсом;
7. протестировать разработанный программный продукт.

Когда поставленные задачи будут решены в полном объеме, тогда разработанное в данном курсовом проекте приложение позволит любому, даже не опытному пользователю, найти наиболее оптимальный путь решения возникшей проблемы, для дальнейшего анализа и принятия окончательного решения исходя из полученных результатов.

1. ОПИСАНИЕ СИСТЕМ РАСЧЕТА СТОИМОСТИ ИЗДЕЛИЙ

Для уменьшения рутинной работы, связанной с проведением многочисленных расчетов и построения графиков, удобнее пользоваться специальными программами, позволяющими существенно упростить этот процесс.

В процессе финансово-хозяйственной деятельности у предприятия или организации возникают расчетные отношения, отражающие взаимные обязательства, связанные с получением или продажей материальных ценностей, выполнением работ или оказанием услуг, с расчетами по налогам и сборам, с банками по кредитам, а также с другими юридическими и физическими лицами.

Необходимость автоматизации расчетов с дебиторами и кредиторами - неизбежное следствие существующей в настоящее время системы расчетов между юридическими и физическими лицами.

Системы автоматизации инженерных расчетов. Это современные программные средства, которые сегодня не только успешно заменяют ученым и инженерам ручку с тетрадью, логарифмическую линейку, таблицы интегралов и другие математические справочники, но и позволяют во многих случаях обойтись без традиционного программирования прикладных задач, решая их в специально разработанных средах, позволяющих выполнять численные расчеты, аналитические преобразования, анализ данных и графическое представление результатов.

Автоматизация расчетов – это когда пользователь вводит начальные данные, а некоторая программа сама проводит расчеты на основе заложенных в нее алгоритмов и выдает конечный результат пользователю. Автоматизация расчетов помогает избавится от ошибок человека, которые могут быть допущены при расчете, особенно когда это большая и сложная формула. Автоматизация расчетов применяется в автоматических системах, которые разрабатываются для автоматизации рабочего места. Автоматические системы повышают эффективность работы сотрудников, избавляя его от однообразной работы.

Важное значение для организации эффективного управления неструктурированными документами имеют методы хранения информации, навигации, поиска и анализа документов. Для этого ввод информации сопровождается классификацией документов путем задания атрибутов и ключевых слов, аннотированием их содержания. Поиск и фильтрация документов производится по запросам на основе контекстного поиска: по атрибутам, по ключевым словам, и по полному содержанию текста на основе индекса. В процессе жизнедеятельности любой организации создается огромное количество административных, бухгалтерских, нормативных и кадровых бумаг. Для продуктивной работы компании чрезвычайно важен их строгий учет и продуманное хранение. Ситуация осложняется тем, что необходимо учитывать и хранить не только внутренние бумаги компании, но и созданные за ее пределами. Таким образом, суммированный объем информации, который необходимо обрабатывать, структурировать и систематизировать, чрезвычайно велик. Переход от бумажного документооборота к электронному обеспечен современным уровнем развития технологий и обусловлен логикой развития информационного общества. Наиболее важной практической причиной перехода к электронной обработке документов стала необходимость оптимизации поиска необходимой информации и уменьшения времени на ее анализ.

Автоматизация бизнес-процессов – это перевод типовых бизнес-задач и стандартных операций под контроль программно-аппаратного комплекса. В результате высвобождаются ресурсы, что позволяет увеличить производительность труда и эффективность стратегического управления. Для оптимизации работы организации используют системы моделирования и оптимизации исполнения бизнес-процессов.

Автоматизация бизнес-процессов является важным условием успеха организации любых масштабов, независимо от сферы деятельности или структуры. Быстро меняющаяся экономическая ситуация, необходимость снижения рисков, высокие скорости принятия решений – все это требует современных подходов. Благодаря использованию автоматики появляется возможность высвободить ценные ресурсы и сосредоточить максимум усилий на ключевых направлениях деятельности.

Существует ряд симптомов, которые указывают на необходимость комплексной автоматизации бизнес-процессов компании:

Фирма работает с перебоями, сотрудники не могут координировать действия и выполнять их слаженно;

1. в компании имеют место регулярные задержки, искажения и потери информации;
2. при выполнении повседневной работы специалисты компании не могут отодвигать собственные интересы на второй план;
3. задачи фирмы выполняются неэффективно, клиентоориентированность развита очень слабо;
4. у сотрудников нет четкого понимания того, как их деятельность сказывается на показателях организации;
5. нет явного распределения, кто из сотрудников отвечает за конкретные цели и функции предприятия;
6. отсутствуют временные рамки для выполнения задач; Компания не всегда работает в интересах клиентов.

Пример результатов автоматизации можно увидеть на рисунке 1.1



Рисунок 1.1 – Результат автоматизации

Таким образом, повсеместное внедрение автоматизации расчетов и электронных баз данных позволит существенно сократить расходы компании, минимизировать возможные ошибки и повысить общую эффективность работы.

1. ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЁ РЕШЕНИЯ

## 2.1 Обзор методов решения поставленной задачи

Проанализировав предметную область, мы можем сказать, что разработка данной системы актуальна и может найти широкое применение.

Целью разработки автоматизированной системы является повышение эффективности работы предприятия.

Задачами разработки приложения являются:

* приложение представляет собой архитектуру клиент-сервер;
* серверная часть приложения является многопоточной, что предусматривает возможность параллельной обработки запросов;
* бизнес-логика системы должна быть реализована только на серверной части приложения. Сервер представляет возможности вычисления и организует взаимодействие с БД;
* в разрабатываемом приложении обеспечить: добавление, удаление, редактирование и записей из базы данных, сохранение табличных результатов в файле (создание текстового отчета), предоставление пользователю аналитической информации (графики, диаграммы);
* построить функциональную модель системы в IDEF0;
* в консольном окне серверной части должна отображаться текущая статистика подключений и параметры запуска сервера;
* предусмотреть механизм авторизации пользователей;
* информационная система моделируется, опираясь на стандарт UML;

Для выполнения данных задач будет использован объектно-ориентированный язык программирования Java. Для параллельного доступа к данным на сервере будут использованы сокеты и потоки. Для хранения и обработки данных будет использован SQL.

Работа приложения основывается на том, что клиентская часть реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него, в то время как серверная часть получает запрос от клиента, выполняет вычисления, после чего формирует ответ и отправляет её клиенту. Выполнение всех описанных сетевых функций приложения без проблем реализуется с помощью Java, так как сетевая работа является одной из сильных сторон данного языка программирования.

В Java имеется изящный и вместе с тем сложный механизм реализации взаимодействия потоков, он позволяет программисту сосредоточиться непосредственно на задаче − без раздумий о том, как бы организовать взаимодействие потоков самому.

Для реализации хранения, обработки и дальнейшего использования информации в данном приложении используется СУБД MySQL.

 MySQL − компактный многопоточный сервер баз данных. MySQL отличатся хорошей скоростью работы, надежностью, гибкостью. Работа с ней, как правило, не вызывает больших трудностей.

Для общения с СУБД MySQL применяется язык SQL (Stuctured Query Language − язык структурированных запросов). В настоящее время SQL является стандартом работы с базами данных, и все основные СУБД понимают его. SQL включает много разных типов операторов, разработанных для взаимодействия с базами данных.

Таким образом, основное достоинство базы данных, в том числе и используемой СУБД MySQL, заключается в том, что она берёт на себя всю работу с жёстким диском и делает это очень эффективно. [1]

## Описание используемых паттернов проектирования

В данном приложении используются следующие паттерны: Singleton и Adapter. Рассмотрим используемые паттерны ниже.

Паттерн Singltone (одиночка) необходим, когда некая сущность должна существовать в единственном экземпляре. То есть должно быть невозможно создание нескольких экземпляров класса этого же типа. Паттерн Singleton предоставляет такие возможности.

Архитектура паттерна Singleton основана на идее использования глобальной переменной, имеющей следующие важные свойства:

1. Такая переменная доступна всегда. Время жизни глобальной переменной − от запуска программы до ее завершения.
2. Предоставляет глобальный доступ, то есть, такая переменная может быть доступна из любой части программ. [2]

UML диаграмма данного паттерна представлена на рисунке 2.1.

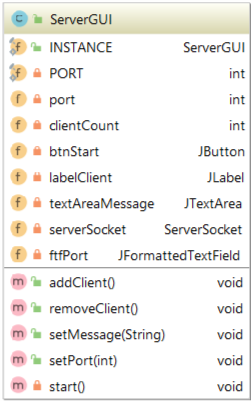


Рисунок 2.1 – UML диаграмма паттерна Singleton

В данном приложении этот паттерн необходим при работе серверной части. Так как интерфейс сервера, даже при наличии нескольких подключений клиентов, которые работают в разных потоках, должен существовать в единственном экземпляре, а не дублироваться при каждом новом подключении, паттерн Singletone является оптимальным решением для данной проблемы. Доступ к этому объекту осуществляется через поле класса, объявленное статическим, это поле является ссылкой на данный класс. Этот объект будет создан только при первом обращении по ссылке, а все последующие вызовы просто возвратят его адрес. Для обеспечения уникальности объекта, конструкторы и оператор присваивания объявляются закрытыми.

Следующий используемый шаблон проектирования – это Adapter. Данный паттерн позволяет работать вместе объектам с несовместимым интерфейсом. [3] Данный паттерн был необходим, в момент когда было принято использовать стороннюю библиотеку для замены уже встроенного поля ввода даты, на более удобное. Так как интерфейс был подготовлен под стандартные функции Java Swing, то встала следующая задача: библиотека поддерживала необходимые данные и поведение, но имела неподходящий интерфейс. Паттерн Адаптер решает эту проблему путем создания класса-оболочки. [4] UML диаграмма данного шаблона представлена на рисунке 2.2.

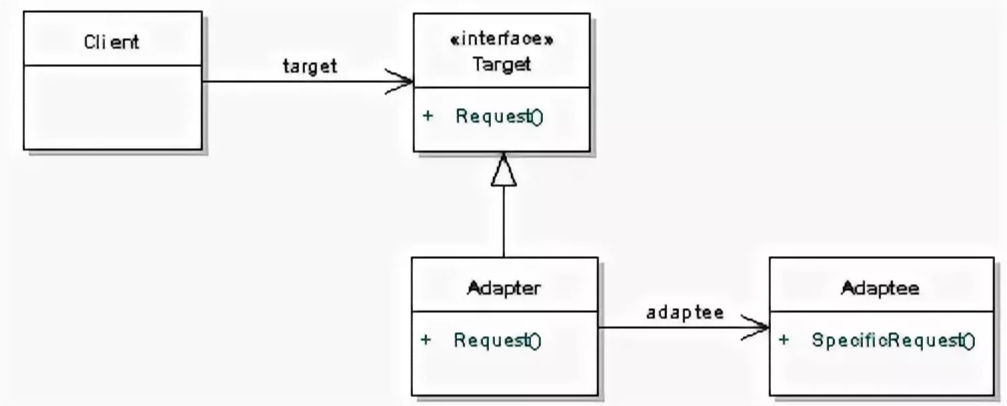


Рисунок 2.2 – UML диаграмма паттерна Adapter

Таким образом, использование шаблона Adapter, позволило включить существующий библиотечный объект в существующую структуру, невзирая на различие в интерфейсе, а просто приведя интерфейс к необходимому виду.

1. ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРОЦЕССА РАСЧЕТА СТОИМОСТИ ИЗДЕЛИЯ

Постоянное усложнение производственно-технических и организационно экономических систем – фирм, предприятий, производств, и др. субъектов производственно-хозяйственной деятельности – и необходимость их анализа с целью совершенствования функционирования и повышения эффективности обусловливают необходимость применения специальных средств описания и анализа таких систем. Для проведения анализа бизнес-процессов используется, в частности, программный продукт AllFusion Modelling Suite, включающий пакет AllFusion Process Modeler (BPWin).

IDEF0 – используется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающие эти функции. [5]

IDEF0 – совокупность блоков, каждый из которых является своеобразным «черным ящиком» с входами и выходами, управлением и механизмами, которые детализируются (декомпозируются) до необходимого уровня. Наиболее важная функция расположена в верхнем левом углу. А соединяются функции между собой при помощи стрелок и описаний функциональных блоков. При этом каждый вид стрелки или активности имеет собственное значение. Данная модель позволяет описать все основные виды процессов, как административные, так и организационные.

Стрелки могут быть:

* Входящие – вводные, которые ставят определенную задачу.
* Исходящие – выводящие результат деятельности.
* Управляющие (сверху вниз) – механизмы управления (положения, инструкции и пр).
* Механизмы (снизу-вверх) – что используется для того, чтобы произвести необходимую работу.

Далее на рисунке 3.1 приведена диаграмма верхнего уровня «Автоматизировать систему расчета стоимости изделия». На входе имеем различного рода входную информацию о:

* стоимости компонентов;
* транспортных расходах;
* зарплатах сотрудников;
* стоимости оборудования;
* количество затраченной электроэнергии;
* информация из внешних источников.

Выходным параметром всей системы будет рассчитанная стоимость изделия. Управляющим механизмом данной системы является финансист и бухгалтер, для облегчения деятельности которых и разрабатывается данный продукт.

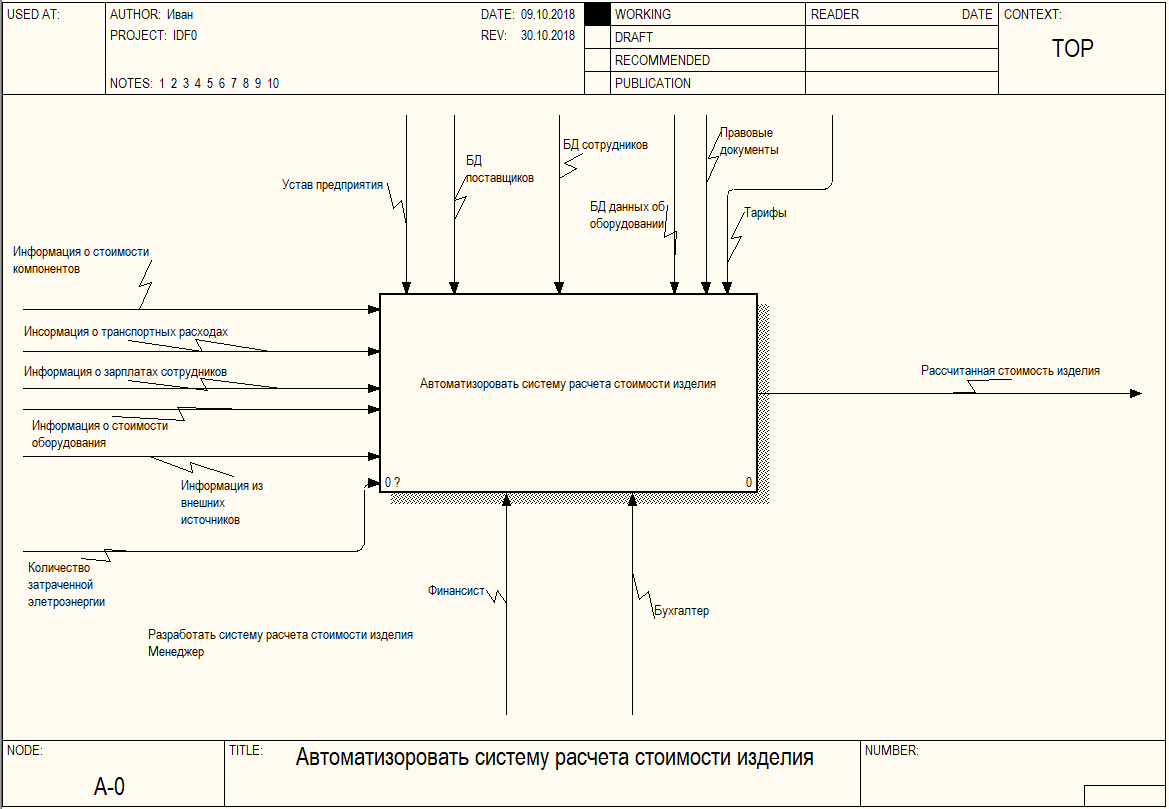


Рисунок 3.1 – Контекстная диаграмма верхнего уровня

Процесс деятельности по автоматизации расчетов можно разделить на 4 процесса: «Рассчитать стоимость комплектующих», «Рассчитать стоимость производства», «Вычислить сумму зарплатных отчислений», «Рассчитать суммарную стоимость изделия» (рисунок 3.2).

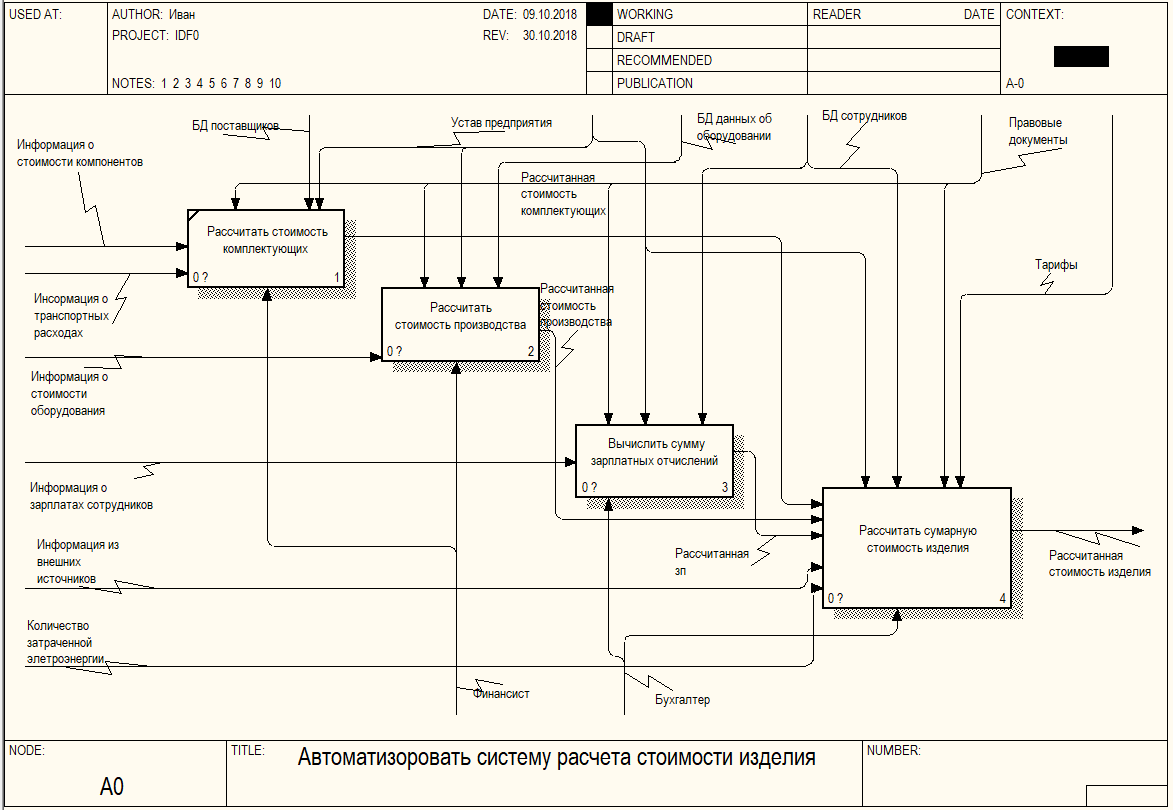


Рисунок 3.2 – Декомпозиция первого уровня

Далее рассмотрим декомпозицию блока «Расчета стоимость производства» (рисунок 3.3). Управление данным процессом осуществляют правовые документы, устав предприятия и БД данных об оборудовании. На вход поступает информация о стоимости оборудования для произведения расчетов. На выходе получаем рассчитанную стоимость производства.

Процесс «Расчёта стоимости производства» состоит из следующих подпроцессов:

1. «Расчет суммы амортизационных отчислений»;
2. «Расчет суммы выплат за приобретенное оборудование»;
3. «Расчет итоговой стоимости отчислений на оборудование».

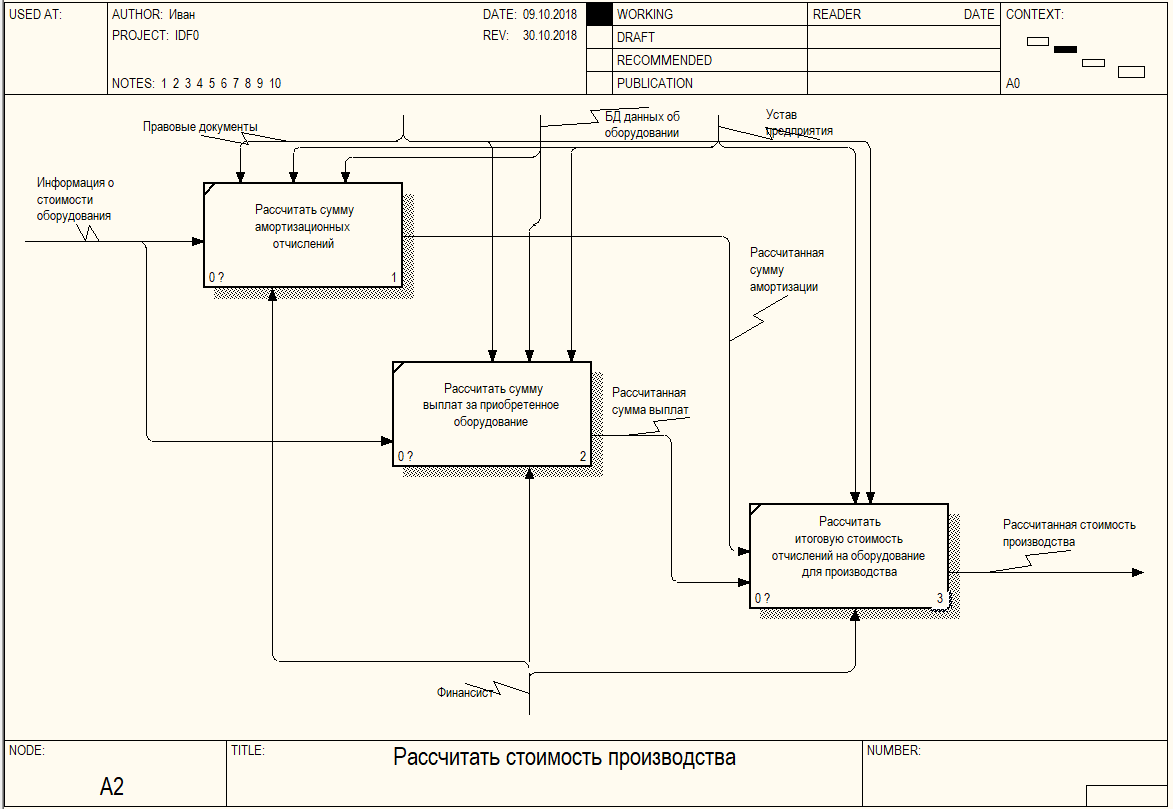


Рисунок 3.3 – Декомпозиция блока «Рассчитать стоимость производства»

Следующий блок, который мы рассмотрим – это блок «Вычисления суммы зарплатных отчислений» (рисунок 3.4). Управление данным процессом осуществляют правовые документы, устав предприятия и БД данных сотрудников. На вход поступает информация о стоимости зарплатах сотрудников для произведения расчетов. На выходе получаем рассчитанную заработную плату.

Процесс «Вычисления суммы зарплатных отчислений» состоит из следующих подпроцессов:

1. «Расчет суммы заработка»;
2. «Начисление премии»;
3. «Расчет отчислений в страховой фонд».

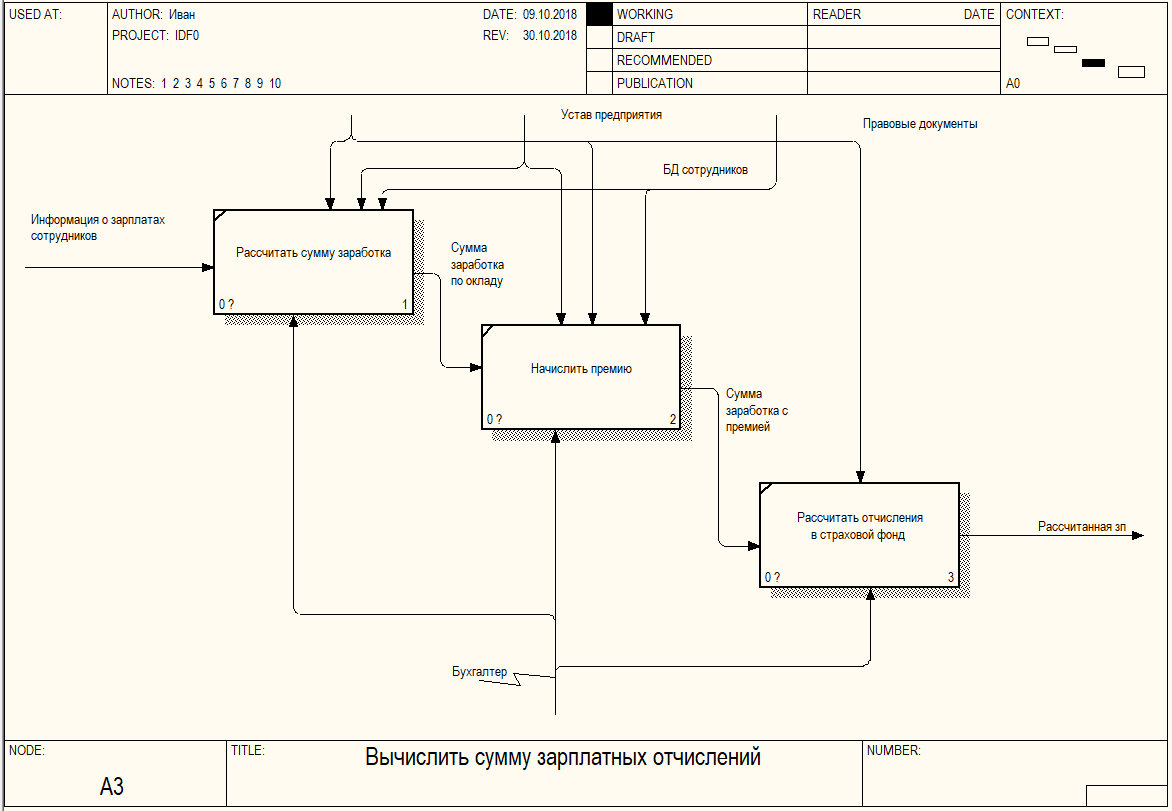


Рисунок 3.4 – Декомпозиция блока «Вычислить сумму зарплатных отчислений»

Следующий блок для рассмотрения – «Расчет суммарной стоимости изделия» (рисунок 3.5). Управление данным процессом осуществляют правовые документы, устав предприятия, БД данных сотрудников и тарифы для проведения расчетов. На вход поступает различная информация: информация из внешних источников, количество затраченной электроэнергии на производство, рассчитанная на предыдущем шаге заработная плата, рассчитанная стоимость производства, рассчитанная стоимость комплектующих. На выходе получаем конечную рассчитанную стоимость изделия.

Процесс «Расчета суммарной стоимости изделия» состоит из следующих подпроцессов:

1. «Расчет суммы дополнительных отчислений»;
2. «Расчет суммы отчислений государству»;
3. «Расчет итоговой стоимости».

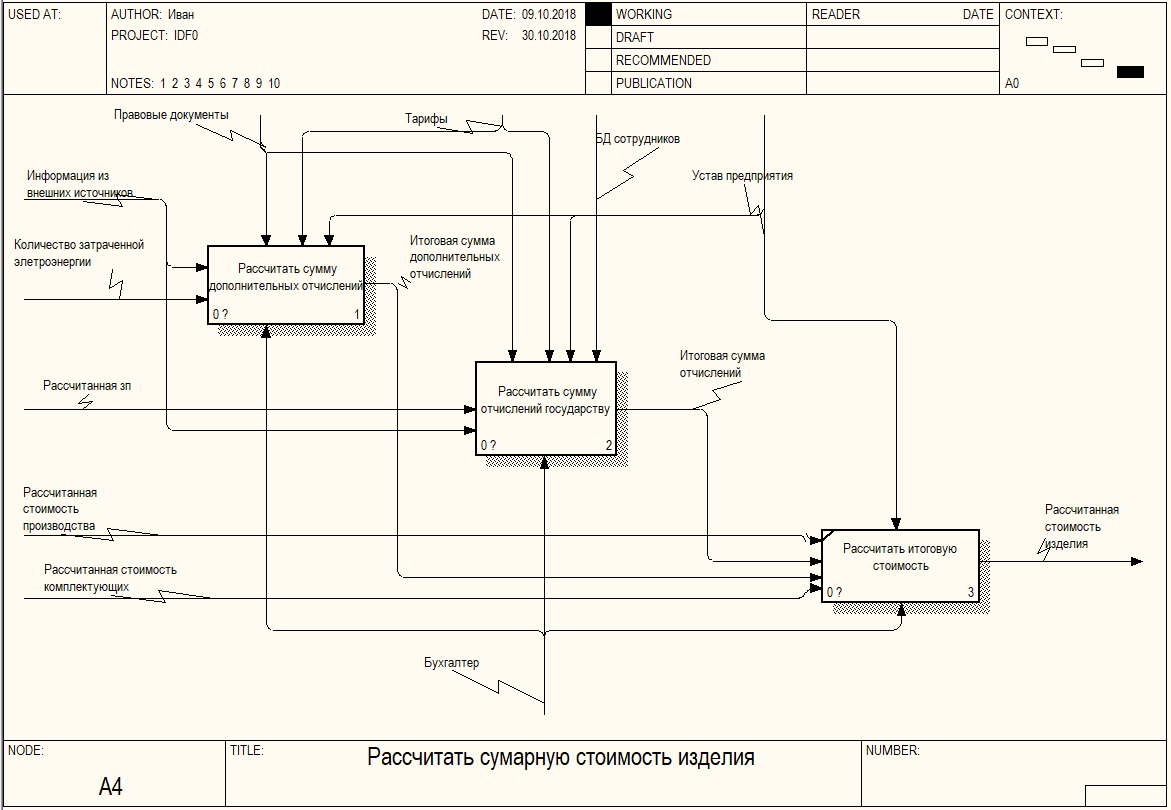


Рисунок 3.5 – Декомпозиция блока «Рассчитать суммарную стоимость изделия»

Также рассмотрим декомпозицию следующего уровня. Блок «Расчета суммы дополнительных отчислений» (рисунок 3.6). Управление данным процессом осуществляют правовые документы, устав предприятия и тарифы для проведения расчетов. На вход поступает информация из внешних источников и количество затраченной электроэнергии на производство. На выходе получаем итоговую сумму дополнительных отчислений.

Процесс «Расчета суммы дополнительных отчислений» состоит из следующих подпроцессов:

1. «Расчет затрат на страховку»;
2. «Расчет затрат на электроэнергию»;
3. «Расчет итоговой суммы дополнительных отчислений».

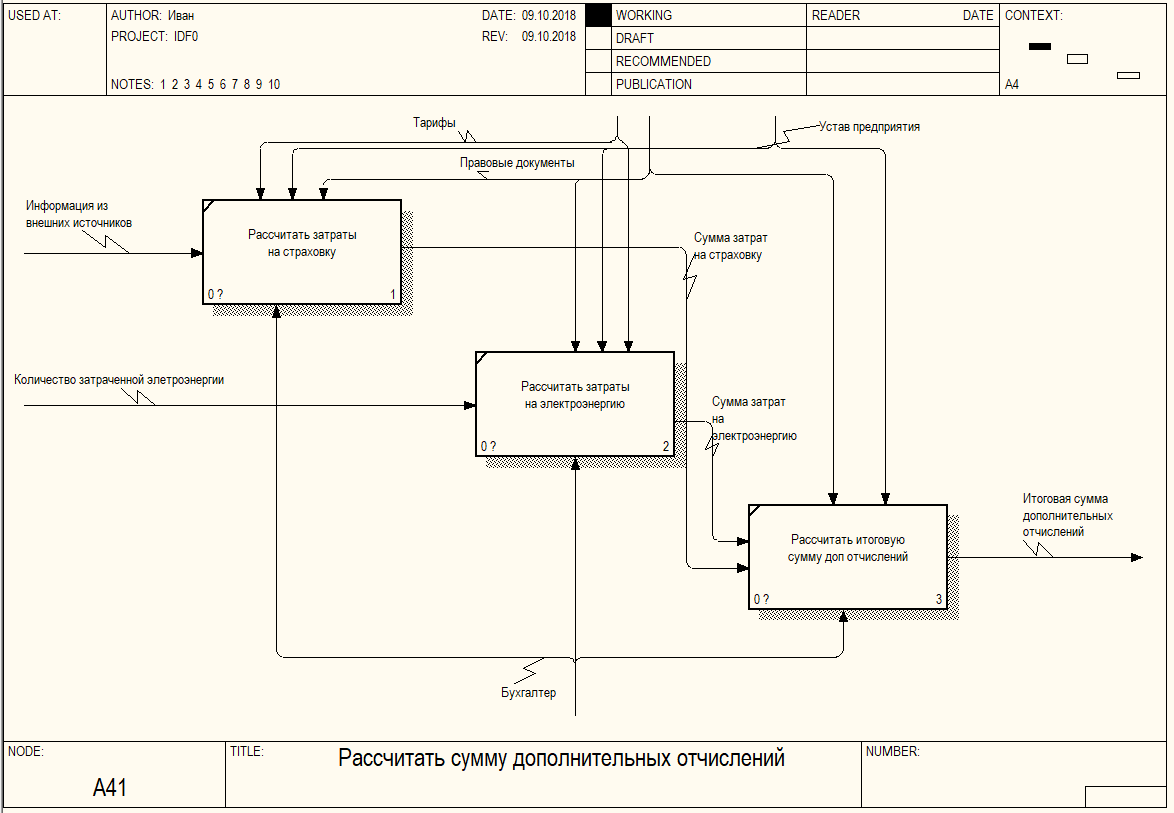


Рисунок 3.6 – Декомпозиция блока «Рассчитать сумму дополнительных отчислений»

Последний блок для рассмотрения – «Расчет суммы отчислений государству» (рисунок 3.7). Управление данным процессом осуществляют правовые документы, устав предприятия, БД данных сотрудников и тарифы для проведения расчетов. На вход поступает информация из внешних источников и рассчитанная в предыдущих шагах заработная плата. На выходе получаем конечную сумму отчислений.

Процесс «Расчета суммы отчислений государству» состоит из следующих подпроцессов:

1. «Расчет суммы пенсионных отчислений»;
2. «Расчет суммы социального страхования»;
3. «Расчет итоговой суммы отчислений».

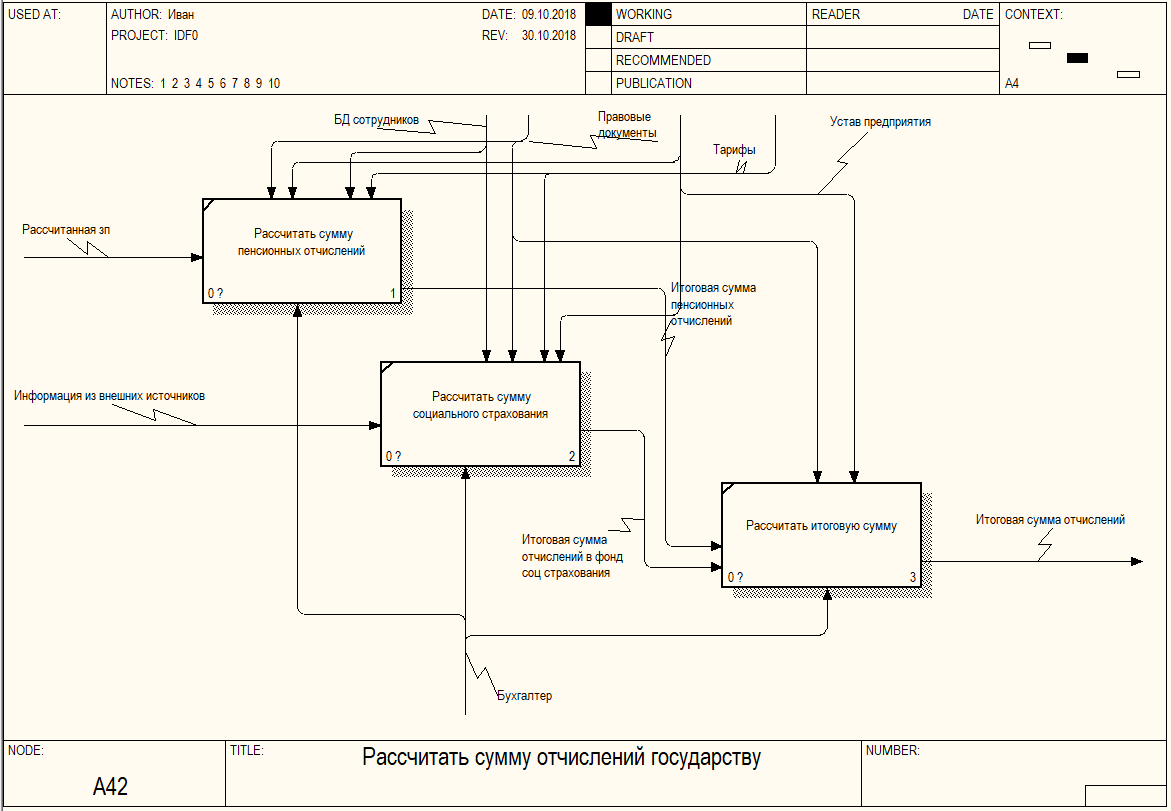


Рисунок 3.7 – Декомпозиция блока «Рассчитать сумму отчислений государству»

1. ОПИСАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ РАСЧЕТА СТОИМОСТИ ИЗДЕЛИЯ

На этапе проектирования информационной модели системы расчета стоимости изделия были выделены следующие сущности:

1. пользователи;
2. персонал;
3. производство;
4. сырье;
5. отчет.

Модель в графическом виде представлена на диаграмме А.1 приложения А.

Первая сущность «Пользователи». При попытке входа в систему каждый пользователь должен пройти авторизацию. В зависимости от введенных данных у него будет определена его роль пользователя или администратора. Данная сущность включает в себя следующие атрибуты:

1. id, хранит уникальный идентификатор пользователя;
2. login, хранит логин пользователя;
3. password, хранит пароль пользователя;
4. role, хранит роль (администратор или пользователь);
5. name, хранит имя пользователя;
6. lastname, хранит фамилию пользователя;
7. phone, ханит телефон пользователя;

Следующая сущность «Персонал» содержит информацию об общих сведениях затрат на персонал на конкретное время:

1. id, хранит уникальный идентификатор единицы данных о персонале;
2. salary, хранит общую зарплату, выделенную персоналу за производство;
3. government, хранит процент необходимых отчислений в фонд государства;
4. date, хранит дату внесения данных.

Сущность «Производство», содержит данные о производственных расходах.

1. id, хранит уникальный идентификатор единицы данных о производственных расходах;
2. energy, хранит количество затраченной электроэнергии при производстве;
3. tariff, хранит стоимость за 1к/вт энергии при производстве;
4. date, дата внесения данных о производстве.

Сущность «Сырье», хранит информацию об общих затратах на ресурсы при производстве:

1. id, хранит уникальный идентификатор единицы данных о сырье;
2. name, название набора сырья;
3. count, количество использованного сырья;
4. price, цена за набор сырья.

Главная сущность «Отчет» используется для хранения итоговых данных по расчету стоимости изделия:

1. id, хранит уникальный идентификатор отчета;
2. date, хранит дату создания отчета;
3. result, хранит итоговую сумму расчетов;
4. id\_user, уникальный идентификатор пользователя, производившего расчет;
5. id\_staff, уникальный идентификатор данных о персонале используемых для расчета;
6. id\_rawpackage, уникальный идентификатор данных о сырье используемых для расчета;
7. id\_prodaction, уникальный идентификатор данных о производстве данного продукта;
8. total\_staff, итоговая рассчитанная сумма затрат на персонал;
9. total\_rawpackage, иторовая рассчитанная сумма затрат на сырье;
10. total\_prodaction, итоговая рассчитанная сумма затрат на производство;

Результат моделирования – разработанная информационная модель, оптимально соответствующая требованиям системы, приведенная к третьей нормальной форме, это значит исключена зависимость не ключевых полей, от других не ключевых полей, что позволяет значительно снижать избыточность данных.

1. ДИАГРАММЫ ПРЕДСТАВЛЕНИЯ И ИХ ОПИСАНИЕ

## 5.1 Диаграмма вариантов использования

В приложении Б отображены диаграммы представления.

На рисунке Б.1 приведена диаграмма вариантов использования, производящей расчет стоимости изделия предприятия. Данная система содержит две основные роли администратора и пользователя.

Основные функции администратора:

1. просмотр любых данных в табличной форме;
2. работа с данными персонала;
3. работа с данными сырья;
4. работа с данными поставщиков;
5. работа с данными пользователя, что включает в себя возможности по восстановлению доступа к аккаунту, удаление записи пользователя, регистрация нового пользователя, возможность по смене активной роли пользователя;
6. робота с данными для расчета, в свою очередь включающая в себя расчет конечной стоимости изделия, для чего необходимо выбрать данные: основные и дополнительные;

Общие функции администратора и пользователя:

1. поиск и фильтрация данных по расчетам;
2. просмотр данных по расчетам с возможностями просмотра данных в табличной форме, сохранение файлового отчета, просмотр результатов на графике с возможностями сохранения диаграммы в виде картинки и вывода её на печать.

## 5.2 Диаграмма последовательности

Диаграмма последовательностей представлена на рисунке Б.2. Участником процесса является администратор. Администратор осуществляет запрос на добавление нового пользователя и осуществляет ввод данных о нем. Далее введенные данные проверяются и при правильном вводе данные передаются на сервер. После этого происходит запрос в базу данных, при успешной обработке данных происходит добавление нового пользователя. В результате администратору отправляется сообщение об результате операции.

## 5.3 Диаграмма состояний

Диаграмма состояний представлена на рисунке Б.3. Эта диаграмма описывает состояние процесса регистрации нового пользователя. В начале программа ожидает ввода данных, данные проверяются на валидность, проверяются на уникальность логина. При отсутствии конфликтов и ошибок будет добавлен новый пользователь, иначе будет выведено сообщение об ошибке.

## 5.4 Диаграмма компонентов

Для представления программных компонентов, зависимостей между ними служит диаграмма компонентов, которая представлена на рисунке Б.4.

## 5.5 Диаграмма развертывания

Данная диаграмма показывает элементы на этапе исполнения. Клиент и сервер взаимодействуют по протоколу TCP/IP. Сервер с базой данных соединяется по локальной сети используя драйвер JDBC.

## 5.6 Диаграмма классов

На диаграмме классов отображены отношения между классами и интерфейсами. На каждую сущность в базе данных есть свой класс с соответствующим названием. Классы для работы с базой данных, обеспечивающие соединение и выполнение различных запросов. Классы для обеспечения соединения с сервером и его многопоточной работы. Диаграммы классов представлены на рисунках Б.6 и Б.7.

1. ОПИСАНИЕ АЛГОРИТМОВ РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СИСТЕМЫ РАСЧЕТА СТОИМОСТИ ИЗДЕЛИЯ

## 6.1 Описание алгоритма работы всей программы

В приложении В представлена блок-схема работы всей программы на рисунке В.1.

Программа продолжает свою работу пока не будет получен сигнал о выходе. Далее программа пытается соединится с сервером, при неудачном соединении пользователь видит сообщение об ошибке и имеет возможность повторного переподключения. Если соединение прошло удачно, то пользователю необходимо авторизироваться. При удачной авторизации, в зависимости от роли пользователя, мы попадаем соответственно в меню пользователя или администратора. Для администратора доступны функции по работе с данными производства, сырья, персонала, просмотр любых данных в табличном виде. Различные операции с пользователями. Расчет итоговой стоимости, создание отчетов и вывод их в удобной форме. Пользователю доступно лишь создание отчета по готовым данным и просмотр их в удобной форме.

## 6.2 Описание алгоритма расчета итоговой стоимости

В приложении В представлена блок-схема алгоритма по расчету итоговой стоимости изделия на рисунке В.2.

Перед расчетом происходит подключение к базе данных и извлечение данных о производстве, персонале и сырье. После чего происходит вывод всех данных на экран. Далее необходимо выбрать данные для расчета. Если будет выбрано недостаточно данных, то расчет не будет произведен, а пользователь увидит сообщение об ошибке. При корректном выборе будет произведен расчет и итоговые данные будут внесены в базу данных.

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## 7.1 Руководство серверной части приложения

В первую очередь необходим запуск сервера проекта. Для этого необходимо установить порт и запустить сервер (рисунок 7.1). В окне сервера отображается статистика подключений и команды, приходящие от клиента (рисунок 7.2). После запуска выводится окно ожидания подключения первого клиента (рисунок 7.3).

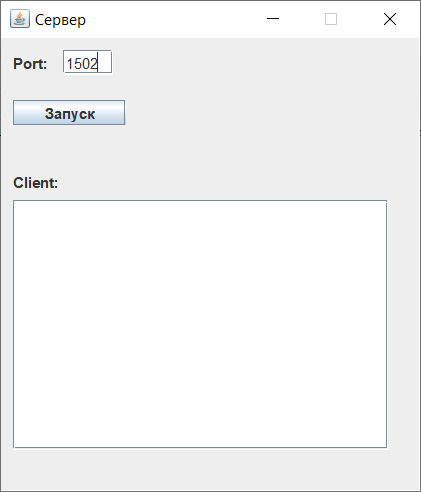


Рисунок 7.1 – Сервер до запуска

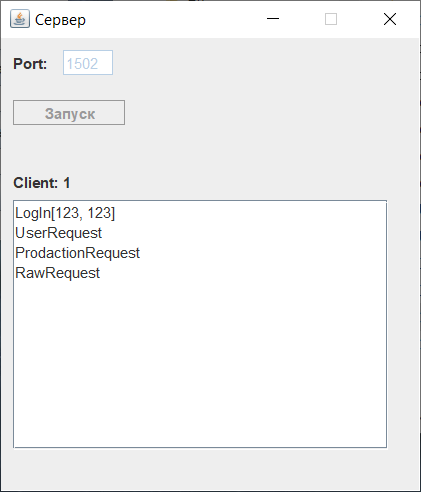


Рисунок 7.2 – Статистика подключений к серверу

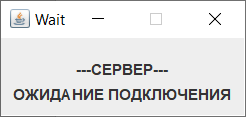


Рисунок 7.3 – Сервер в ожидании подключения

## 7.2 Руководство клиентской части приложения

Клиентская часть приложения при запуске в автоматическом режиме производит подключение к серверу. Если же сервер не доступен, то мы увидим соответствующее сообщение с возможностью переподключиться (рисунок 7.4).

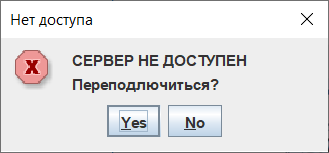


Рисунок 7.4 – Ошибка при подключении к серверу

Если подключение к серверу прошло удачно, то мы попадаем в окно входа в программу (рисунок 7.5). При корректном вводе данных будет отображено рабочее окно посетителя, в зависимости от роли пользователя или администратора. В противном случае, если введенные данные не корректны, то мы получим соответствующее предупреждение (рисунок 7.6).

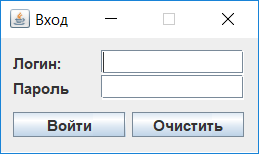


Рисунок 7.5 – Вход в программу

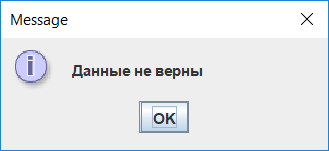


Рисунок 7.6 – Сообщение о не корректно введенных данных

После успешной авторизации, мы попадаем в главное меню (рисунок 7.7). Если вход произведен с учетной записи пользователя, то при попытке доступа к некоторым пунктам меню и частям программы будет выведено соответствующее сообщение (рисунок 7.8).

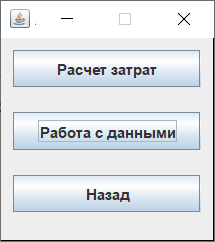


Рисунок 7.7 – Главное меню программы

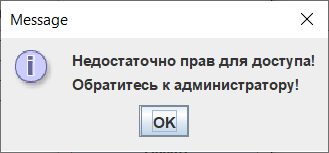


Рисунок 7.8 – Сообщение при попытке доступа к закрытым частям программы

В пункте меню работы с базой данных (рисунок 7.9), мы можем добавлять, изменять и удалять записи во всех имеющихся таблицах, а также проводить регистрацию пользователя. При удалении мы увидим сообщение о необратимости действий (рисунок 7.10). При какой-либо ошибке при обращении к базе данных мы увидим соответствующее сообщение (рисунок 7.11).

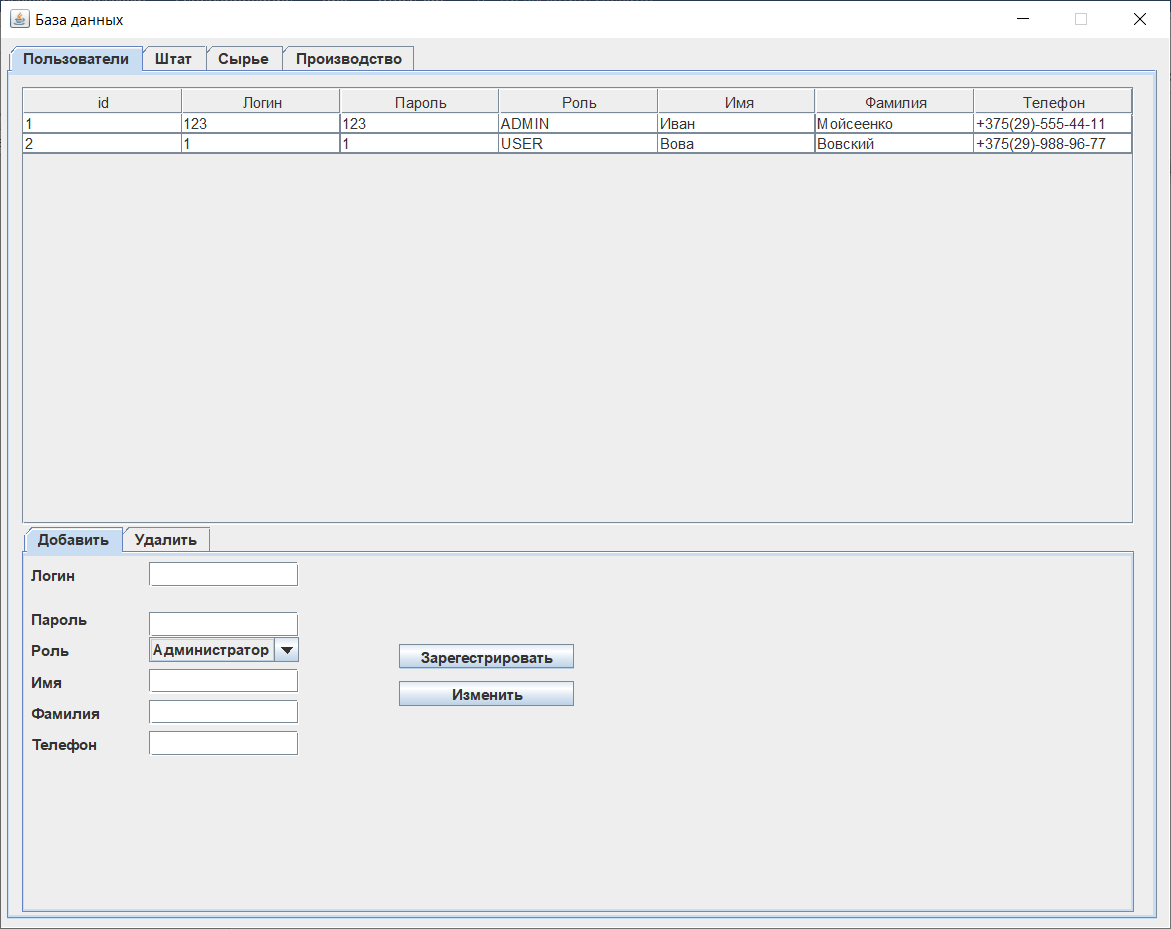


Рисунок 7.9 – Меню работы с базой данных, регистрация пользователя

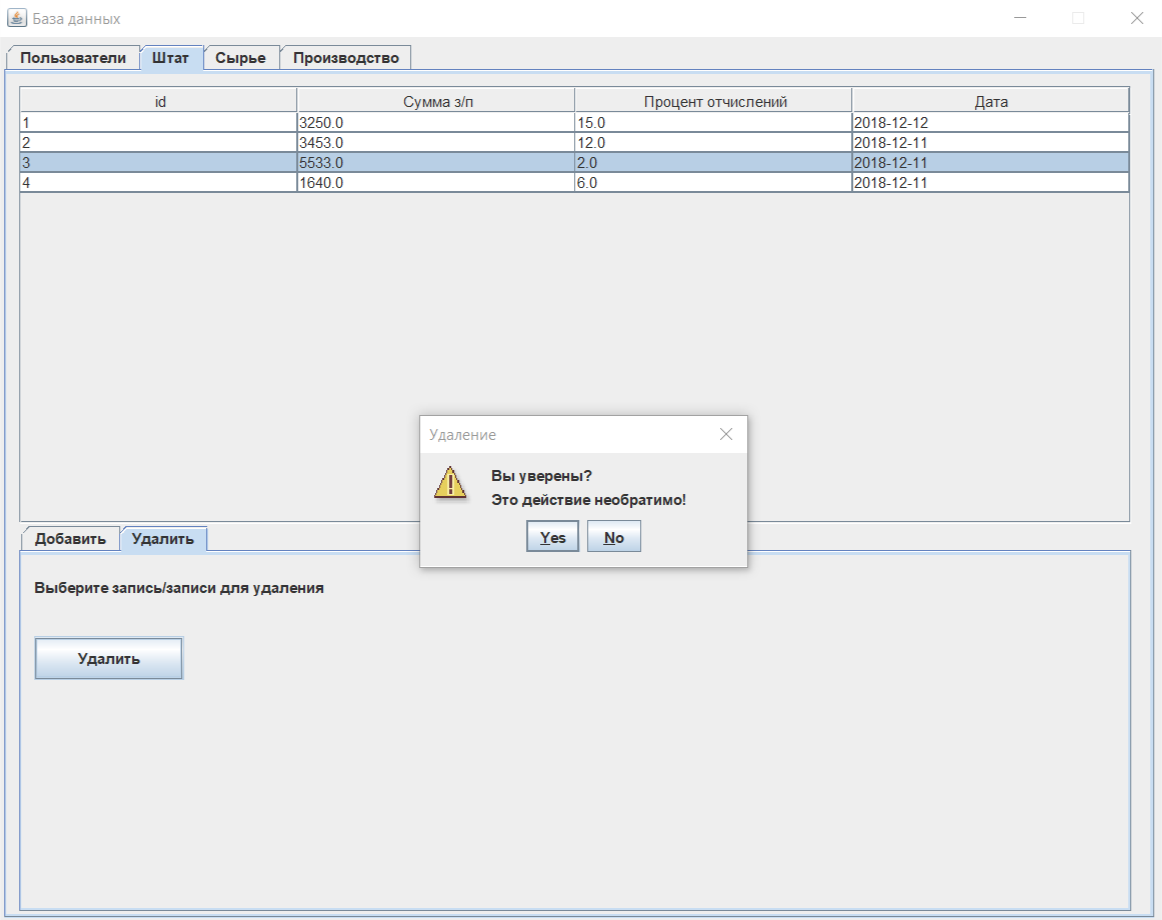


Рисунок 7.10 – Попытка удаления из таблицы штата

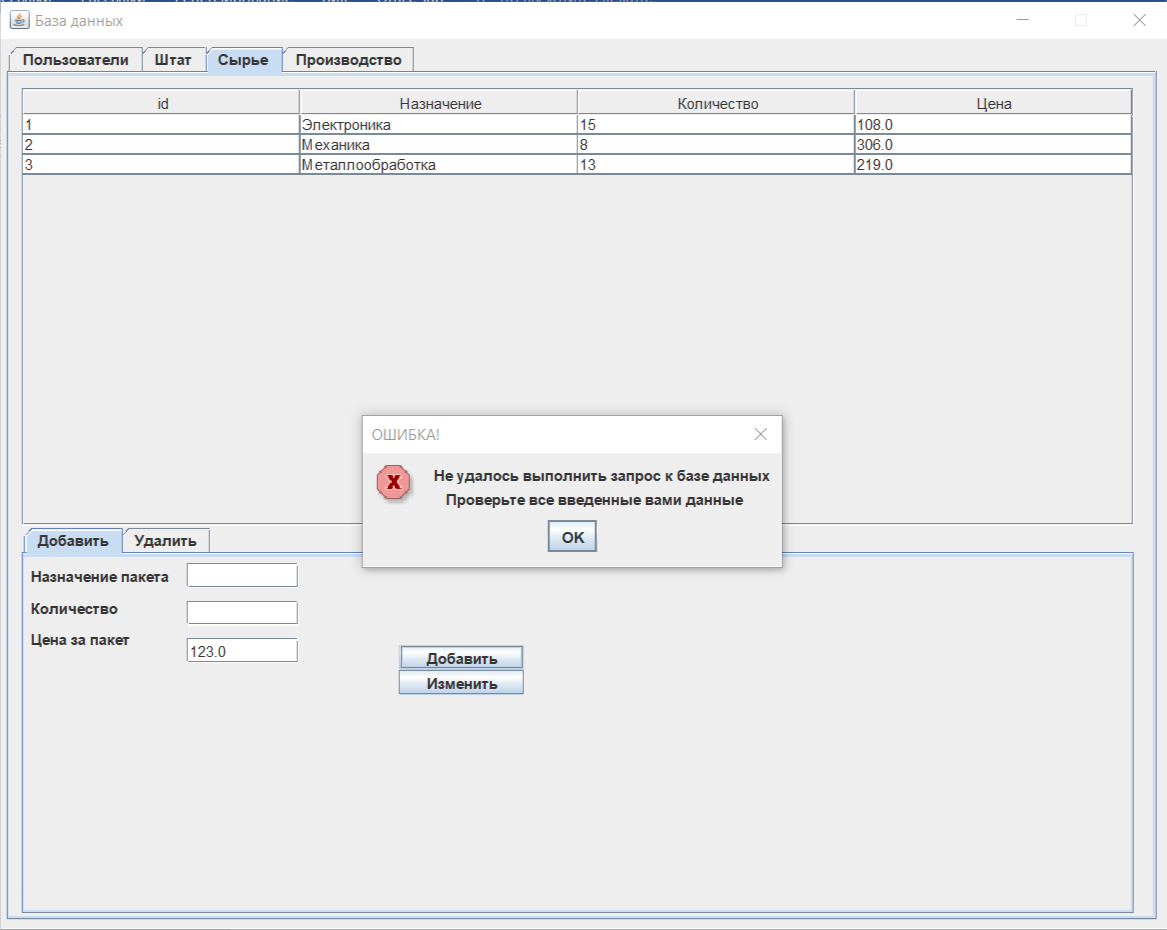


Рисунок 7.11 – Ошибка при добавлении записи в таблицу сырья

В пункте меню расчета затрат у нас отображаются все необходимые данные для расчетов. Для расчета нам нужно выбрать запись в каждой таблице и ввести необязательные дополнительные данные при необходимости. Если выбрать недостаточно данных, то при попытке расчета, мы увидим соответствующее сообщение (рисунок 7.12).

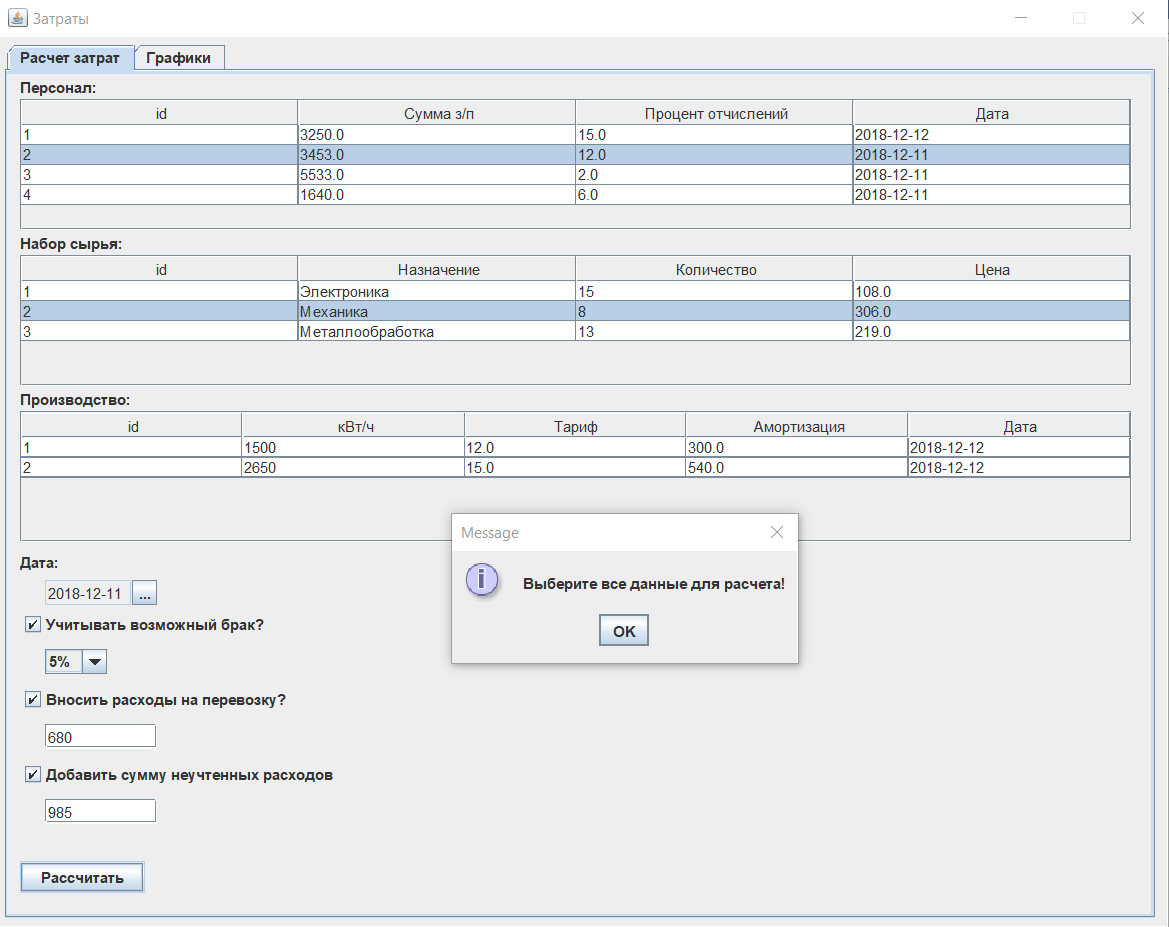


Рисунок 7.12 – Меню расчета затрат, ошибка при попытке выполнить расчет

В пункте графики (рисунок 7.14) мы видим все рассчитанные данные, можем вывести данные в графическом виде, распечатать график, сохранить его в виде картинки. Отфильтровать и найти необходимые данные в расчетах. Так же имеем возможность сохранить отчет в текстовом виде (рисунок 7.13). Пример отчета на рисунке 7.15.

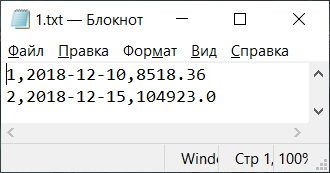


Рисунок 7.13 – Сохраненный отчет

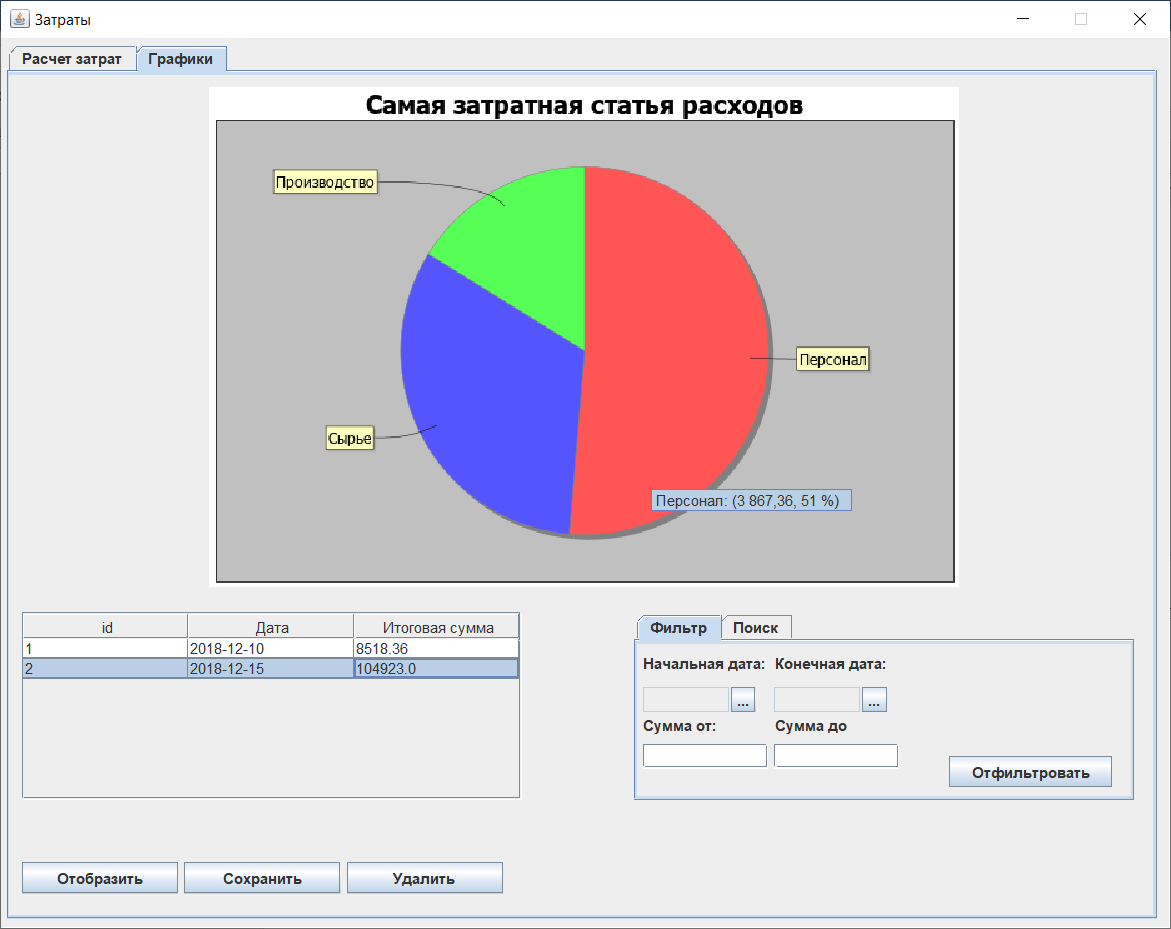


Рисунок 7.14 – Пункт меню с графиком.

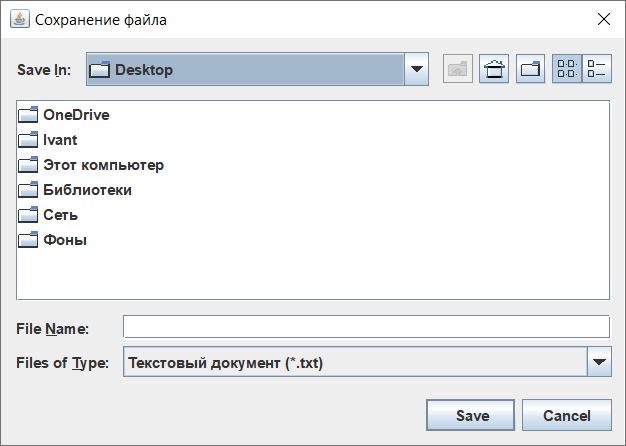


Рисунок 7.15 – Выбор места сохранения отчета

1. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СИСТЕМЫ РАСЧЕТА СТОИМОСТИ ИЗДЕЛИЯ

Для тестирования приложения была использована библиотека JUnit, позволяющая производить модульное тестирование отдельных компонентов системы.

Данный подход позволяет проверить на корректность выполнения отдельных модулей системы. Что позволяет проверять работоспособность системы после внесения каких-либо изменений.

Цель данного подхода – изолировать части (модули) системы и показать их работоспособность по отдельности.

Результаты тестирования показаны на рисунке 8.1.

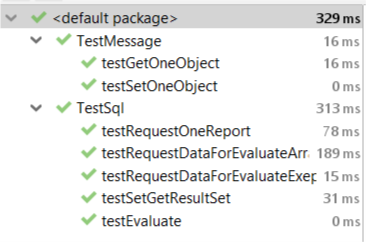


Рисунок 8.1 – Результаты тестирования частей программы

JUnit достаточно удобная и продуктивная система для модульного тестирования. Периодический запуск тестов в процессе разработки позволяет контролировать процесс и не допускать ошибок при внесении изменений в работающую часть программы.

ЗАКЛЮЧЕНИЕ

В ходе работы данного курсового проекта была разработана программа по оптимизации процесса расчета стоимости изделия, для облегчения и ускорения процесса расчета на предприятии. При выполнении курсового проекта, разработке программы была глубоко и детально исследована предметная область в сфере автоматизации расчетов. Результатом выполнения курсового проекта имеем конечное, законченное решение по автоматизации системы расчета стоимости изделия с удобным интерфейсом, имеющим все необходимые подсказки для пользователя при неправильном обращении. Как следствие – нет необходимости в дополнительном обучении персонала для работы с данным программным продуктом.

Все цели и задачи, поставленные для выполнения данного курсового проекта выполнены в полном объеме. Разработанное клиент-серверное приложение удовлетворяем всем современным требованиям, учтены все возможные нюансы, которые могут возникнуть при работе с программой. Обработаны все возможные варианты некорректного ввода, разработаны различные варианты проверок с сопутствующими сообщениями об ошибках, что в свою очередь дает возможность пользователю самостоятельно решать возникающие проблемы при работе с программой.

Архитектура клиент-сервер позволяет многим пользователем общаться с одним сервером и общей базой данных, что позволяет в разы ускорить процесс ручных расчетов и сократить общую стоимость вычислительной системы.

Система, разработанная в данном курсовом проекте, имеет достаточно четкое практическое применение и может внедряться на предприятиях для своей прямой задачи: автоматизация системы расчета стоимости предприятия. Учитывая ориентированность приложения на неподготовленного пользователя, внедрение данного приложения не повлечет за собой дополнительных затрат на обучение или переобучение персонала, которому предстоит использовать данный продукт для более удобного решения своих повседневных задач.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Дейт, К. Введение в системы баз данных/ К. Дейт: пер. с англ. – 6–е изд. – СПб.: Издательский дом «Вильямс», 2000. – 848 с.

[2] Кати Сьерра, Элизабет Фримен. Паттерны проектирования. СПб: «Питер», 2004. – 421 с.

[3] Э. Гамма, Р. Хелм, Р. Джонсон, [Дж. Влиссидес](https://ru.wikipedia.org/wiki/%D0%92%D0%BB%D0%B8%D1%81%D0%B8%D0%B4%D0%B8%D1%81,_%D0%94%D0%B6%D0%BE%D0%BD). Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб: «Питер», 2007. – 366 с.

[4] Алан Шаллоуей, Джеймс Р. Тротт. Шаблоны проектирования. Новый подход к объектно-ориентированному анализу и проектированию. – СПб.: Издательский дом «Вильямс», 2002. – 288 c.

[5] Методология функционального моделирования IDEF0, Госстандарт России, Москва. – ИПК Издательство стандартов, 2000. – 75 с.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Диаграмма информационной модели системы**



Рисунок А.1 – Информационная модель системы расчета стоимости изделия

ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Диаграммы представления**

****

Рисунок Б.1 – Диаграмма вариантов использования системы расчета стоимости изделия

**Продолжение приложения Б**

****

Рисунок Б.2 – Диаграмма последовательности добавления нового пользователя

**Продолжение приложения Б**

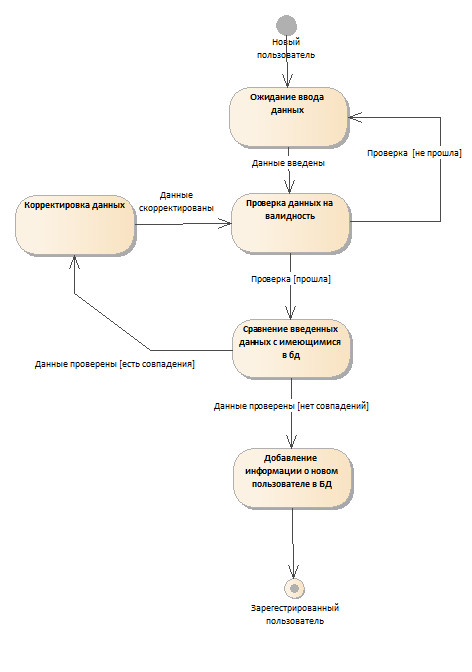


Рисунок Б.3 – Диаграмма состояний процесса регистрации нового пользователя

**Продолжение приложения Б**

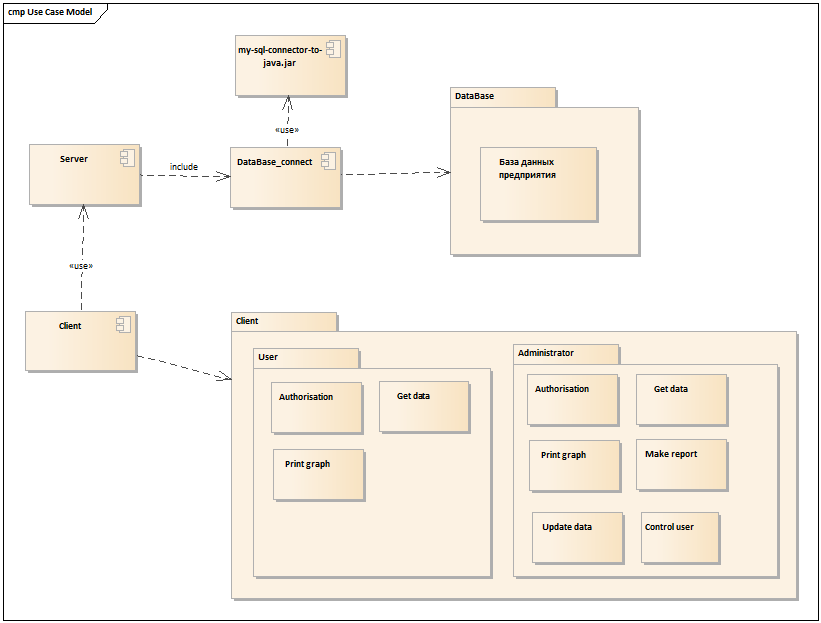


Рисунок Б.4 – Диаграмма компонентов системы расчета стоимости изделия

**Продолжение приложения Б**



Рисунок Б.5 – Диаграмма развертывания системы расчета стоимости изделия

**Продолжение приложения Б**

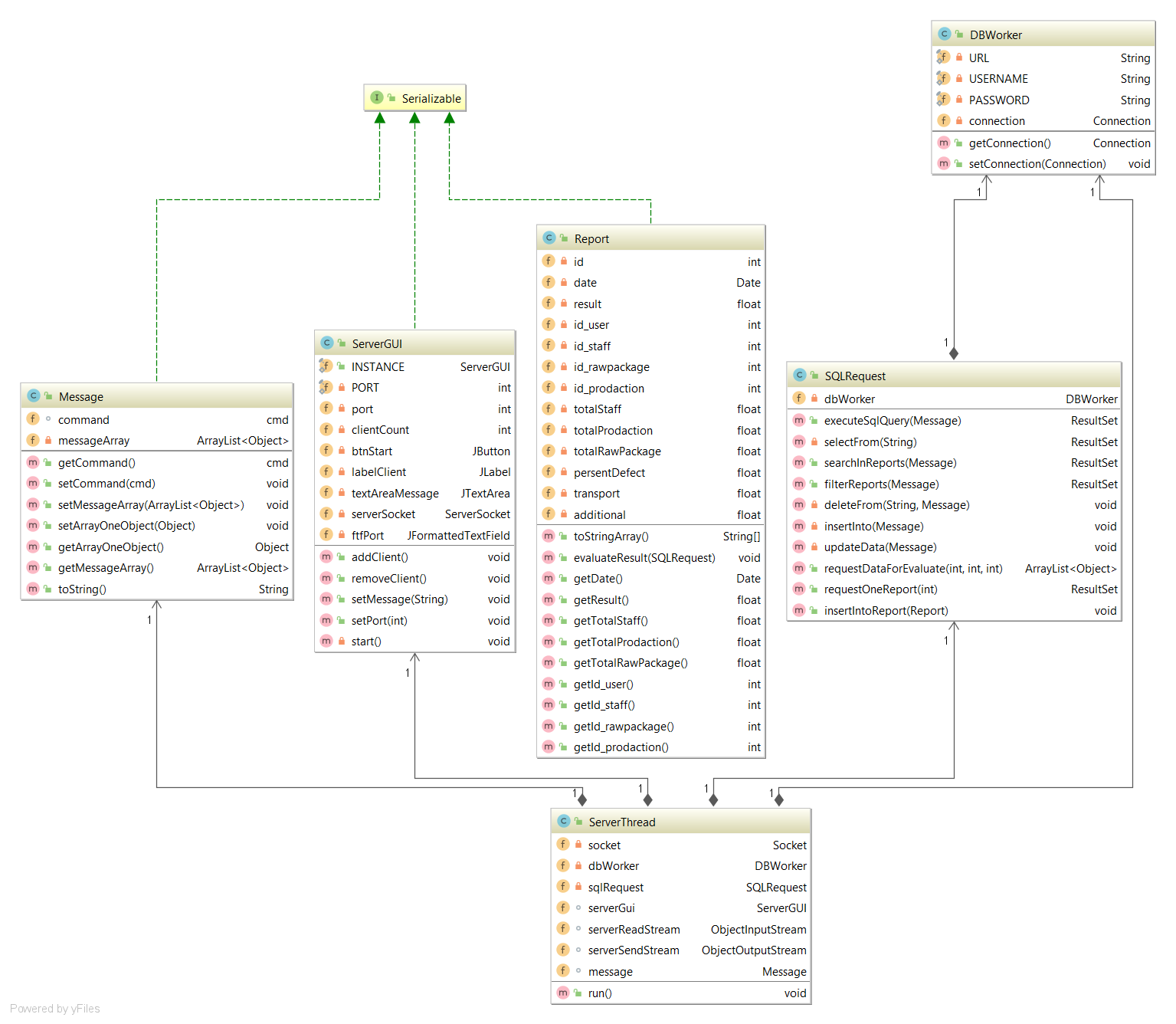


Рисунок Б.6 – Диаграмма классов системы расчета стоимости изделия

**Продолжение приложения Б**

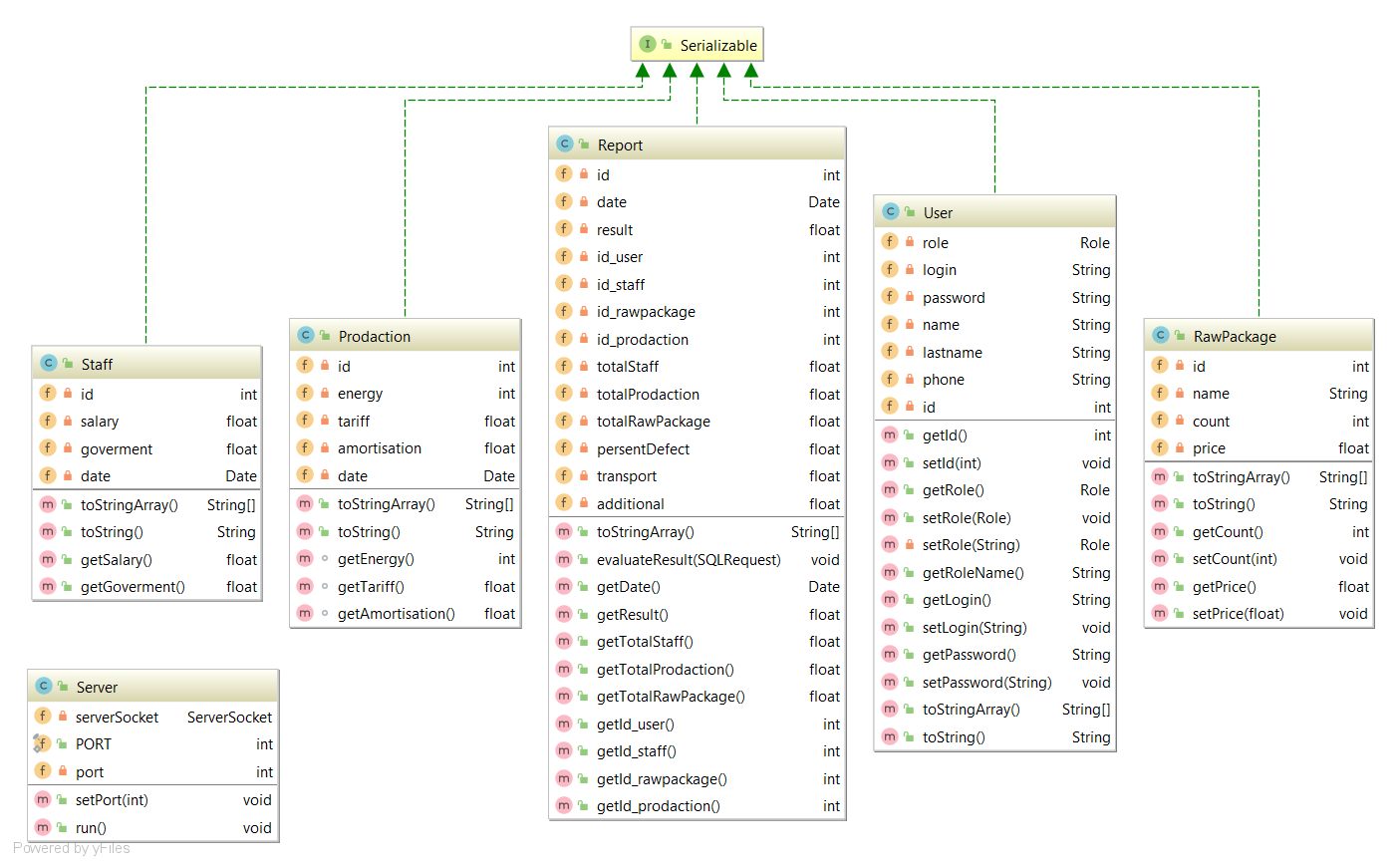


Рисунок Б.7 – Диаграмма классов системы расчета стоимости изделия

ПРИЛОЖЕНИЕ В

**(обязательное)**

**Алгоритмы работы программы**

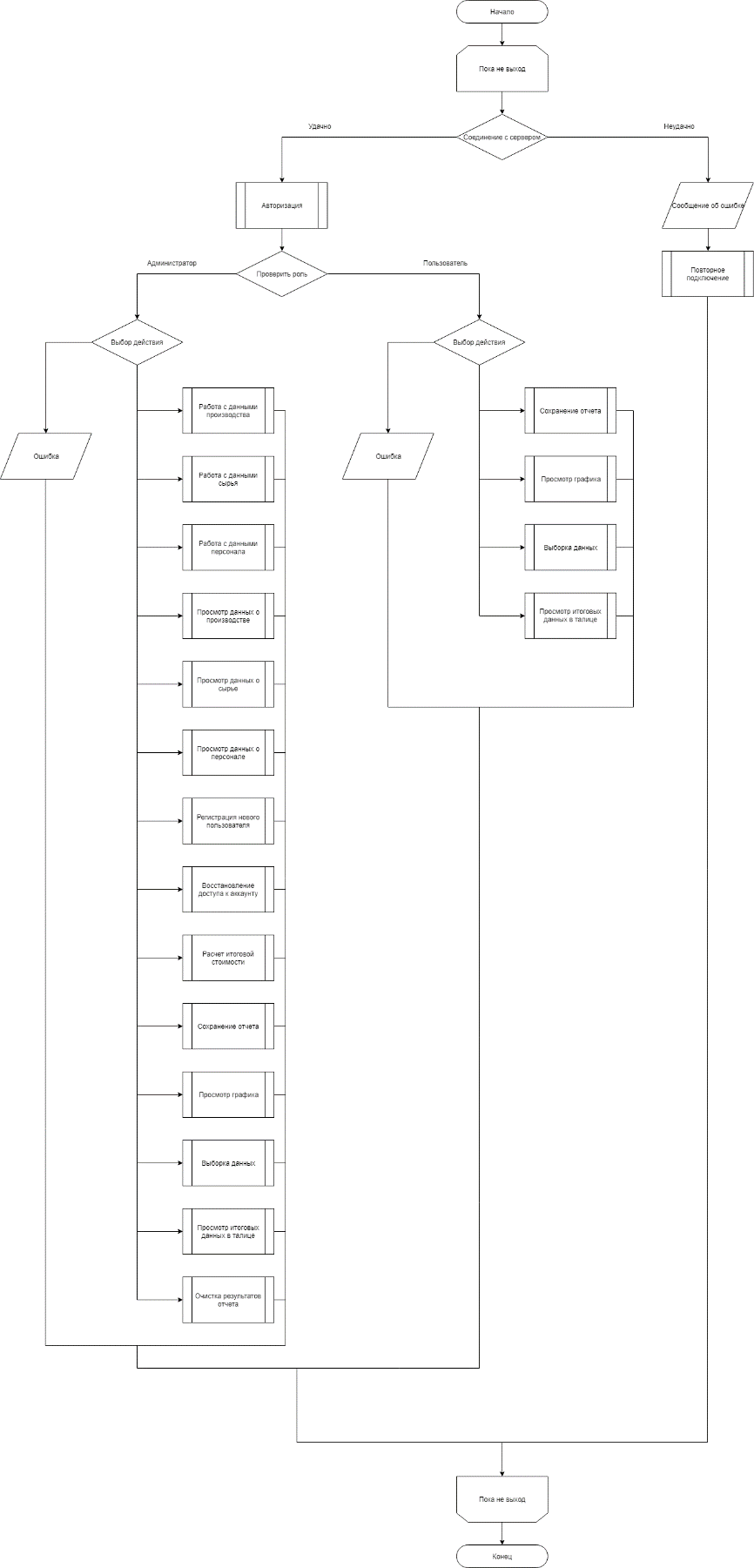


Рисунок В.1 – Блок-схема работы всей программы

**Продолжение приложения В**

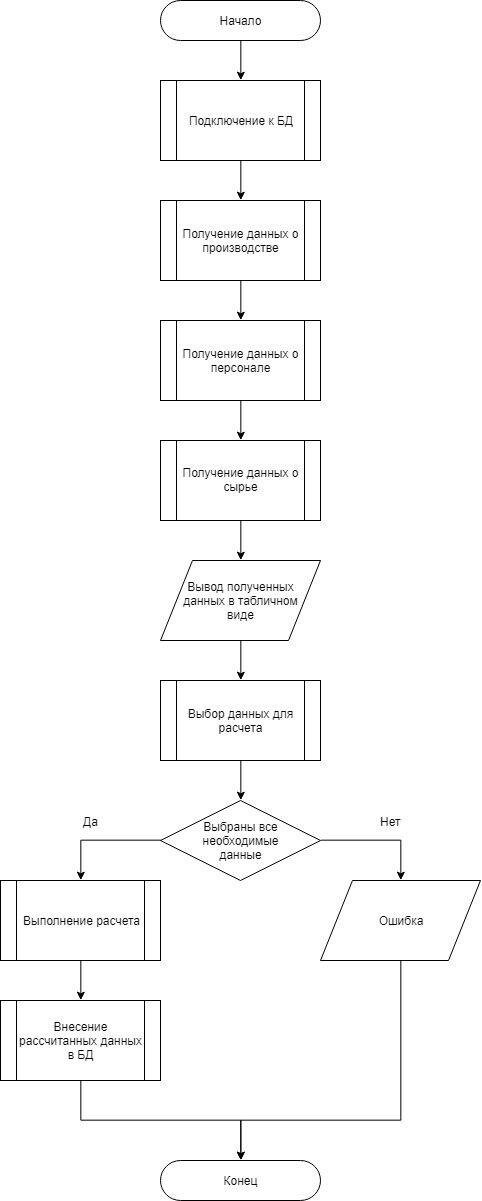


Рисунок В.2 – Блок-схема процесса расчета итоговой стоимости изделия

ПРИЛОЖЕНИЕ Г

**(Обязательное)**

**Листинг скрипта генерации базы данных**

CREATE SCHEMA IF NOT EXISTS `server` DEFAULT CHARACTER SET utf8 COLLATE utf8\_unicode\_ci ;

USE `server` ;

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

-- Table `server`.`prodaction`

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

CREATE TABLE IF NOT EXISTS `server`.`prodaction` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`energy` INT(11) NOT NULL,

`tariff` FLOAT NULL DEFAULT '0.12',

`amortisation` FLOAT NOT NULL,

`date` DATE NOT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

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

-- Table `server`.`rawpackage`

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

CREATE TABLE IF NOT EXISTS `server`.`rawpackage` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) NOT NULL,

`count` INT(11) NOT NULL,

`price` FLOAT NOT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 5

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

**Продолжение приложения Г**

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

-- Table `server`.`staff`

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

CREATE TABLE IF NOT EXISTS `server`.`staff` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`salary` FLOAT NOT NULL,

`goverment` FLOAT NOT NULL,

`date` DATE NOT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

AUTO\_INCREMENT = 5

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

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

-- Table `server`.`users`

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

CREATE TABLE IF NOT EXISTS `server`.`users` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`login` VARCHAR(45) NOT NULL,

`password` VARCHAR(45) NOT NULL,

`role` VARCHAR(45) NOT NULL,

`name` VARCHAR(45) NULL DEFAULT NULL,

`lastname` VARCHAR(45) NULL DEFAULT NULL,

`phone` VARCHAR(45) NULL DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `id\_UNIQUE` (`id` ASC) VISIBLE,

UNIQUE INDEX `login\_UNIQUE` (`login` ASC) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

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

**Продолжение приложения Г**

-- Table `server`.`report`

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

CREATE TABLE IF NOT EXISTS `server`.`report` (

`id` INT(11) NOT NULL AUTO\_INCREMENT,

`date` DATE NOT NULL,

`result` FLOAT NOT NULL,

`id\_user` INT(11) NOT NULL,

`id\_staff` INT(11) NOT NULL,

`id\_rawpackage` INT(11) NOT NULL,

`id\_prodaction` INT(11) NOT NULL,

`total\_staff` FLOAT NOT NULL,

`total\_rawpackage` FLOAT NOT NULL,

`total\_prodaction` FLOAT NOT NULL,

PRIMARY KEY (`id`),

INDEX `accessories\_id\_idx` (`id\_staff` ASC) VISIBLE,

INDEX `accessories\_id\_idx1` (`id\_rawpackage` ASC) VISIBLE,

INDEX `prodaction\_id\_idx` (`id\_prodaction` ASC) VISIBLE,

INDEX `user\_id\_idx` (`id\_user` ASC) VISIBLE,

CONSTRAINT `prodaction\_id`

FOREIGN KEY (`id\_prodaction`)

REFERENCES `server`.`prodaction` (`id`)

ON UPDATE CASCADE,

CONSTRAINT `rawpackage\_id`

FOREIGN KEY (`id\_rawpackage`)

REFERENCES `server`.`rawpackage` (`id`),

CONSTRAINT `staff\_id`

FOREIGN KEY (`id\_staff`)

REFERENCES `server`.`staff` (`id`)

ON UPDATE CASCADE,

CONSTRAINT `user\_id`

FOREIGN KEY (`id\_user`)

REFERENCES `server`.`users` (`id`)

ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8

ПРИЛОЖЕНИЕ Д

**(обязательное)**

**Листинг кода**

**package** Instruments;  
  
**import** ui.ServerGUI;  
  
**public class** Main {  
 **public static void** main(String[] args) {  
 ServerGUI serverGui = ServerGUI.***INSTANCE***;  
 serverGui.setSize(350,400);  
 serverGui.setVisible(**true**);  
 serverGui.setResizable(**false**);  
 serverGui.setLocationRelativeTo(**null**);  
 }  
  
}

**package** Instruments;  
  
**public class** Server **extends** Thread {  
 **private** ServerSocket **serverSocket**;  
  
 **public static final int *PORT*** = 1502;  
  
 **public void** setPort(**int** port) {  
 **this**.**port** = port;  
 }  
  
 **private int port** = ***PORT***;  
  
 **public** Server(ServerSocket socket) {  
 **this**.**serverSocket** = socket;  
 }  
 @Override  
 **public void** run() {  
 WaitFrame waitFrame = **new** WaitFrame(**"Wait"**);  
 waitFrame.setVisible(**true**);  
 waitFrame.setResizable(**false**);  
 waitFrame.setLocationRelativeTo(**null**);  
 **try** {  
 **while** (**true**) {  
 **new** ServerThread(**serverSocket**.accept()).start();  
 waitFrame.setVisible(**false**);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }

**Продолжение приложения Д**

}

**package** Instruments;  
  
**public class** ServerThread **extends** Thread {  
  
 **private** Socket **socket**;  
 **private** DBWorker **dbWorker**;

**private** SQLRequest **sqlRequest**;  
  
 ServerGUI **serverGui**;  
  
 ObjectInputStream **serverReadStream**;  
 ObjectOutputStream **serverSendStream**;  
  
 Message **message** = **new** Message();  
  
 **public** ServerThread(Socket socket) {  
 **this**.**socket** = socket;  
 }  
  
 @Override  
 **public void** run() {  
  
 **serverGui** = ServerGUI.***INSTANCE***;  
 **serverGui**.setVisible(**true**);  
 **serverGui**.setResizable(**false**);  
 **serverGui**.setLocationRelativeTo(**null**);  
 **serverGui**.addClient();  
  
 System.***out***.println(**socket**.getInetAddress().getHostName() +  
 **" "** + **socket**.getInetAddress() + **"connected"**);  
  
 **dbWorker** = **new** DBWorker();  
 **sqlRequest** = **new** SQLRequest();  
  
 **try** {  
 **serverReadStream** = **new** ObjectInputStream(**socket**.getInputStream());  
 **serverSendStream** = **new** ObjectOutputStream(**socket**.getOutputStream());  
  
 **while** (!**message**.getCommand().equals(Message.cmd.***Stop***)) {  
 **message** = (Message) **serverReadStream**.readObject();  
 System.***out***.println(**"incoming: "** + **message**);  
 **serverGui**.setMessage(**message**.toString());  
  
 **try** {  
 **switch** (**message**.getCommand()) {  
 **case *LogIn***: {

**Продолжение приложения Д**

String login = (String) **message**.getMessageArray().get(0);  
 String password = (String) **message**.getMessageArray().get(1);  
 User user = **new** User();  
 **message**.setCommand(Message.cmd.***LogIn***);  
 **if** (!(login.isEmpty() || password.isEmpty())) {  
  
 String query = **"select** *\** **from users where "** +  
 **"login = "** + login + **" and"** + **" password = "** + password;  
 Statement statement = **dbWorker**.getConnection().createStatement();  
 ResultSet resultSet = statement.executeQuery(query);  
  
 **if** (resultSet.next()) {  
 user = **new** User(resultSet);  
 }  
 }  
 **message**.setArrayOneObject(user);  
 **serverSendStream**.writeObject(**message**);  
 }  
 **break**;  
 **case *UserRequest***: {  
 ArrayList<Object> userArrayList = **new** ArrayList<>();  
 ResultSet resultSet = **sqlRequest**.executeSqlQuery(**message**);  
 **while** (resultSet.next()) {  
 userArrayList.add(**new** User(resultSet));  
 }  
 **message**.setMessageArray(userArrayList);  
 **serverSendStream**.writeObject(**message**);  
 }  
 **break**;  
...  
 }  
 } **catch** (SQLException | NumberFormatException e) {  
 e.printStackTrace();  
  
 **message** = **new** Message();  
 **message**.setCommand(Message.cmd.***FailSQL***);  
 **try** {  
 **serverSendStream**.writeObject(**message**);  
 } **catch** (IOException e1) {  
 e1.printStackTrace();  
 }  
 }  
 }  
  
 } **catch** (IOException | ClassNotFoundException e) {

**Продолжение приложения Д**

**serverGui**.removeClient();  
 }  
  
 }  
}

**package** DataBase;  
  
**public class** DBWorker {  
 **private static final** String ***URL*** = **"jdbc:mysql://localhost:3306/server?autoReconnect=true&useSSL=false&useLegacyDatetimeCode=false&serverTimezone=UTC"**;  
 **private static final** String ***USERNAME*** = **"root"**;  
 **private static final** String ***PASSWORD*** = **"root"**;  
  
 **private** Connection **connection**;  
  
 **public** DBWorker() {  
 **try** {  
 **connection** = DriverManager.*getConnection*(***URL***,***USERNAME***,***PASSWORD***);  
 } **catch** (SQLException e){  
 e.printStackTrace();  
 }  
 }  
  
 **public** Connection getConnection() {  
 **return connection**;  
 }  
  
 **public void** setConnection(Connection connection) {  
 **this**.**connection** = connection;  
 }  
  
  
}

**package** DataBase;  
  
**public class** SQLRequest {  
  
 **private** DBWorker **dbWorker** = **new** DBWorker();  
  
 **public** ResultSet executeSqlQuery(Message message) **throws** SQLException {  
 ResultSet resultSet = **null**;  
 **switch** (message.getCommand()){  
 **case *UserRequest***:  
 resultSet = selectFrom(**"users"**);  
 **break**;  
 **case *UserAdd***:

insertInto(message);

**Продолжение приложения Д**

**break**;  
 **case *UserRedact***:  
 updateData(message);  
 **break**;  
*...*

}  
 **return** resultSet;  
 }  
  
 **private** ResultSet selectFrom(String table) **throws** SQLException {  
 String query;  
 query = **"select** *\** **from %s"**;  
 query = String.*format*(query, table);  
 PreparedStatement preparedStatement;  
 preparedStatement = **dbWorker**.getConnection().prepareStatement(query);  
  
 **return** preparedStatement.executeQuery();  
 }  
  
 **public** ResultSet searchInReports(Message message) **throws** SQLException {  
 String query;  
 query = **"select** *\** **from report WHERE id LIKE ? and date LIKE ? and result LIKE ?"**;PreparedStatement preparedStatement;  
 preparedStatement = **dbWorker**.getConnection().prepareStatement(query);  
 String id = (String) message.getMessageArray().get(0);  
 **if**(id.equals(**""**)){  
 preparedStatement.setString(1,**"%"**);  
 } **else** {  
 preparedStatement.setString(1,id);  
 }  
 String data = (String) message.getMessageArray().get(1);  
 **if**(data.equals(**""**)){  
 preparedStatement.setString(2,**"%"**);  
 } **else** {  
 preparedStatement.setString(2,data);  
 }  
 String result = (String) message.getMessageArray().get(2);  
 **if**(result.equals(**""**)){  
 preparedStatement.setString(3,**"%"**);  
 } **else** {  
 preparedStatement.setString(3,result);  
 }  
 **return** preparedStatement.executeQuery();  
 }  
...

**public** ArrayList<Object> requestDataForEvaluate(**int** idStaff, **int** idProdaction, **int** idRawPackage) **throws** SQLException {  
 String query = **"select \* from %s where id=?"**;

**Продолжение приложения Д**

PreparedStatement preparedStatement;  
  
 String queryTable;  
  
 queryTable = String.*format*(query, **"staff"**);  
 preparedStatement = **dbWorker**.getConnection().prepareStatement(queryTable);  
 preparedStatement.setInt(1,idStaff);  
 ResultSet resultSet = preparedStatement.executeQuery();  
 Staff staff = **null**;  
 **if**(resultSet.next()) {  
 staff = **new** Staff(resultSet);  
 }  
  
 queryTable = String.*format*(query, **"prodaction"**);  
 preparedStatement = **dbWorker**.getConnection().prepareStatement(queryTable);  
 preparedStatement.setInt(1,idProdaction);  
 resultSet = preparedStatement.executeQuery();  
 Prodaction prodaction = **null**;  
 **if**(resultSet.next()) {  
 prodaction = **new** Prodaction(resultSet);  
 }  
  
 queryTable = String.*format*(query, **"rawPackage"**);  
 preparedStatement = **dbWorker**.getConnection().prepareStatement(queryTable);  
 preparedStatement.setInt(1,idRawPackage);  
 resultSet = preparedStatement.executeQuery();  
 RawPackage rawPackage = **null**;  
 **if**(resultSet.next()) {  
 rawPackage = **new** RawPackage(resultSet);  
 }  
  
 ArrayList<Object> arrayList = **new** ArrayList<>();  
  
 arrayList.add(staff);  
 arrayList.add(prodaction);  
 arrayList.add(rawPackage);  
  
 **return** arrayList;  
 }  
  
 **public** ResultSet requestOneReport(**int** id) **throws** SQLException {  
 String query;  
 query = **"select** *\** **from report WHERE id = ?"**;  
 PreparedStatement preparedStatement;  
 preparedStatement = **dbWorker**.getConnection().prepareStatement(query);  
 preparedStatement.setInt(1, id);  
  
 **return** preparedStatement.executeQuery();  
 }

**Продолжение приложения Д**

...  
}

**package** Message;  
  
**public class** Message **implements** Serializable {  
 **public enum** cmd{  
 ***Start***,  
 ***LogIn***,  
 ***UserRequest***,  
 ***Stop***,  
 ***UserDelete***,  
 ***UserAdd***,  
 ***UserRedact***,  
...  
 ***FailSQL*** }  
 cmd **command** = cmd.***Start***;  
  
 **private** ArrayList<Object> **messageArray**;  
  
 **public** cmd getCommand() {  
 **return command**;  
 }  
  
 **public void** setCommand(cmd command) {  
 **this**.**command** = command;  
 }  
  
  
 **public void** setMessageArray(ArrayList<Object> messageArray) {  
 **this**.**messageArray** = **new** ArrayList<>(messageArray);  
 }  
  
 **public void** setArrayOneObject(Object messageObject){  
 **this**.**messageArray** = **new** ArrayList<>();  
 **this**.**messageArray**.add(messageObject);  
 }  
  
 **public** Object getArrayOneObject(){  
 **return messageArray**.toArray()[0];  
 }  
  
 **public** ArrayList<Object> getMessageArray() {  
 **return messageArray**;  
 }  
  
 @Override  
 **public** String toString() {

**Продолжение приложения Д**

String str = **new** String();  
 str += String.*valueOf*(**command**);  
 **if**(**messageArray** != **null**)  
 str += **messageArray**.toString();  
 **return** str;  
 }  
}

**package** BDTable;  
  
**public class** Report **implements** Serializable {  
 **...**  
  
 **public void** evaluateResult(SQLRequest sql){  
 **try** {  
 ArrayList<Object> arrayList = sql.requestDataForEvaluate(**id\_staff**, **id\_prodaction**, **id\_rawpackage**);  
 Staff staff = (Staff)arrayList.get(0);  
 Prodaction prodaction = (Prodaction)arrayList.get(1);  
 RawPackage rawPackage = (RawPackage)arrayList.get(2);  
  
 **totalStaff** += staff.getSalary()+staff.getSalary()\*(staff.getGoverment()/100);  
  
 **totalProdaction** += prodaction.getAmortisation() + prodaction.getEnergy()\*prodaction.getTariff()/100 + **transport**;  
 **totalProdaction** \*= **persentDefect**;  
  
 **totalRawPackage** += rawPackage.getCount()\* rawPackage.getPrice();  
  
 **this**.**result** += **totalStaff** + **totalProdaction** + **totalRawPackage** + **additional**;  
  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
**...**  
}

**package** ui;  
  
**public class** ServerGUI **extends** JFrame {  
  
 **public static final** ServerGUI ***INSTANCE*** = **new** ServerGUI(**"Сервер"**);  
  
 **private static final int *PORT*** = 1502;  
 **private int port** = ***PORT***;  
  
 **private int clientCount** = 0;

**Продолжение приложения Д**

**private** JButton **btnStart**;  
  
 **private** JLabel **labelClient**;  
 **private** JTextArea **textAreaMessage**;  
 **private** ServerSocket **serverSocket**;  
 **private** JFormattedTextField **ftfPort**;  
  
 **private** ServerGUI(String str) {  
 **super**(str);  
 setSize(500, 500);  
 setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
  
 **btnStart** = **new** JButton(**"Запуск"**);**labelClient** = **new** JLabel(**"Client: "**);  
 **textAreaMessage** = **new** JTextArea(9, 9);  
 **textAreaMessage**.setEditable(**false**);  
 JScrollPane sp = **new** JScrollPane(**textAreaMessage**);  
  
 setLayout(**null**);  
  
 **btnStart**.setBounds(10, 50, 90, 20);  
 **labelClient**.setBounds(10, 105, 90, 20);  
 sp.setBounds(10, 130, 300, 200);  
  
 JLabel lbPort = **new** JLabel(**"Port:"**);  
 lbPort.setBounds(10, 10, 40, 20);  
  
 MaskFormatter mf = **null**;  
 **try** {  
 mf = **new** MaskFormatter(**"####"**);  
 mf.setPlaceholderCharacter(**'\*'**);  
 } **catch** (ParseException e) {  
 e.printStackTrace();  
 }  
 **ftfPort** = **new** JFormattedTextField(mf);  
 **ftfPort**.addKeyListener(**new** TftCaractersListener());  
 **ftfPort**.setBounds(50,10,40,20);  
 **ftfPort**.setText(String.*valueOf*(***PORT***));  
  
 add(**btnStart**);  
 add(lbPort);  
  
 add(**labelClient**);  
 add(sp);  
 add(**ftfPort**);  
  
 **btnStart**.addActionListener(**new** ServerGUI.ButtonActionListener());  
  
 }

**Продолжение приложения Д**

**public void** setMessage(String message) {  
 **textAreaMessage**.setText(**textAreaMessage**.getText() + message + **"\r\n"**);  
 }  
  
 **public void** setPort(**int** port) {  
 **this**.**port** = port;  
 }  
  
 **public class** ButtonActionListener **implements** ActionListener {  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (e.getSource() == **btnStart**) {  
 System.***out***.println(**"Start server"**);  
 setPort(**port**);  
 start();  
 **btnStart**.setEnabled(**false**);  
 **ftfPort**.setEnabled(**false**);  
 }  
 }  
 }  
  
 **private void** start() {  
 **try** {  
 **serverSocket** = **new** ServerSocket(**port**);  
 **new** Server(**serverSocket**).start();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
  
 **public class** TftCaractersListener **extends** KeyAdapter {  
  
 **public void** keyTyped(KeyEvent e) {  
 **char** c = e.getKeyChar();  
 **if** (!((c >= **'0'**) && (c <= **'9'**) ||  
 (c == KeyEvent.***VK\_BACK\_SPACE***) ||  
 (c == KeyEvent.***VK\_DELETE***))) {  
 JOptionPane.*showMessageDialog*(**null**, **"Вводите олько цифры!"**, **"ОШИБКА!"**, JOptionPane.***ERROR\_MESSAGE***);  
 e.consume();  
 }  
 }  
  
 }  
}

**package** ui;

**Продолжение приложения Д**

**public interface** SocketGuiInterface {  
  
 **void** setClientSendStream(ObjectOutputStream clientSendStream);  
 **void** setMessage(Message message);  
}

**package** Instruments;  
  
**public class** Client {  
 **private static final int *PORT*** = 1502;  
  
 **private** MainMenu **uiAdminMain** = **null**;  
 **private** DB **db**;  
 **private** EvaluateUI **mrpUI**;  
 **private** LogIN **uiLogIN**;  
  
  
 **private** User **user**;  
 **private** JPanel **panel**;  
  
 **public static void** main(String[] args) {  
 Client client = **new** Client();  
 client.run();  
 }  
  
 **private** Client() {  
 **panel** = **new** JPanel();  
 }  
  
 **private void** run() {  
  
  
 Message message = **new** Message();  
  
 Socket clientSocket;  
 **try** {  
 **uiLogIN** = **new** LogIN(**"Вход"**);  
  
 clientSocket = **new** Socket(**"127.0.0.1"**, ***PORT***);  
 ObjectOutputStream clientSendStream = **new** ObjectOutputStream(clientSocket.getOutputStream());  
 ObjectInputStream clientReadStream = **new** ObjectInputStream(clientSocket.getInputStream());  
  
  
 **uiLogIN**.setVisible(**true**);  
 **uiLogIN**.setResizable(**false**);  
 **uiLogIN**.setLocationRelativeTo(**null**);  
  
 **uiLogIN**.setClientSendStream(clientSendStream);

**Продолжение приложения Д**

**uiLogIN**.setMessage(message);  
  
 **uiAdminMain** = **new** MainMenu(clientSendStream, message);  
 **uiAdminMain**.setVisible(**false**);  
 **uiAdminMain**.setResizable(**false**);  
 **uiAdminMain**.setLocationRelativeTo(**null**);  
  
 **db** = **uiAdminMain**.getDbUI();  
 **mrpUI** = **uiAdminMain**.getEvaluateUI();  
  
 **while** (!message.getCommand().equals(Message.cmd.***Stop***)) {  
  
 **uiLogIN**.setClientSendStream(clientSendStream);  
 **uiLogIN**.setMessage(message);  
  
 message = (Message) clientReadStream.readObject();  
 **switch** (message.getCommand()) {  
 **case *LogIn***:  
 **user** = (User) message.getMessageArray().get(0);  
 **switch** (**user**.getRole()) {  
 **case *ADMIN***:  
 **case *USER***:  
 **uiAdminMain**.setVisible(**true**);  
 **uiAdminMain**.setUser(**user**);  
 **break**;  
 **case *FAIL***:  
 JOptionPane.*showMessageDialog*(**null**, **"Данные не верны"**);  
 **break**;  
 }  
 **break**;  
 **case *UserRequest***: {  
 System.***out***.println(message.getMessageArray());  
  
 ArrayList<User> userArrayList = **new** ArrayList<>();  
 **for** (Object object : message.getMessageArray()) {  
 userArrayList.add((User) object);  
 }  
  
 String[][] stringDa = **new** String[userArrayList.size()][];  
 **int** i = 0;  
 **for** (User user1 : userArrayList) {  
 stringDa[i] = user1.toStringArray();  
 i++;  
 }  
 **db**.setUserData(stringDa);  
 }  
 **break**;  
**...**

**Продолжение приложения Д**

**case *FailSQL***:  
 JOptionPane.*showMessageDialog*(**null**, **"Не удалось выполнить запрос к базе данных\r\n"** +  
 **" Проверьте все введенные вами данные"**, **"ОШИБКА!"**, JOptionPane.***ERROR\_MESSAGE***);  
 **break**;  
 **default**:  
 **break**;  
 }  
  
 System.***out***.println(message);  
 }  
 } **catch** (IOException | ClassNotFoundException e) {  
 e.printStackTrace();  
  
 **if** (**uiAdminMain** != **null**)  
 **uiAdminMain**.setVisible(**false**);  
 **if** (**mrpUI** != **null**)  
 **mrpUI**.setVisible(**false**);  
 **if** (**db** != **null**)  
 **db**.setVisible(**false**);  
 **if** (**uiLogIN** != **null**)  
 **uiLogIN**.setVisible(**false**);  
  
 **int** dialogResult = JOptionPane.*showConfirmDialog*(**panel**,  
 **"СЕРВЕР НЕ ДОСТУПЕН\r\nПереподлючиться?"**,  
 **"Нет доступа"**,  
 JOptionPane.***YES\_NO\_OPTION***,  
 JOptionPane.***ERROR\_MESSAGE***);  
 **if** (dialogResult != JOptionPane.***YES\_OPTION***) {  
 **return**;  
 }  
 run();  
 }  
 }  
  
}

**package** ui;  
  
**public class** DatePanel **extends** JPanel {  
  
  
 **private** JDatePickerImpl **datePicker**;  
 **public** DatePanel() {  
  
 UtilDateModel model = **new** UtilDateModel();  
 Properties p = **new** Properties();  
 p.put(**"text.today"**, **"Today"**);  
 p.put(**"text.month"**, **"Month"**);

**Продолжение приложения Д**

p.put(**"text.year"**, **"Year"**);  
 JDatePanelImpl datePanel = **new** JDatePanelImpl(model, p);  
  
 **datePicker** = **new** JDatePickerImpl(datePanel, **new** DateLabelFormatter());  
 **datePicker**.setPreferredSize(**new** Dimension(90,20));  
 **datePicker**.getJFormattedTextField().setPreferredSize(**new** Dimension(90,20));  
 add(**datePicker**);  
 }  
 @Override  
 **public void** setBounds(**int** x, **int** y, **int** width, **int** height) {  
 **super**.setBounds(x, y, width, height+5);remove(**datePicker**);  
 **datePicker**.setPreferredSize(**new** Dimension(width,height));  
 **datePicker**.getComponent(0).setPreferredSize(**new** Dimension(width-height,height));  
 **datePicker**.getComponent(1).setPreferredSize(**new** Dimension(height,height));  
 add(**datePicker**);  
 }  
   
 **public** String getText(){  
 **return datePicker**.getJFormattedTextField().getText();  
 }  
  
 **public void** setValue(Object value){  
 **datePicker**.getJFormattedTextField().setValue(value);  
 }  
 **private class** DateLabelFormatter **extends** JFormattedTextField.AbstractFormatter {  
  
 **private** String **datePattern** = **"yyyy-MM-dd"**;  
 **private** SimpleDateFormat **dateFormatter** = **new** SimpleDateFormat(**datePattern**);  
  
 @Override  
 **public** Object stringToValue(String text) **throws** ParseException {  
 **return dateFormatter**.parseObject(text);  
 }  
  
 @Override  
 **public** String valueToString(Object value) {  
 **if**(value **instanceof** String)  
 **return** (String) value;  
 **if** (value != **null**) {  
 Calendar cal = (Calendar) value;  
 **return dateFormatter**.format(cal.getTime());  
 }  
  
 **return ""**;  
 }  
  
 }  
}