СОДЕРЖАНИЕ

[Введение 8](#_Toc127119621)

[1 Анализ и моделирование предметной области программного средства 9](#_Toc127119622)

[1.1 Описание предметной области 9](#_Toc127119623)

[1.2 Разработка функциональной модели предметной области 9](#_Toc127119624)

[1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 14](#_Toc127119625)

[1.4 Разработка информационной модели предметной области 16](#_Toc127119626)

[1.5 UML-модели представления программного средства и их описание 18](#_Toc127119627)

[1.5.1 Диаграмма последовательности 18](#_Toc127119628)

[1.5.2 Диаграмма деятельности 20](#_Toc127119629)

[1.5.3 Диаграмма развертывания 21](#_Toc127119630)

[2 Проектирование и конструирование программного средства 23](#_Toc127119631)

[2.1 Постановка задачи 23](#_Toc127119632)

[2.2 Обоснование выбора компонентов и технологий для реализации программного средства 24](#_Toc127119633)

[2.3 Архитектурные решения 25](#_Toc127119634)

[2.4 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства 28](#_Toc127119635)

[2.5 Проектирование пользовательского интерфейса 30](#_Toc127119636)

[2.6 Методы и средства, используемые для обеспечения безопасности данных 31](#_Toc127119637)

[3 Тестирование и проверка работоспособности программного средства 32](#_Toc127119638)

[4 Руководство по развертыванию и использованию программного средства 36](#_Toc127119639)

[4.1 Методы и средства, используемые для обеспечения безопасности данных 36](#_Toc127119640)

[4.2 Руководство пользователя 36](#_Toc127119641)

[Заключение 54](#_Toc127119642)

[Список использованных источников 55](#_Toc127119643)

[Приложение А (обязательное) Отчет о проверке на заимствование в системе «Антиплагиат» 56](#_Toc127119644)

[Приложение Б (обязательное) Листинг кода алгоритмов, реализующих бизнес-логику 57](#_Toc127119645)

[Приложение В (обязательное) Листинг скрипта генерации базы данных 68](#_Toc127119646)

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. |
| *ORM (объектно-реляционное отображение)* | – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». |
|  |  |
|  |  |

# Введение

Качество обучения и особенно эффективность использования научно-педагогического потенциала зависят в определенной степени от уровня организации учебного процесса. Одна из основных составляющих этого процесса – расписание занятий – регламентирует трудовой ритм, влияет на творческую отдачу преподавателей, поэтому его можно рассматривать как фактор оптимизации использования ограниченных ресурсов – преподавательского состава и аудиторного фонда.

Проблему составления расписания следует воспринимать не только как трудоемкий процесс, объект автоматизации с использованием ЭВМ, но и как акцию оптимального управления. Поскольку все факторы, влияющие на расписание, практически невозможно учесть, а интересы участников учебного процесса многообразны, задача составления расписания является многокритериальной с нечетким множеством факторов. Решение таких задач, как правило, осуществляется в два этапа: получение оптимального (с точки зрения используемых критериев) варианта и его последующая доработка человеком (диспетчером) с целью максимального учета неформализованных факторов.

Поэтому целью данного курсового проекта будет создание программного средства, которое будет учитывать различные критерии и на их основе формировать начальное оптимальное расписание, и предоставлять возможность корректировать расписание.

Для достижения поставленной цели необходимо выполнить ряд задач:

* изучить алгоритм, требования, критерии оптимального составления расписания;
* 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.

Курсовой проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности составляет 97,84%. Цитирования обозначены ссылками на публикации, указанными в «Списке использованных источников». Скриншот приведен в приложении А.

# Анализ и моделирование предметной области программного средства

## Описание предметной области

Cуществующая теория расписаний применима при составлении расписаний работы машин в цехах, и в то же время имеет существенные ограничения по применению для составления расписания занятий в вузе. В настоящей главе предложено решение первого этапа проблемы – разработка алгоритма получения оптимального расписания. Для разработки алгоритма в первую очередь были выделены требования к расписанию занятий. В основу алгоритма была положена идея оценки свободы расположения занятий в расписании, после чего разработана структурная схема алгоритма, и описаны все этапы его работы, а также был предложен вариант повышения качества составленного расписания при помощи генетических алгоритмов.

**Требования к расписанию**

При составлении расписания возникает проблема оптимального управления ресурсами: преподавательским составом и аудиторным фондом. В процессе решения задачи необходимо учитывать обязательные ограничения, а также дополнительные требования, которые могут нарушаться в некоторых случаях. К обязательным ограничениям относятся:

- вместительность аудиторий должна быть достаточной для групп, которые в ней занимаются, при этом возможен вариант, когда в одной аудитории проводятся занятия одновременно для нескольких групп студентов;

- должны выполняться требования занятий к оборудованию аудиторий, в которых они проводятся;

- преподаватели из других вузов могут проводить занятия только в определенные дни и часы.

К дополнительным требованиям относятся:

- лекции должны проводиться в начале дня, практики – в конце;

- нагрузка каждой группы должна быть равномерной, во избежание переутомления студентов, то есть в те дни, когда проводится лекция по сложному предмету, остальные занятия должны проводиться по относительно простым;

- в занятиях студентов не должно появляться окон, в то же время возможно наличие окна в расписании преподавателя;

- по возможности преподавателям должны предоставляться дни, свободные от проведения занятий в вузе;

- в пятницу количество занятий должно быть меньше, чем в остальные дни недели;

- первым занятием в понедельник должен идти относительно простой предмет, иначе успеваемость студентов может существенно снизиться.

**Разработка алгоритма**

В результате анализа требований к расписанию занятий было принято решение о необходимости в разработке алгоритма, в котором были бы заложены возможности по расширению списка требований к расписанию занятий, а также возможности регулирования приоритетов выполнения отдельных требований при составлении расписания. В основе предлагаемого алгоритма составления расписания была положена идея оценки свободы расположения отдельного занятия в полученном расписании. Было установлено, что занятия, для проведения которых требуется выполнение обязательных требований по специальному оборудованию, могут быть проведены только в существенно ограниченном количестве аудиторий, а занятия, которые проводит преподаватель, приходящий только в определенные дни недели, могут быть проведены только в определенные дни недели, а, следовательно, такие занятия имеют меньшую свободу расположения в расписании. Для успешного размещения занятий с относительно малой свободой расположения в расписании решено начинать составление расписания с добавления в него занятий с наименьшими оценками свободы расположения. На рис. 1 приведена структурная схема предлагаемого алгоритма.

Составление расписания занятий начинается с ввода данных об учебной нагрузке. Также для работы алгоритма требуются данные об аудиторном фонде, в случае если они не были введены ранее – производится их ввод. В зависимости от вариантов реализации пользовательского интерфейса списки преподавателей и групп могут формироваться автоматически, на основании данных об учебной нагрузке, и после этого дополняться недостающими данными, либо могут быть введены в ручном режиме.

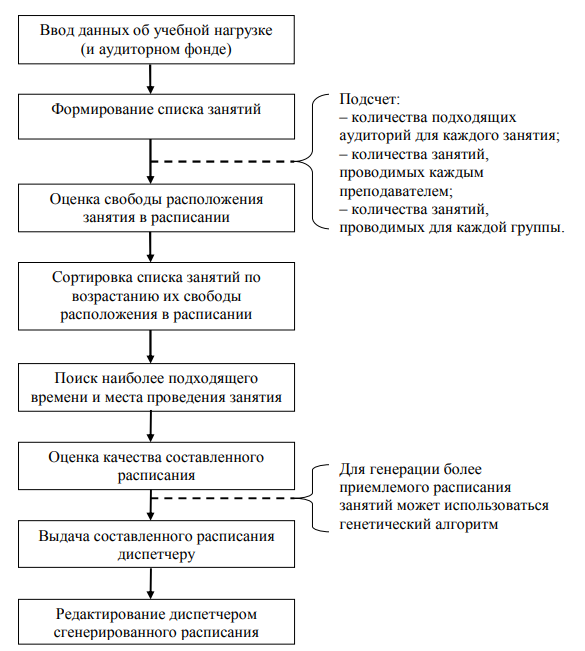


Рисунок 1.1 – структурная схема составления расписания

В результате обработки данных об учебной нагрузке потоков формируется список занятий (см. таблицу 1).

Таблица 1 – Пример составленного списка занятий

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа | Название дисциплины | Тип занятия | Преподаватель | Дополнительные требования к аудитории |
| 073601 | ВДиШП | Лекция | Логинова И.П. | Нет |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа | Название дисциплины | Тип занятия | Преподаватель | Дополнительные требования к аудитории |
| Нет073601 | ВДиШП | Лекция | Логинова И.П. | Нет |
| 073601 | ВДиШП | Практика | Милентьев В.А | Компьютеры |
| 073601 | Физкультура | - | - | Спортивный инвентарь |
| 073603 | УКвЛ | Лекция | Шевченко В.И. | Нет |
| 073603 | УКвЛ | Практика | Шевченко В.И. | Компьютеры |

Как видно из примера, в списке занятий могут встречаться повторения. Это обусловлено тем, что каждое занятие является отдельным объектом, который необходимо расположить в расписании. Дублирование происходит вследствие того, что в течение одной недели проводятся два занятия одинакового типа.

При оценке свободы расположения *i*-го занятия в расписании производится подсчет:

- количества аудиторий *ai*, подходящих для проведения занятия, на основании требований занятия к оборудованию аудитории и количеству рабочих мест;

- количества занятий в неделю *pi*, которые проводит преподаватель этого занятия;

- количества занятий в неделю *gi* для заданной группы студентов.

На основании этих данных определяется оценка свободы расположения занятия в расписании:

где *Si* – оценка свободы расположения *i*-го занятия в расписании;

*n* – количество занятий.

После оценивания свободы расположения занятий в расписании производится сортировка списка занятий по возрастанию их оценок свободы расположения:

, (2)

После проведения сортировки в расписание в первую очередь добавляются занятия, находящиеся в голове списка, то есть с наименьшей свободой расположения в расписании. При добавлении занятий в расписание производится поиск наиболее выгодной аудитории и времени для ее проведения. Для этого необходим полный перебор вариантов проведения занятия в пространстве (аудитории) и времени (номер пары, день недели). В процессе перебора вариантов расположения занятия – в первую очередь происходит проверка возможности проведения занятия по трем условиям:

* + - 1. не происходит «перекрытия» занятий. В случае если оно произошло, то занятие не может быть проведено;
      2. аудитория оборудована всем необходимым для проведения занятия, например компьютерами, стендами для проведения экспериментов проектором и т.д.;
      3. количество рабочих мест в аудитории не меньше количества учащихся в группе.

В случае если обязательные условия выполняются, происходит оценка качества расположения занятия по следующим критериям:

1. появление окна в расписании группы студентов;
2. появление окна в расписании преподавателя;
3. избыточность количества мест в аудитории по отношению к количеству

учащихся;

1. проведение занятия в неудачное время, например четвертым или пятым по счету в этот день для этой группы студентов;
2. исчезновение окна в расписании группы студентов;
3. исчезновение окна в расписании преподавателя;
4. исчезновение окна в расписании использования аудитории.

Каждый из критериев оценки качества расположения занятия в расписании дол жен быть реализован в виде отдельной специализированной функции, возвращающей значения в диапазоне [0;1]. Гибкость и адаптационные способности алгоритма достигнуты за счет возможности дополнять список критериев качества без существенных изменений программного кода. Одним из возможных дополнений может быть проверка необходимости перемещения группы студентов или преподавателя между корпусами

вуза для проведения занятия.

Оценка качества расположения занятия по всем критериям может быть использована для получения общей оценки, необходимой в дальнейшем для выбора максимально выгодного времени и места проведения занятия.

Для получения оценки качества расположения занятия в расписании применяется формула вида:

где *Ril* – качество расположения *i*-го занятия на l-й позиции в расписании;

*kjl* – значение, полученное по *j*-му критерию оценки качества расположения занятия на l-й позиции в расписании;

*wj* – весовой коэффициент j-го критерия оценки качества;

*m* – количество критериев оценки качества.

После оценки качества всех возможных вариантов расположения занятия в расписании выбирается вариант, при котором достигается максимальное значение оценки качества расположения:

(4)

где *l* – возможная позиция *i*-го занятия в расписании;

*Ri* – качество расположения *i*-го занятия в расписании;

*h* – количество возможных вариантов расположения занятия в расписании.

После расположения всех занятий в расписании производится оценка качества составленного расписания. Для оценки качества расписания решено использовать сумму оценок качества расположения всех занятий в расписании. Ввиду того, что оценка качества расположения каждого занятия по критериям появления и исчезновения окна в расписаниях студентов и преподавателей зависит от взаимного расположения занятий в расписании – необходимо произвести повторную оценку качества расположения каждого занятия в составленном расписании.

Для оценки качества составленного расписания используется формула вида:

где *R* – качество составленного расписания;

*n* – количество занятий.

Полученные результаты работы алгоритма предоставляются диспетчеру, который решает, стоит ли провести повторную генерацию расписания с новыми настроечными коэффициентами *wj*, либо модифицировать полученное расписание вручную с целью дальнейшего использования.

**Генетический алгоритм**

Для повышения качества полученного расписания занятий, основной алгоритм составления расписания может быть усовершенствован. Ввиду наличия набора настроечных коэффициентов алгоритма, а также критериев оценки качества полученного расписания, возможно использовать генетический алгоритм.

Предлагается использовать в качестве генов настроечные коэффициенты wj алгоритма составления расписания. В результате мутаций будут получены новые наборы настроечных коэффициентов и как следствие – результаты работы алгоритма будут различны. При этом оценку качества составленного расписания следует проводить с использованием значений wj заданных диспетчером. Таким образом, учитывается возможность генерации расписания наиболее удовлетворяющего оценкам качества с точки зрения диспетчера, при использовании несколько измененных оценок качества в процессе его генерации.

Хромосома такого генетического алгоритма будет представлять собой набор действительных чисел:

(6)

где *w1…wm* – гены хромосомы, настроечные коэффициенты алгоритма составления расписания;

*m* – количество настроечных коэффициентов алгоритма.

Целью работы генетического алгоритма является достижение максимума функционала:

(7)

где *Ril* – качество расположения *i*-го занятия на l-й позиции в расписании;

*kjl* – значение, полученное по *j*-му критерию оценки качества расположения занятия на l-й позиции в расписании;

*wj* – весовой коэффициент *j*-го критерия оценки качества;

*m* – количество критериев оценки качества;

*l* – возможная позиция *i*-го занятия в расписании;

*Ri*– качество расположения *i*-го занятия в расписании;

*h* – количество возможных вариантов расположения занятия в расписании;

*R* – качество составленного расписания.

**Заключение**

Предложенный алгоритм может быть использован для составления расписания занятий в отдельном учреждении образования. Среди преимуществ можно отметить возможность генерации приемлемых вариантов расписания уже с первой итерации. В алгоритме предусмотрена возможность значительного улучшения расписания благодаря добавлению дополнительных критериев оценки свободы и качества расположения занятий в расписании. Например, можно модифицировать алгоритм для составления расписания занятий сразу в нескольких корпусах вуза, добавив при этом дополнительный коэффициент качества расположения занятия, который будет учитывать количество вынужденных перемещений студентов и преподавателей между корпусами – с целью их минимизации. Использование генетического алгоритма в системе составления расписания может значительно улучшить качество составляемого расписания. Реализация алгоритма на начальном этапе подразумевает его работу на локальной ЭВМ с использованием одной из существующих реализаций баз данных с диалоговым интерфейсом взаимодействия.

## Разработка функциональной модели предметной области

Основным бизнес-процессом данного курсового проекта является конструирование одежды из вязанного трикотажа регулярным способом (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 выступает вид изделия (например шапка, кардиган, штаны) и пряжа. Вместе с видом изделия в процесс поступают мерки, которые необходимо снять с человека, однако они оказывают управляющее воздействие на систему. К управляющим воздействиям также относится внутренний регламент предприятия и госты. В гостах описаны типовые фигуры людей разного пола и возраста. Госты 17521(2)-72 [2] «Типовые фигуры мальчиков и мужчин. Размерные признаки для проектирования одежды», госты 17916(7)-86 [3] «Типовые фигуры девочек и женщин. Размерные признаки для проектирования одежды», гост 50713-94 [4] «Изделия для новорожденных и детей ясельной группы». В качестве механизма осуществления главной функции выступают вязальщица, которая участвует во всех под-процессах, инструменты для вязания и расчетов. Выходным параметром для данной системы являются готовые выкройки однако они немного отличаются от привычного понимания. Обычно выкройки — это детали будущего изделия, выполненные из бумаги, кальки, ткани, кожи или других материалов, необходимые для дальнейшего раскроя ткани. Согласно регулярному способу производства мы будем вывязывать изделие полностью на вязальной машине, а для этого нам необходимо знать сколько набрать петель, сколько рядов провязать и так далее. Поэтому в данном случае выкройка – это чертеж изделия с нанесенными на нем размерами, заданными в рядах и петлях(столбцах).

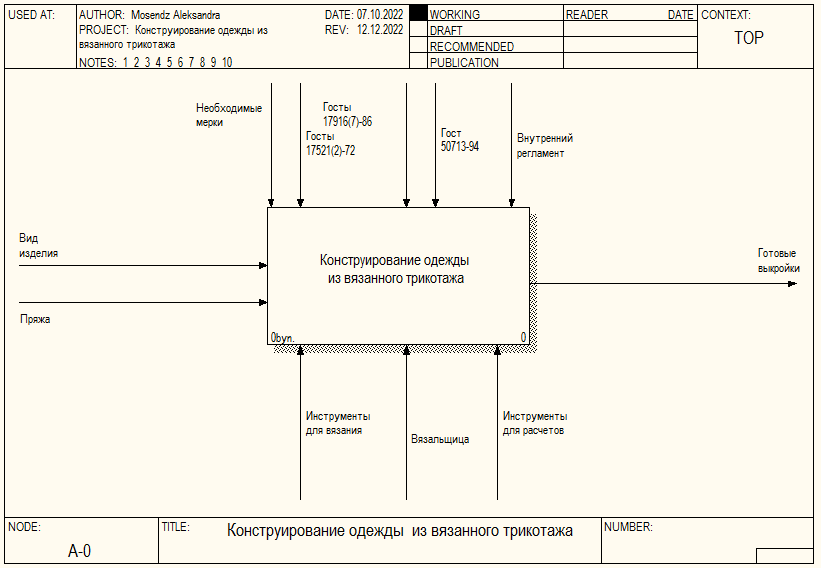


Рисунок 1.1 – Контекстный уровень диаграммы

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

1. Изготовление образца для снятия петельной пробы.
2. Снятие мерок.
3. Составление выкроек.

Нельзя сразу рассчитать выкройку, необходимо вначале сделать образец – вязанное полотно небольшого размера, для последующего вычисления всех необходимых значений. Потом снять необходимые мерки с человека и рассчитать выкройку.

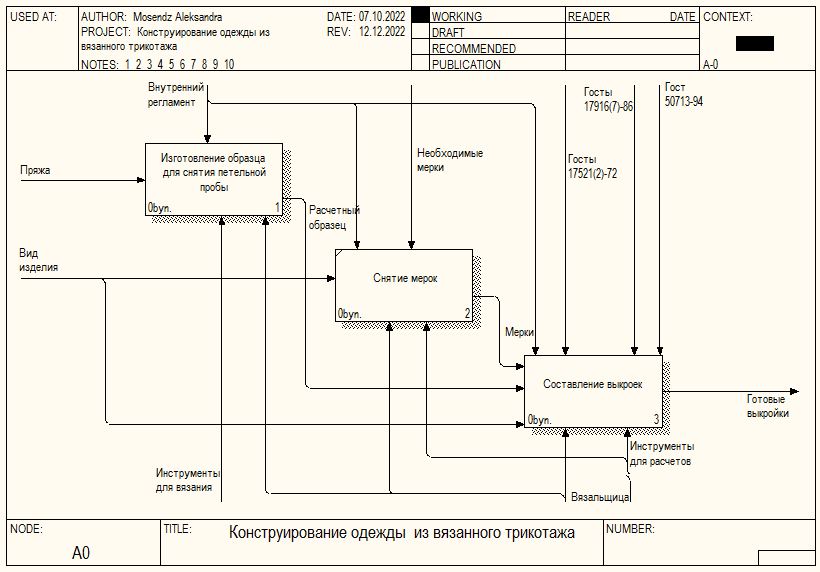


Рисунок 1.2 – Декомпозиция контекстной диаграммы

Этап «Изготовления образца для снятия петельной пробы» разбит на четыре функциональных блока (рисунок 1.3):

1. Выбор размеров образца.
2. Выбор плотности.
3. Изготовление образца.
4. Проведение ВТО.

Обычно изготавливают несколько образцов небольшого размера, чтобы найти подходящую плотность. Все образцы необходимо постирать и только после стирки оценивать подходит плотность или нет. После определения плотности изготавливают финальный образец, размер которого зависит от толщины нити и размера будущего изделия.

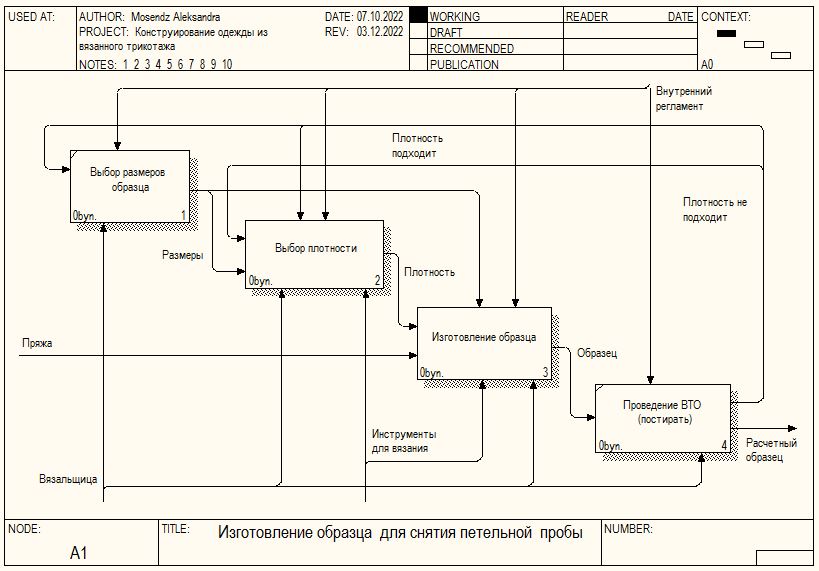


Рисунок 1.3 – Декомпозиция блока «Изготовления образца для снятия петельной пробы»

Этап «Изготовления образца» разбит на четыре функциональных блока (рисунок 1.4):

1. Настроить оборудование.
2. Набрать петли.
3. Выбрать образец.
4. Закрыть петли.

Изготовление образца это самая простая часть, нужно настроить вязальную машину, так как они могут сильно отличатся друг от друга сведем процесс настройки к выбору плотности, обычно это круглое колесо с цифрами которое нужно повернуть на определенный угол. Далее необходимо выдвинуть иглы в работу, набрать петли и провязать полотно на нужную длину, далее закрыть петли. Образец до проведения ВТО готов.

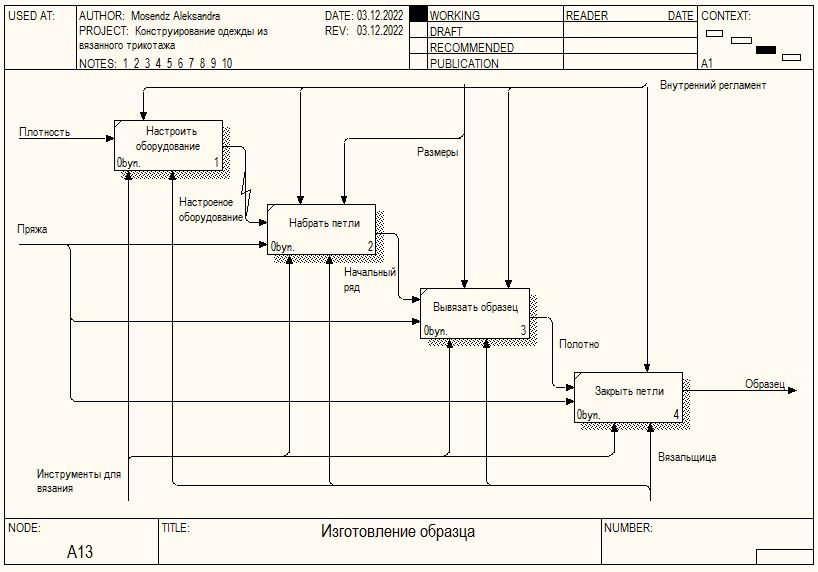


Рисунок 1.4 – Декомпозиция блока «Изготовление образца»

Этап «Составления выкроек» разбит на четыре функциональных блока (рисунок 1.5):

1. Расчет петельной пробы.
2. Определение особенностей изделия.
3. Составление чертежа изделия (размеры в см).
4. Перевод значений относительно петельной пробы.

Полотно вязанного изделия в упрощенном виде можно представить как таблицу, в которой одна ячейка имеет свою высоту и ширину. Для расчета необходимо измерить сколько сантиметров образец в длину и ширину, далее посчитать количество рядов и столбцов, поделить количество на длины и мы получим плотность в 1 сантиметре по вертикали и горизонтали.

На этапе определения особенностей изделия мы учитываем особенности строения человека, например при изготовлении свитера для девушки часть петель со спины перемещают на грудь, а у мужчин наоборот.

Чертеж - это изделие на плоскости, на нем подписываются все необходимые значения, а потом зная петельную пробу эти значения переводятся в ряды и столбцы, итоговые значения не могут быть дробными.

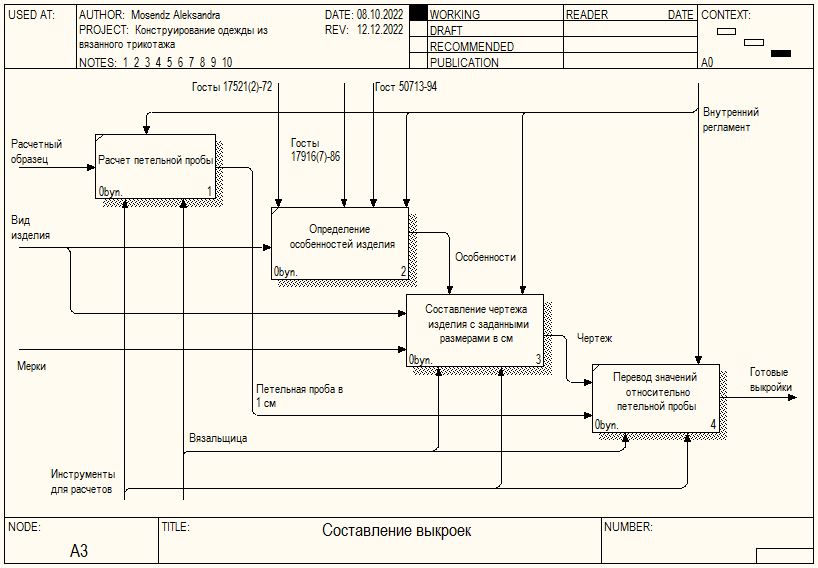


Рисунок 1.5 – Декомпозиция блока «Изготовление образца»

Составление выкройки включает в себя множество вычислений, ошибка в которых приводит к изготовлению неправильного изделия, можно автоматизировать этот процесс, что снизит число ошибок допущенных при вычислениях.

## 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).

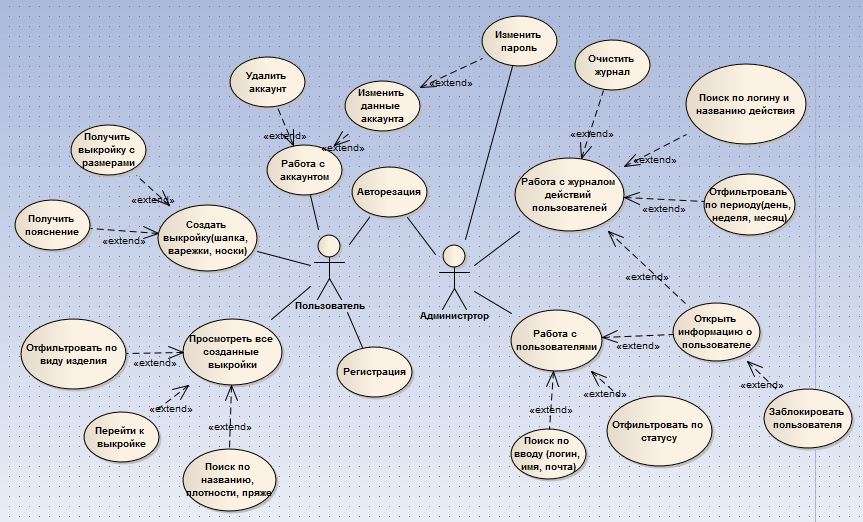


Рисунок 1.6 – Диаграмма вариантов использования

При первом входе в приложение пользователь должен зарегистрироваться, после этого авторизоваться В приложении запрограммированы различные выкройки и пользователь может создать любую из них, в дальнейшем при заполнении всех необходимых полей он может получить расчеты и пояснения к выкройке, сохранить данные и в последующем открывать данную выкройку. Чтобы среди созданных выкроек найти определенную, есть возможность отфильтровать данные по виду изделия или по введенным данным (поиск производиться по всем полям кроме вида изделия). При необходимости пользователь может изменить данные о себе, свой логин и пароль в личном аккаунте. Для изменения пароля необходимо ввести старый пароль, при нежелании дальше взаимодействовать с приложением, пользователь может удалить свой аккаунт.

Администратор имет возможности наблюдать за активностью пользователей, посредством журнала действий, в который автоматически заносятся все важные события в приложении. Для удобной навигации по журналу, можно вывести данные за определенный период времени (день, недулю, месяц, все время) или отфильтровать данные по введенному значению (поиск производится по логину и названию действий). Администратор может выбрать любое событие и перейти на страницу пользователя совершившего его. На сранице будет представлена информация о пользователе (логин, имя, почта и статус, заблокирован пользователь или нет) и часть журнала с его действиями, есть возможность иззменить статус пользователя, то есть заблокировать или разблокировать. Администрирор так же может просмотреть все записи (аккаунты) пользователей, для навигации можно отфильтровать данные по статусу пользователей (заблокирован или нет) или по введенным данным (поиск производится по логину, имени и почте), так же можно перейти на страницу пользователя выбрав запись.

## Разработка информационной модели предметной области

При проектировании системы было принято решение использовать следующие сущности:

* *Users*;
* *Pattern*;
* *Actions*;
* *Socks*;
* *Mittens*;
* *Hat*.

Графическое отображение информационной модели приведено на диаграмме рисунке 1.7.

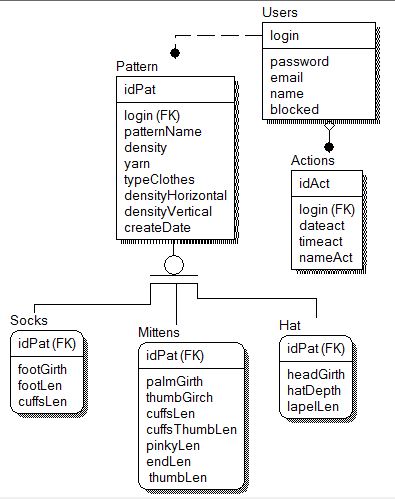


Рисунок 1.7 - Информационная модель системы

Информация о пользователях хранится в сущности «Users». Данные в нее вносятся при регистрации, но так же могут изменятся в последующем.

Для хранения общей информации о выкройках есть сущность «Pattern», Она исключает повторение данных.

Все важные действия пользователей вносятся в таблицу «Actions», например создание, изменение удаление выкройки.

И самое важное это сущности отвечающие за выкройки, в них хранятся значения всех необходимых мерок.

Ниже представлено подробное описание всех сущностей, входящих в модель.

Сущность *Users* содержит в себе следующие атрибуты:

* *login*– хранит уникальный логин пользователя;
* *password* –хранит пароль;
* *email* – хранит почту, необязательно для заполнения;
* *name* –хранит имя пользователя, необязательно для заполнения;
* *blocked*–хранит статус пользователя, заблокирован ли он.

Сущность *Pattern* содержит в себе следующие атрибуты:

* *idPat*– хранит уникальный идентификатор выкройки;
* *login(FK)* - атрибут унаследованный от сущности *Users*
* *patternName* – Название выкройки введенное пользователем;
* *density* – плотность или номер спиц;
* *yarn* – информация о пряже;
* *typeClothes* – вид одежды(шапка, варежки);
* *densityHorizontal* – плотность по горизонтали в 1 см;
* *densityVertical* – плотность по вертикали в 1 см;
* *createDate* – дата создания;

Сущность *Socks* содержит в себе следующие атрибуты:

* *idPat*– атрибут унаследованный от сущности *Pattern*;
* *footGirth* – обхват стопы;
* *footLen* – длина стопы;
* *сuffsLen* – высота манжета;

Сущность *Mittens* содержит в себе следующие атрибуты:

* *idPat*– атрибут унаследованный от сущности *Pattern*;;
* *palmGirth* – обхват ладони;
* *thumbGirth* – обхват большого пальца;
* *cuffsLen*- высота манжеты;
* *cuffsThumbLen*- длина от манжета до основания большого пальца;
* *pinkyLen*- длина от манжета до конца мизинца;
* *endLen* – длина ладони;
* *thumbLen* – длина большого пальца;

Сущность *Hat* содержит в себе следующие атрибуты:

* *idPat*– атрибут унаследованный от сущности *Pattern*;;
* *headGirth* – окружность головы;
* *hatDepth* – глубина шапки;
* *lapelLen*- высота отворота;

Сущность *Actions* содержит в себе следующие атрибуты:

* *idAct*– уникальный идентификатор события;
* *login* - атрибут унаследованный от сущности *Users*;;
* *dateact* – дата совершения события;
* *timeact –* время совершения события
* *nameAct*- название события;

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.

## UML-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 вершинам (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

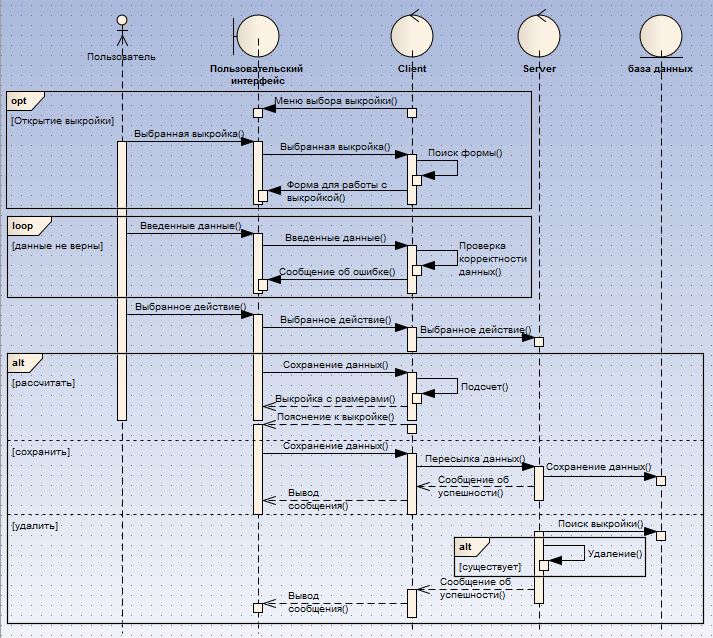


Рисунок 1.8 – Диаграмма последовательности процесса составления выкройки

На данной диаграмме пользователь выбирает изделие (выкройку) и вводит необходимую информацию в окно пользовательского интерфейса, по выбору пользователя можно рассчитать выкройку, сохранить данные или удалить их. Если пользователь выбрал рассчитать, данные считываются обработчиком на стороне клиента и вызывается функция подсчета, после выполнения функции на экране отобразится выкройка с рассчитанными параметрами и под ней описание к данной выкройке. Если пользователь выбрал сохранить, данные считывает обработчик и передает из по протоколу TCP на сервер. Сервер подключается к базе данных, проверяет существует ли данная выкройка и в зависимости от результатов обновляет данные или сохраняет новые данные в базу и отправляет Клиенту сообщение об успешности операции. Если пользователь выбрал удалить выкройку, данные считывает обработчик, отправляет на Сервер. Сервер подключается к БД, проверяет существует ли данная выкройка, если существует, удаляет ее и отправляет сообщение об успешности операции.

* + 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).

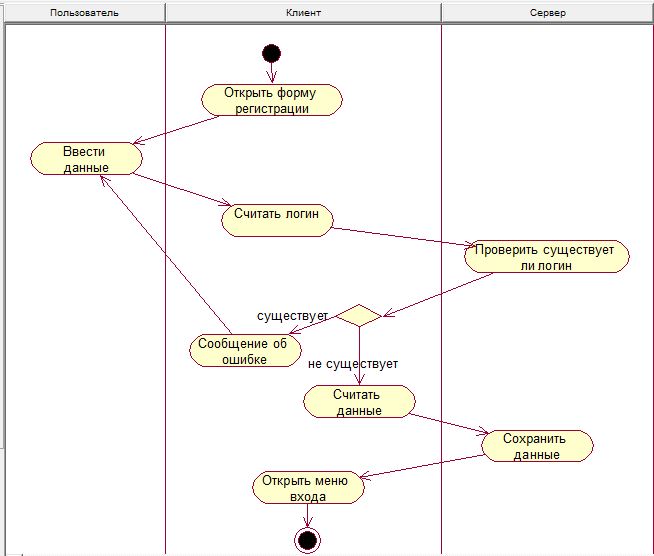


Рисунок 1.9 – Диаграмма деятельности

* + 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.

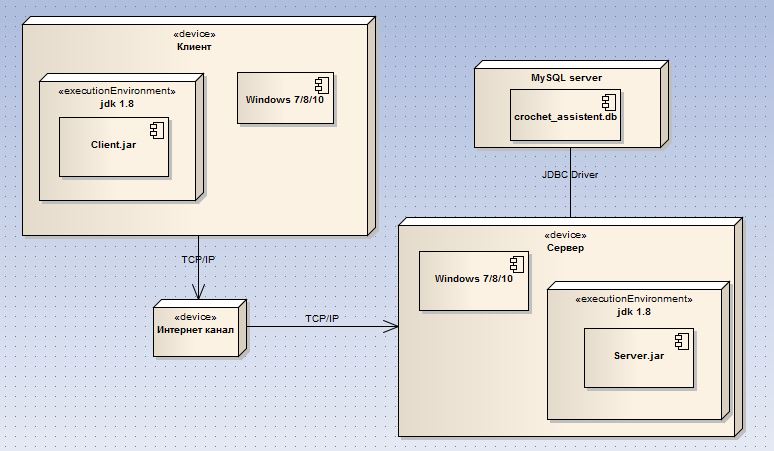


Рисунок 1.10 – Диаграмма развертывания приложения

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

Для запуска разрабатываемого приложения необходимо наличие исполняемой среды JDK18 на компьютерах пользователей приложения «Crochet Assistent». Так же на компьютерах должна стоять операционная система Windows выше 7.

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).

## 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 r7 (BPwin) используется для проектирование IDEF0 модели. CA AllFusion ERwin Data Modeler r7 (ERwin) используется для проектирования информационной модели. Microsoft Visio для проектирования блок схем и чертежей.

## Архитектурные решения

Диаграммы классов используются при моделировании программных средств наиболее часто. На диаграммах классов показываются классы, интерфейсы и отношения между ними. В данном курсовом проекте были реализованы классы *Hat, Mittens, Socks* – которые реализуют абстрактные методы интерфейса *Product* и хранят все необходимые мерки для соответствующих выкроек «Шапка», «Варежки» «Носки». Вместе с классом *Pattern* реализуют бизнес логику приложения. В классе Pattern хранится общая информация для всех выкроек (плотность, дата создания, пряжа, тип изделия и так далее).

Для работы с пользователями, в том числе с администратором создан класс *User*, он хранит информацию о пользователе (логин, пароль, почта, имя и статус, а именно заблокирован пользователь или нет). Класс *User* имеет 2 метода для работы с администратором, а именно авторизация и изменение пароля (пароль администратора хранится в файле config.txt). Для отслеживания событий реализован класс *Action*, который хранит данные о событии (дата и время совершения события, название события и кто совершил).

Класс *DBConnector* отвечает за соединение с базой данных, данный класс реализует порождающий паттерн одиночка, чтобы соединение с базой данных было только одно. Его поля:*connection*, *DATABASE\_URL, JDBC\_DRIVER, PASSWORD, USER* используются при вызове функции, которая осуществляет подключение к базе данных.

Все классы на рисунке 2.1 имплементируют интерфейс *Serializable***, необходимо для пересылки данных по сети. Сериализация (Serialization)** — это процесс, который переводит объект в последовательность байтов, по которой затем его можно полностью восстановить.

Диаграммы классов показаны на рисунках 2.1 – 2.2.

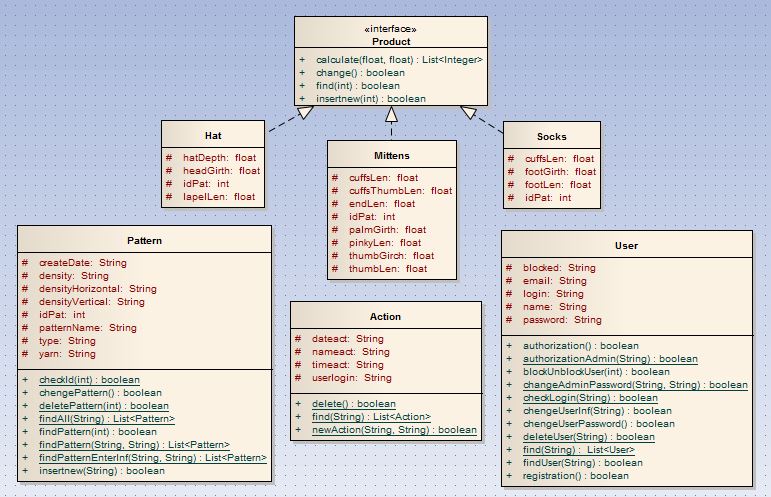


Рисунок 2.1 - Диаграмма классов пакетов bisnessLogic и user

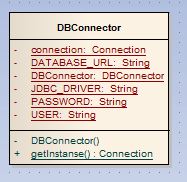


Рисунок 2.2 – Класс для соединения с базой данных

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.

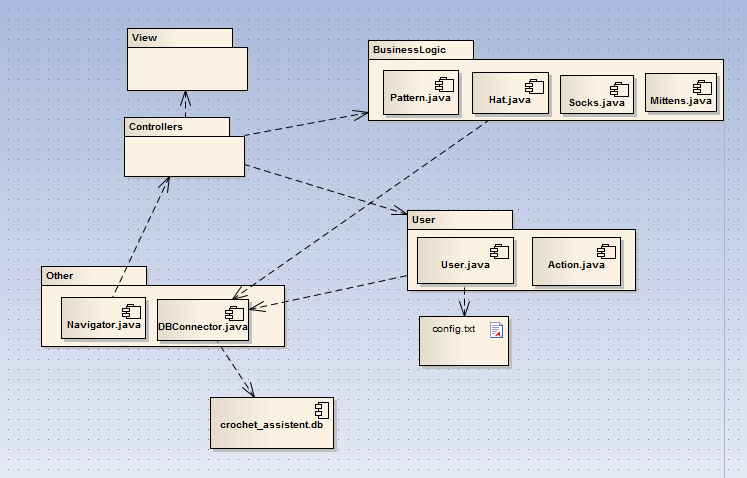


Рисунок 2.3 – Диаграмма компонентов

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 ключевую бизнес-логику разрабатываемого программного средства

Бизнес логика данного курсового проекта это составление выкроек для вязания, остальной функционал является вспомогательным, поэтому в этой главе рассмотрим три процесса: создание и работа с выкройкой и расчет выкройки на примере выкройки шапки (на высоком уровне и сами расчеты в приложении).

Когда пользователь решил создать выкройку и выбрал изделие откроется окно выкройки, пользователю нужно заполнить все поля и выбрать действие: рассчитать, сохранить, удалить. После выбора действия проверяются все данные, если они не верны, выведется сообщение об ошибке. Если данные верны выполнятся соответствующие функции расчета, сохранения ил удаления.

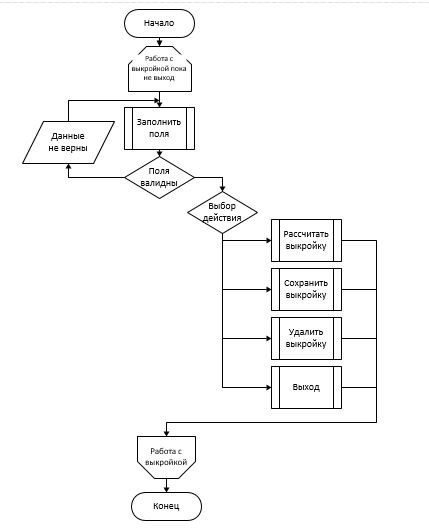


Рисунок 2.4 – Создание и работа с выкройкой

Второй процесс это расчет выкройки, это функция вызывающаяся при нажатии на кнопку «Рассчитать». Вначале из полей формы данные преобразуются в нужные типы данных и сохраняются в поля объекта типа Pattern и выкройки, например типа Hat. Далее создается List объектов типа Integer который будет хранить параметры необходимые для выкройки, вызывается метод класса Pattern, для подсчета выкройки. После выполнения функции , выводим значения из List на выкройку и генерируем пояснение (текстовая строка в которую подставляются некоторые поля из List) и выводим его пользователю.

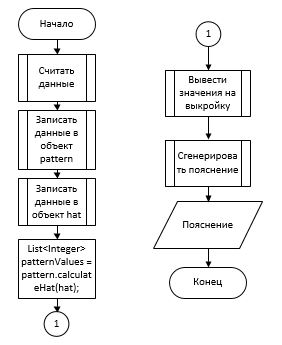


Рисунок 2.5 – Расчет выкройки

В классе каждого объекта, отвечающего за выкройку есть метод calculate, который рассчитывает значения для выкройки. На вход принимает два значения *densityVertical* (плотность в 1 сантиметре по вертикали) и *densityHorizontal* (плотность в 1 сантиметре по горизонтали) и возвращает List<Integer>. Вначале создается объект типа List<Integer>, далее вычисляются необходимые значения:

* petalW - ширина одного лепестка макушки;
* hatW - ширина всей шапки;
* hatH – высота шапки;
* topHatH – высота макушки;
* countTighten – количество петель для стягивания;
* hBeforeTopHat – высота макушки.

Значения записываются в объект типа List<Integer>, и возвращаются как результат функции.

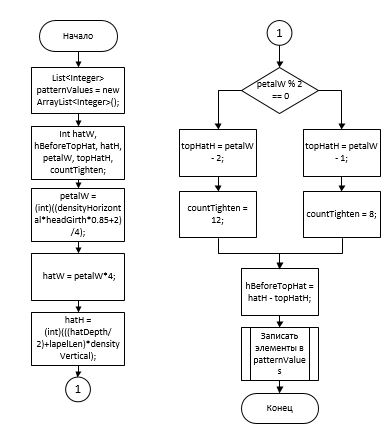


Рисунок 2.6 – Расчет параметров для выкройки

## 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.

## Методы и средства, используемые для обеспечения безопасности данных

# 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 подсчета параметров для выкроек был написан класс *TestCalculate,* с тремя методами для выкройки шапки, носков и варежек. Для проверки методов, которые проверяют вводимые значения был написан класс *TestValidator,* с тремя методами для проверки почты, чисел с плавающей точкой и даты (рисунок 3.1 – 3.2).

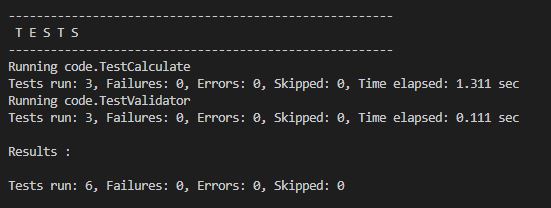


Рисунок 3.1 – Тестирование с помощью maven

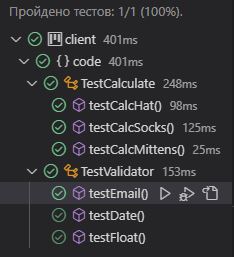


Рисунок 3.2 – Тестирование в VSCode

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;
* попытка регистрации с указанием неправильной почты.

Программа реагирует на все четыре ситуации, над кнопкой выводится сообщение об ошибке и красным подсвечиваются неправильные поля (рисунки 3.3)

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 три ситуации аналогично реакции в форме для регистрации (рисунки 3.4)

При авторизации пользователя нужно ввести лишь пароль, программа реагирует ошибкой если пароль не введен или он не правильный (рисунок 3.5).

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

Рисунок 3.3 – Ошибки при регистрации

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

Рисунок 3.4 – Ошибки при авторизации пользователя

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

Рисунок 3.5 – Ошибки при авторизации администратора

Далее была протестирован процесс создания выкройки шапки. Программа реагирует если в поля с мерками и плотностью не вводятся значения или если значения не являются числами с плавающей запятой. Поля название, плотность и пряжа являются необязательными для заполнения, для них проверок нет (рисунок 3.6).

Проверка выкроек «Варежки» и «Носки» аналогична.

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

Рисунок 3.6 – Ошибки при не заполнении формы

При изменении данных пользователя проверки аналогичны регистрации (рисунок 3.7).

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

Рисунок 3.7 –Ошибки при изменении пользовательских данных

При изменении пароля пользователя или администратора необходимо помнить старый пароль, если он не правильный выведется сообщение об ошибке (рисунок 3.7).

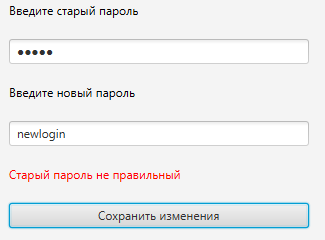


Рисунок 3.8 – Ошибки при изменении пароля

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

# Руководство по развертыванию и использованию программного средства

## Методы и средства, используемые для обеспечения безопасности данных

## Руководство пользователя

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

Далее открыть файлы condig.txt на сервере и клиенте, этом файле хранится необходимая информация о запуске. На сервере это пароль администратора, порт на котором запускается приложение, и данные для подключения к базе данных (IP адрес, порт, название базы данных, имя пользователя и пароль) (рисунок 4.1). На клиенте это IP адрес и номер порта (рисунок 4.2).

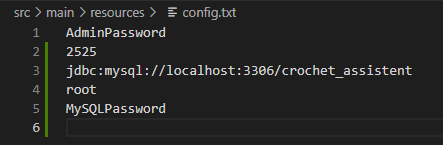


Рисунок 4.1 – Файл конфигурации на сервере

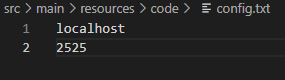


Рисунок 4.2 – Файл конфигурации на клиенте

Далее необходимо запустить сервер проекта. Для этого пользователь должен использовать *Server*.*jar* файл. Один из вариантов запуска в командной строке с помощью команд *cd* *\d* перейти в папку, в которой находиться файл, далее с помощью команды *java* –*jar* *server*.*jar* запустить сервер. Текущее состояние работы сервера отображается в консольном окне. Пример запуска показан на рисунке 4.3. Аналогично запускаем *Client.jar.* Пример отображения состояния подключений показан на рисунке 4.4.

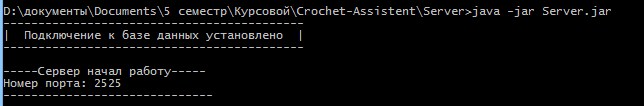


Рисунок 4.3 – Запуск сервера



Рисунок 4.4 – 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 4.5)

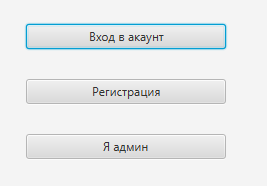


Рисунок 4.5 – Меню входа и регистрации

Допустим у нас нет аккаунта, выбираем пункт меню 2, после чего нам отобразиться форма для регистрации. Обязательными полями для вводя является логин и пароль, имя и почту можно не указывать (рисунок 4.6). При успешной регистрации нас вернет в меню входа.

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

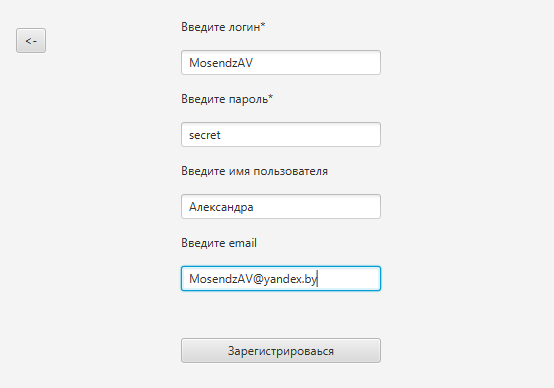


Рисунок 4.6 – Форма для регистрации

**Действия пользователя**

Вначале нам необходимо пройти процесс авторизации, выбираем кнопку «Войти в аккаунт». Заполняем поля и нажимаем кнопку «Войти» (рисунок 4.7). При успешной авторизации откроется главное меню пользователя

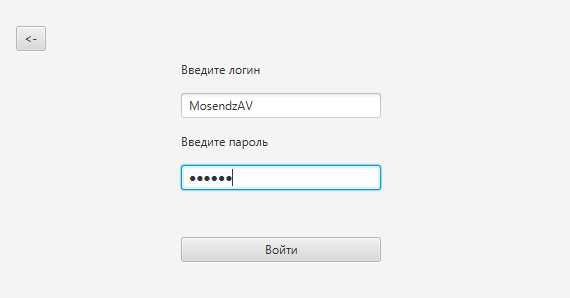


Рисунок 4.7 – Процесс авторизации пользователя

На главном меню можно увидеть все возможности пользователя (рисунок 4.8). Он может создать выкройку , открыть уже существующую и изменить данные своего аккаунта (или удалить его).

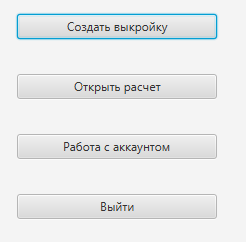


Рисунок 4.8 – Главное меню пользователя

Начнем с создания выкройки, при нажатии на данную кнопку на экране появится меню выбора выкройки. В приложении запрограммированы 3 выкройки: «Шапка», «Варежки», «Носки» (рисунок 4.9).

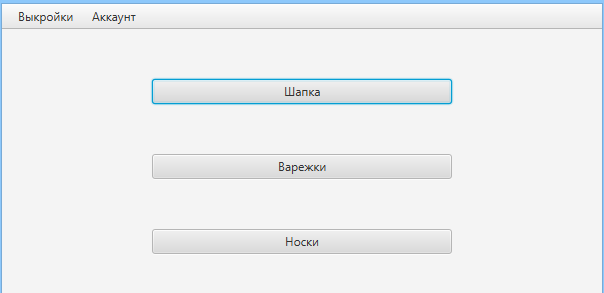


Рисунок 4.9 – Меню выбора выкройки

Для удобной навигации по приложению, сверху есть меню, разделенное на 2 отдела, в соответствии с возможностями пользователя. Чтобы выйти в главное меню пользователя нужно выбрать «Выкройки»-> «Выйти» (рисунок 4.10-4.11).

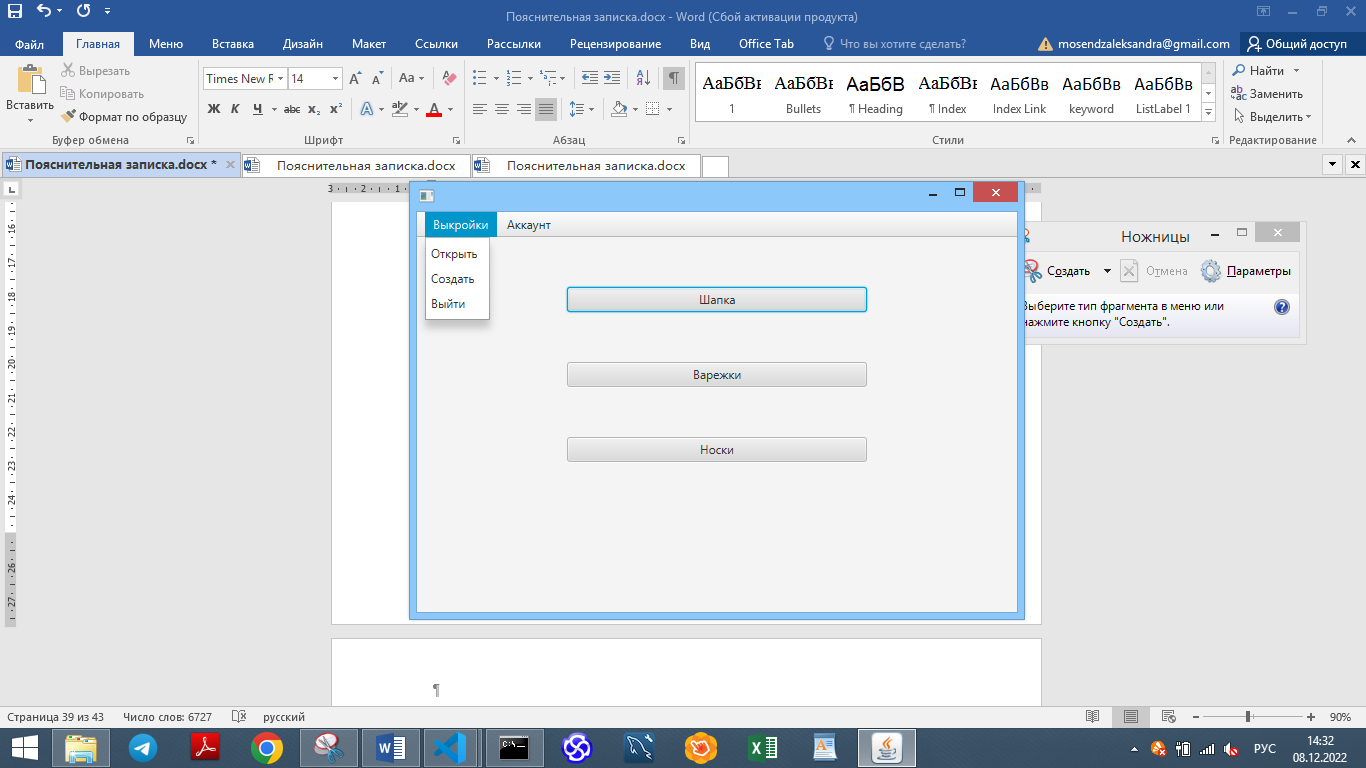


Рисунок 4.810 – Навигация по приложению, раздел «Выкройки»

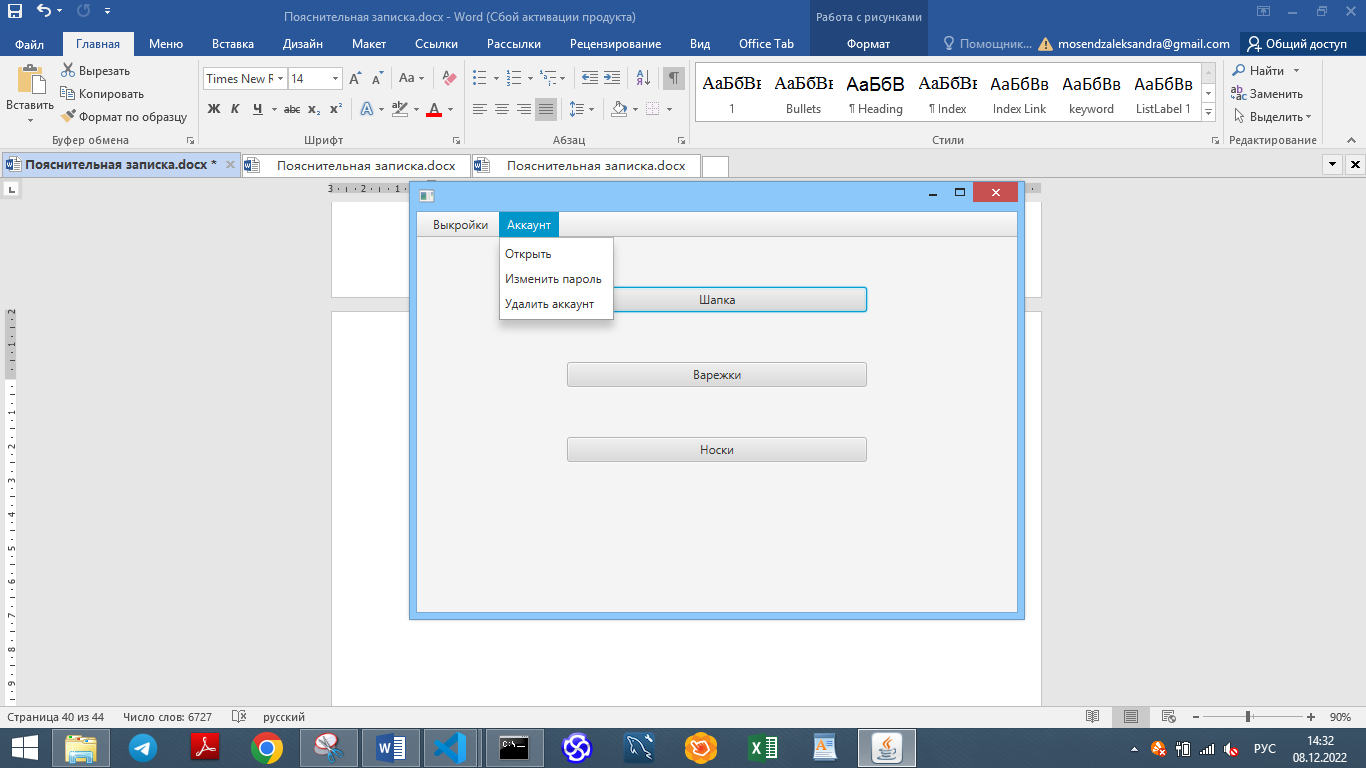


Рисунок 4.11 – Навигация по приложению, раздел «Аккаунт»

Рассмотрим создание выкройки шапки. Поля название, плотность и пряжа и высота отворота являются не обязательными, остальные поля нужно заполнить числами (дробными) (рисунок 4.12).

При нажатии на кнопку сохранить, в случае успешности операции выведется сообщение «Данные сохранены» (рисунок 4.13)

При нажатии на кнопку рассчитать (не обязательно перед эти сохранять), на выкройке отображаются значения в рядах и петлях, поэтому эти числа являются целыми. После выкройки идет ее текстовое описание (рисунок 4.14).

При нажатии на кнопку удалить, в случае успешности мы вернемся в главное меню, иначе выведется одно из двух сообщений: «Данной выкройки нет в системе», «Выкройка не удалена».

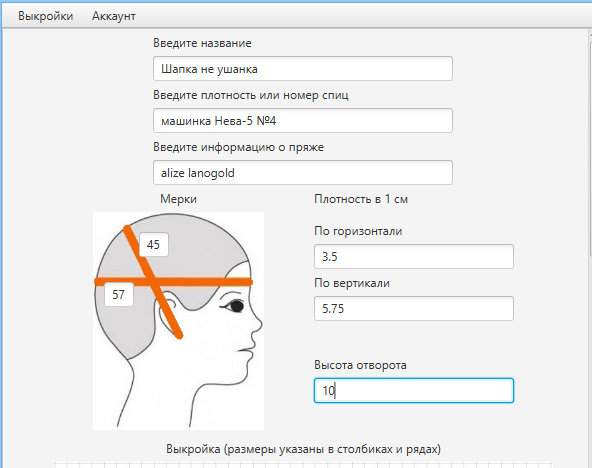


Рисунок 4.12 – Заполнение полей для создания выкройки шапки



Рисунок 4.13 – Сохранение данных о выкройке

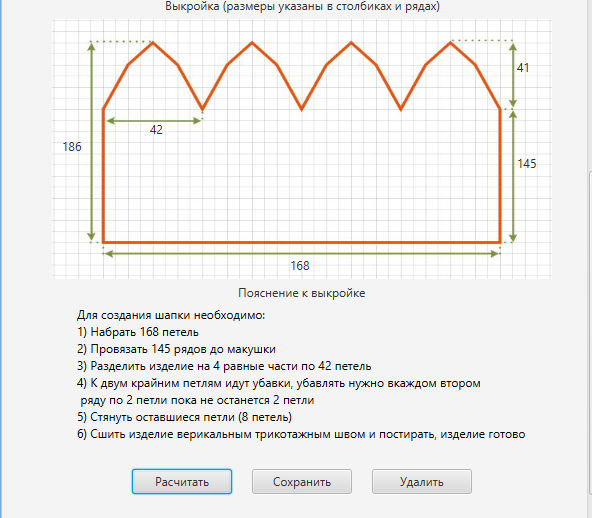


Рисунок 4.14 – Выкройка шапки и текстовое пояснение

Процесс составления остальных выкроек аналогичен, отличается лишь количество полей для ввода, выкройка и текстовое описание (рисунки 4.15 -4.20).

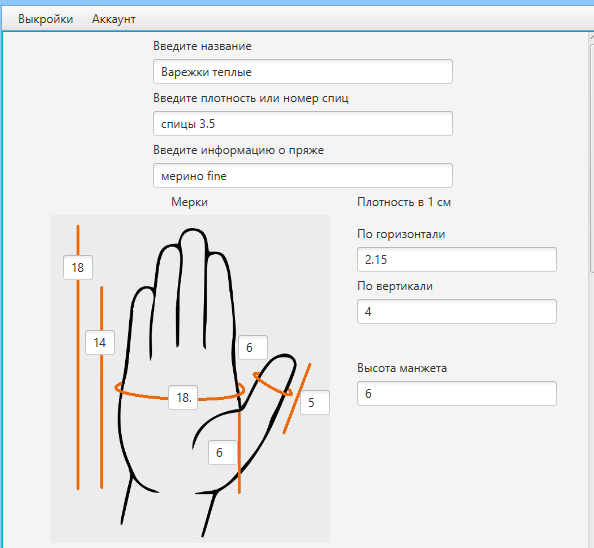


Рисунок 4.15 – Заполнение полей для создания выкройки варежек

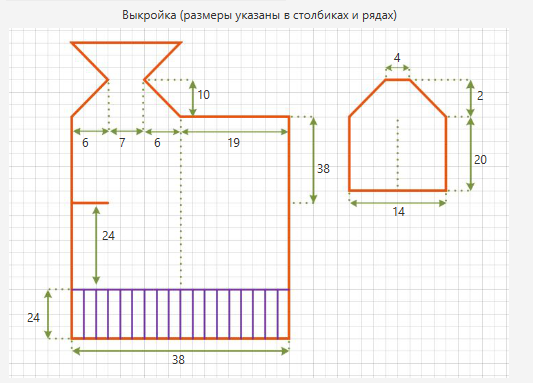


Рисунок 4.16 – Выкройка варежки

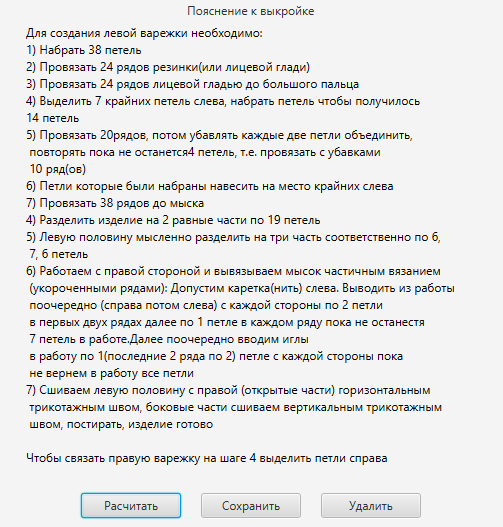


Рисунок 4.17 –Текстовое пояснение к выкройке варежки

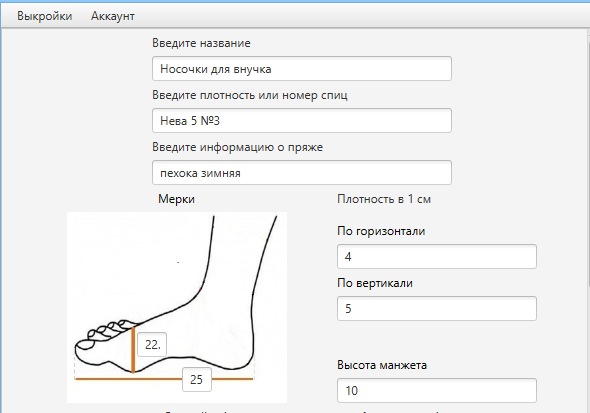


Рисунок 4.18 - Заполнение полей для создания выкройки носков

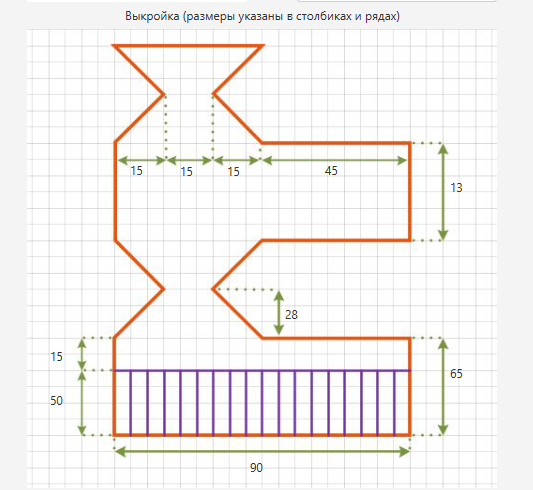


Рисунок 4.19 – Выкройка носка

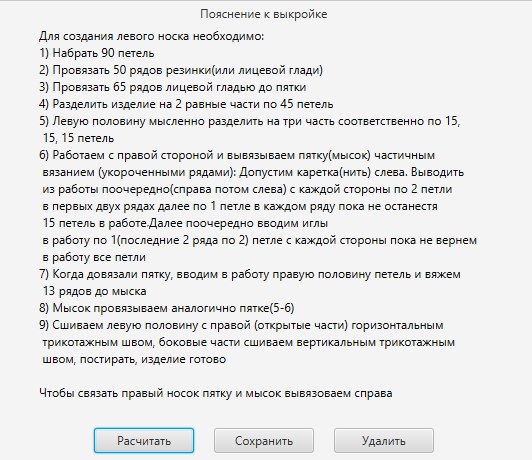


Рисунок 4.20 –Текстовое пояснение к выкройке варежки

Мы создали три выкройки (я создала еще 2 для наглядности), теперь можем просмотреть список выкроек и открыть интересующую Чтобы просмотреть выкройки нужно выбрать «Выкройки»->«Открыть», после чего отобразится таблица со всеми выкройками (рисунок 4.21).

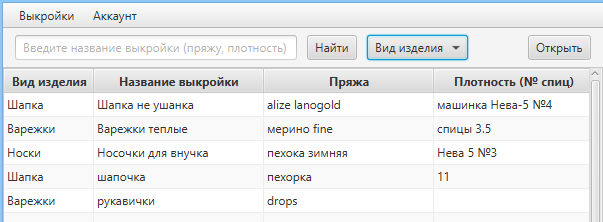


Рисунок 4.21 – Форма открытия выкроек

Можно ввести строку в поле ввода и найдутся все выкройки, у которых название, пряжа или плотность совпадает полностью или частично (рисунок 4.22).

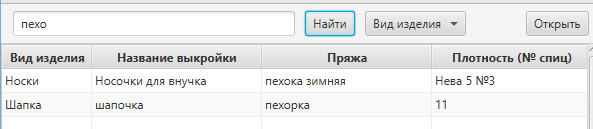


Рисунок 4.22 – Поиск по введенной строке

Можно отфильтровать по виду изделия: шапки, варежки, носки (рисунок 4.23). Выведутся все выкройки у которых вид совпадает с выбранным (рисунок 4.24).

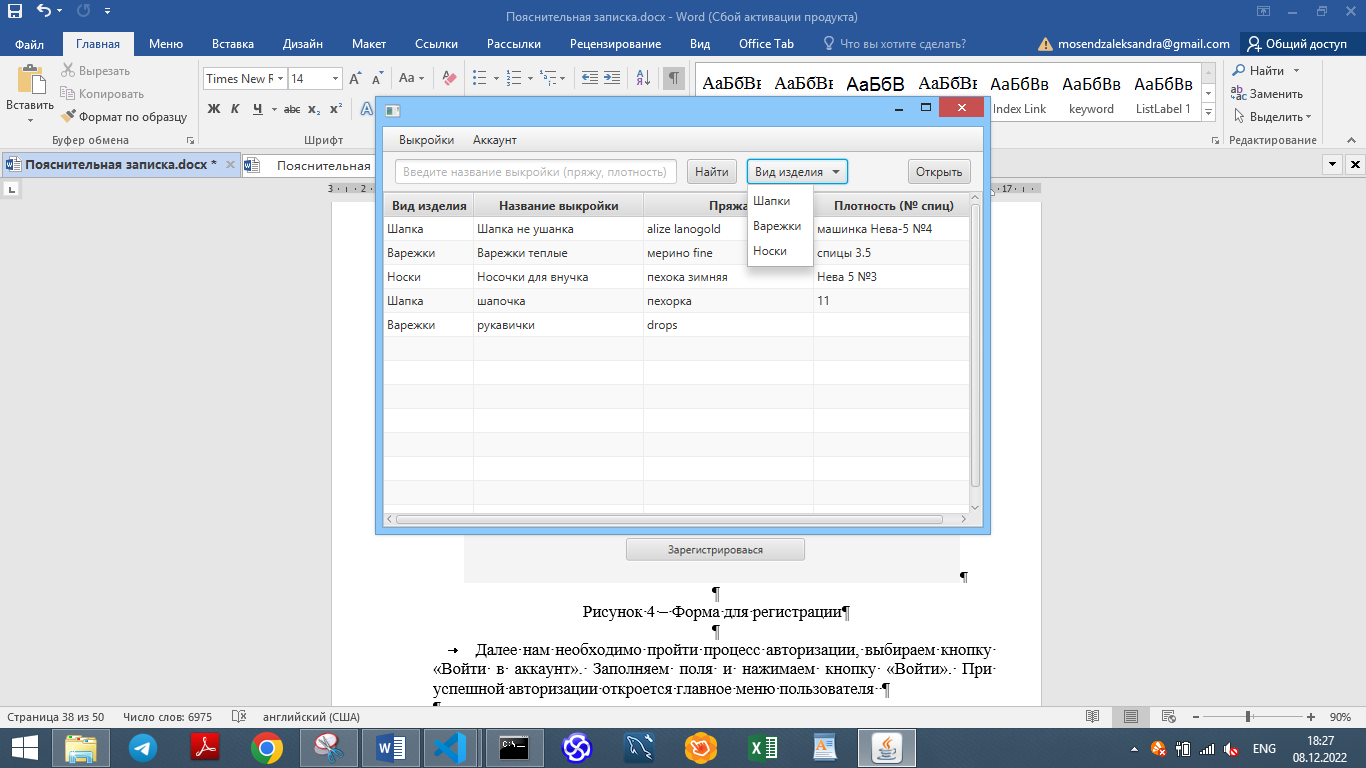


Рисунок 4.23 – Виды фильтрации

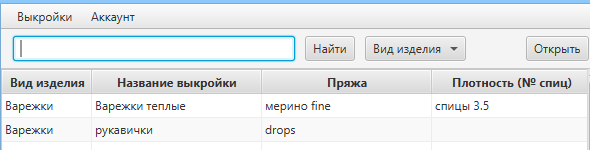


Рисунок 4.24 – Фильтрация по виду изделия

На любую выкройку можно нажать, она подсветиться синим, после чего нажать на кнопку открыть, и выкройка откроется все поля, которые вы ранее заполняли, будут заполнены (если захотите рассчитать нужно нажать кнопку «Рассчитать»)

Теперь перейдем в аккаунт пользователя нажав «Аккаунт»-> «Открыть». Форма заполнится нашими данными, не отображается только пароль (рисунок 4.25).

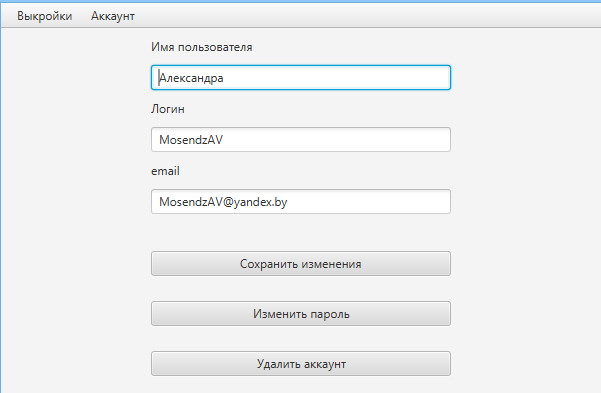


Рисунок 4.25 – Аккаунт пользователя

Если вы захотели что-то поменять, измените текст в полях и нажмите кнопку изменить. Если логин изменен, будет произведена проверка на существование данного логина, если логина не существует, данные сохранятся, иначе нужно изменить логин и попробовать снова (рисунок 4.26).

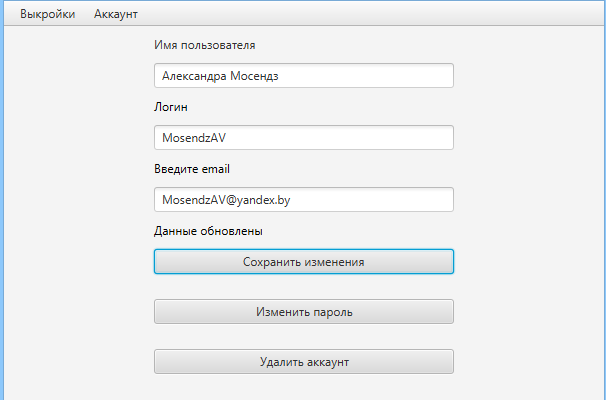


Рисунок 4.26 – Изменение данных пользователя

Чтобы изменить пароль, нужно нажать на кнопку «Изменить пароль», после чего появится форма для ввода старого и нового пароля. Если старый пароль верен данные обновятся (рисунок 4.27).

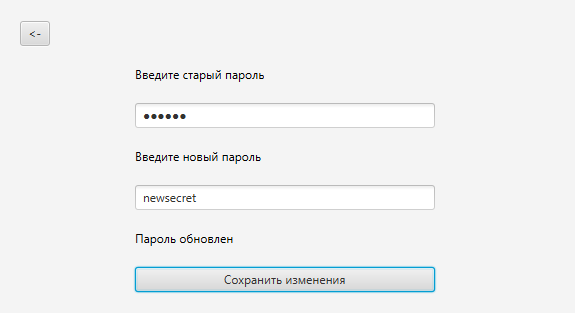


Рисунок 4.27 – Изменение пароля

Чтобы удалить аккаунт, нужно нажать на кнопку «Удалить аккаунт». После чего нужно подтвердить действие (рисунок 4.28).

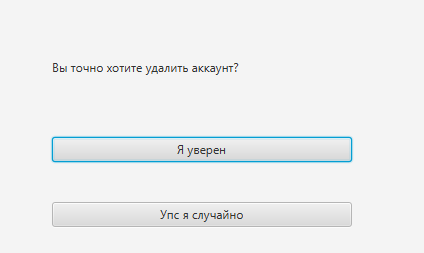


Рисунок 4.28 – Удаление аккаунта

**Действия пользователя**

Вначале нам необходимо пройти процесс авторизации, выбираем кнопку «Я админ». Вводим пароль и нажимаем кнопку «Войти» (рисунок 4.29). При успешной авторизации откроется главное меню администратора

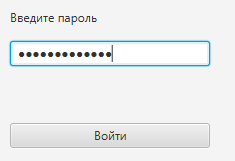


Рисунок 4.29 – Авторизация администратора

На главном меню можно увидеть все возможности администратора (рисунок 4.30). Он может работать с журналом запросов, с пользователями и изменить свой пароль.

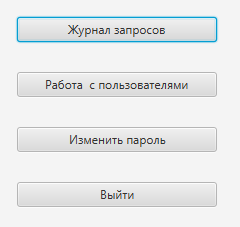


Рисунок 4.30 – Главное меню администратора

Далее во всех окнах (сверху) будет размешено меню, для удобной навигации по приложению (рисунок 4.31)

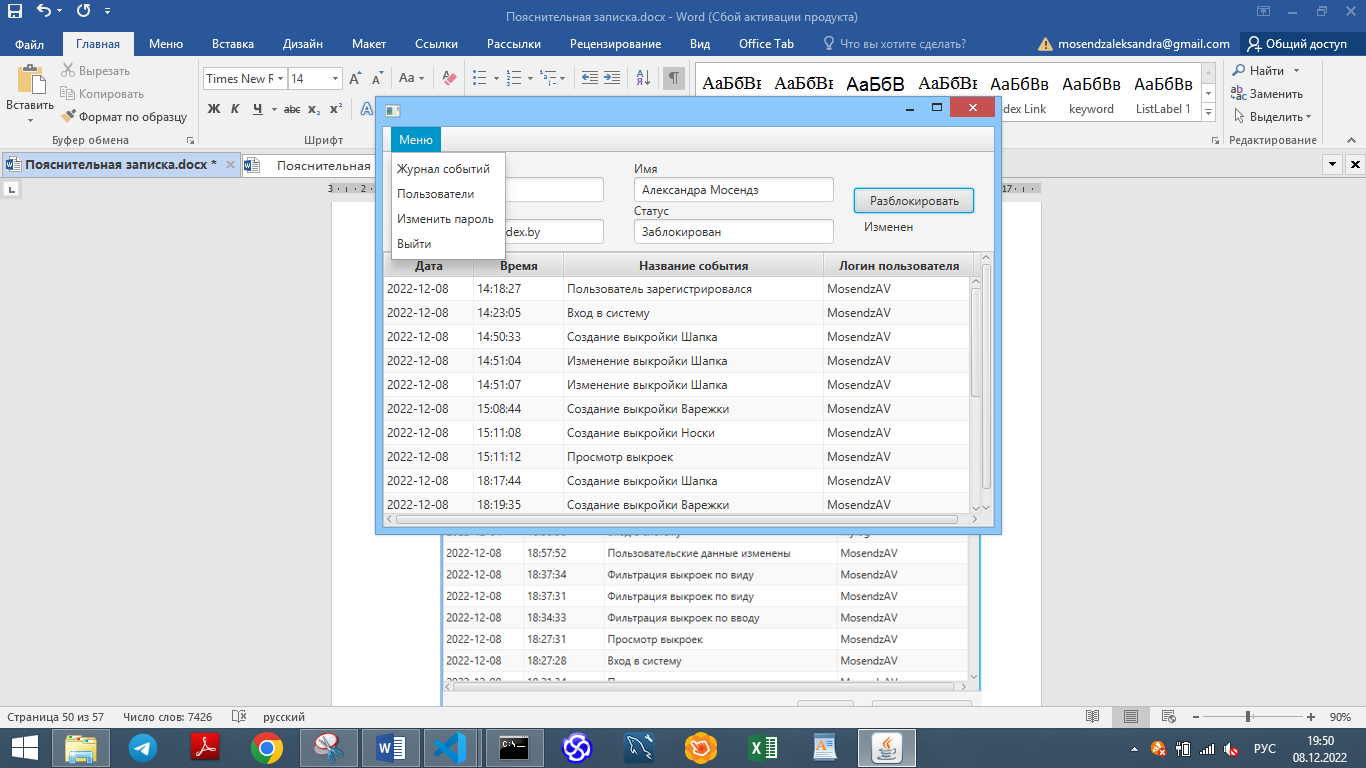


Рисунок 4.31 – Навигация по приложению

Откроем журнал запросов (рисунок 4.32), в нем хранятся все важные события в приложении. Каждое событие имеет название, дата и время совершения события и логин пользователя совершившего его.

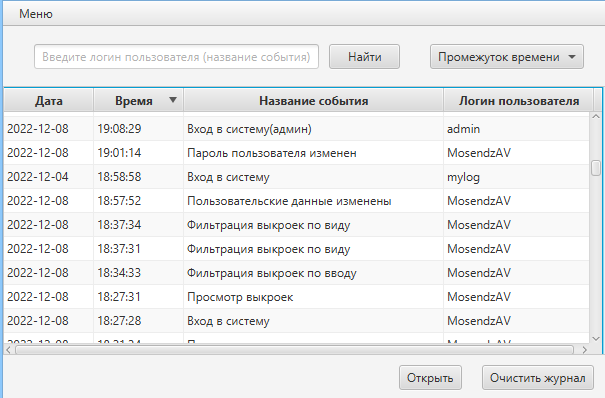


Рисунок 4.32 – Журнал запросов

Можно ввести строку в поле ввода и найдутся все действия, у которых название и/или логин совпадает полностью или частично (рисунок 4.33).

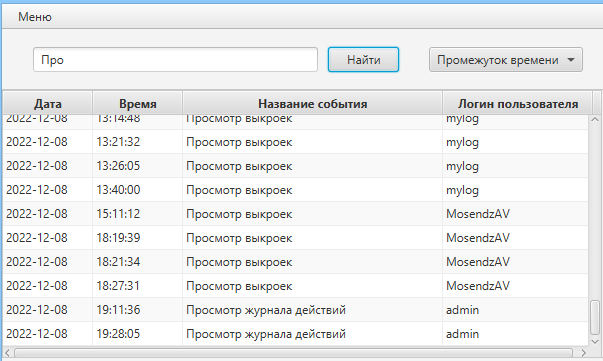


Рисунок 4.33 – Фильтрация данных по введенной строке

Можно отфильтровать данные по периоду: сегодня, за неделю, за месяц, за все время (рисунок 4.34) Например отфильтруем данные за сегодня, можно заметить что первая запись за сегодня сделана в 6 часов утра, до этого записи не отображаются (рисунок 4.35 )

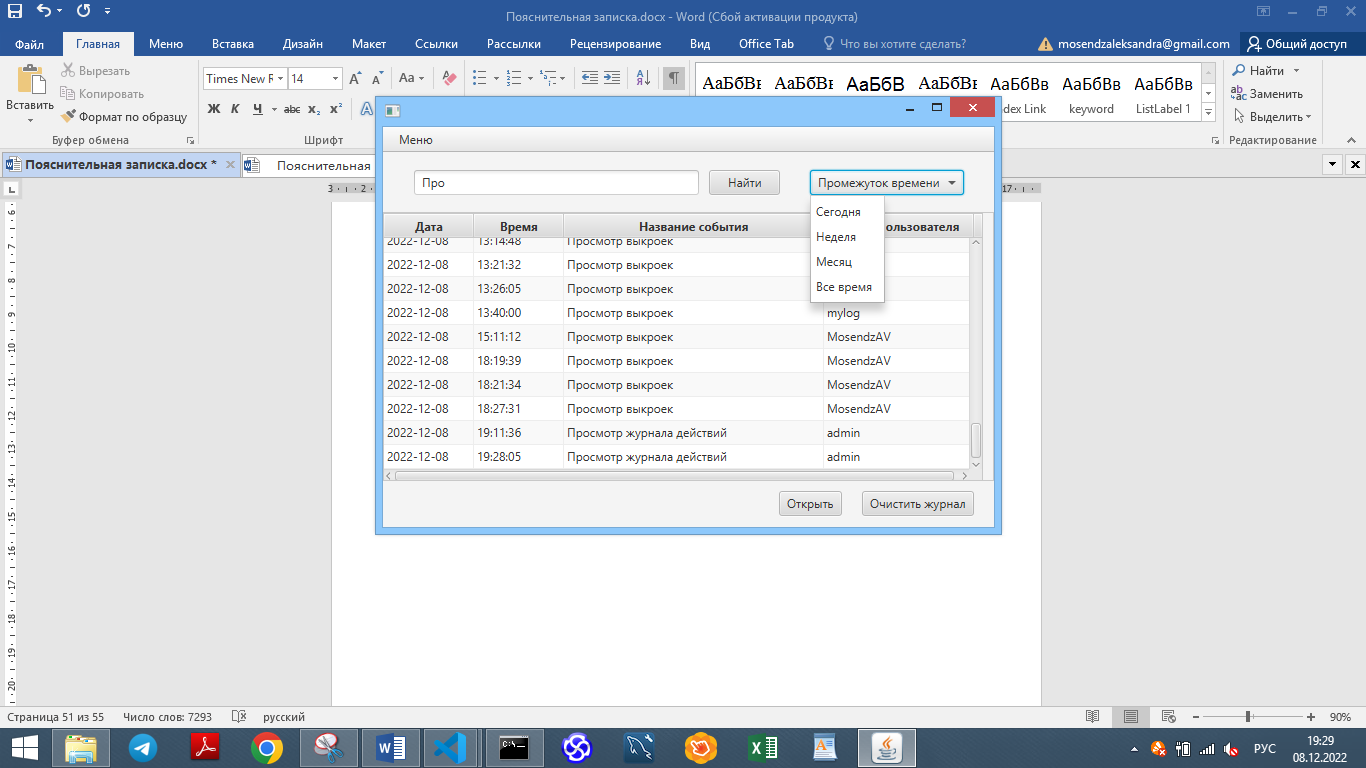


Рисунок 4.34 – Параметры фильтрации по периоду

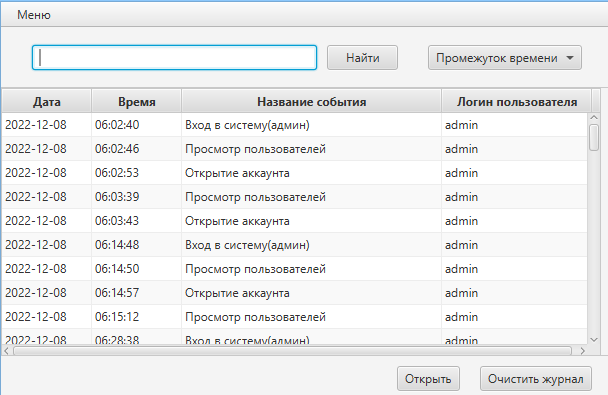


Рисунок 4.35 – Отфильтрованные данные за сегодня

Если нажать кнопку «Очистить», все данные из журнала будут удалены. Но я это сделаю, после демонстрации страницы пользователя, на которую можно перейти выбрав строку в таблице журнала и нажав кнопку «Открыть». На странице отображается информация о пользователе, кроме пароля, пароль пользователя администратору не доступен. В таблицу выводятся все действия пользователя (рисунок 4.36)

В данном окне можно изменить статус пользователя (рисунок 4.37), если статус «Заблокирован», пользователь не может войти в приложение.

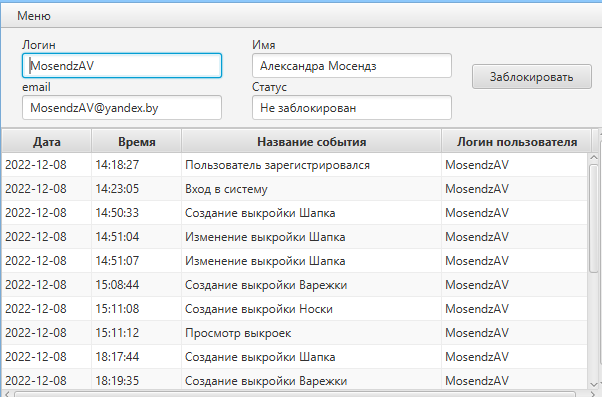


Рисунок 4.36 – Страница пользователя

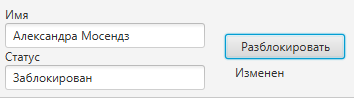


Рисунок 4.37 – Изменение статуса пользователя

Чтобы просмотреть всех пользователей, нужно выбрать «Меню»-> «Пользователи». В таблицу будут выведены все пользователи, их логин, имя, почта и статус. Аналогично поиску по журналу, можно искать пользователя по введенной строке (поиск осуществляется по логину, имен и почте). Можно отфильтровать пользователей по статусу (заблокирован или нет) (рисунок 4.38).

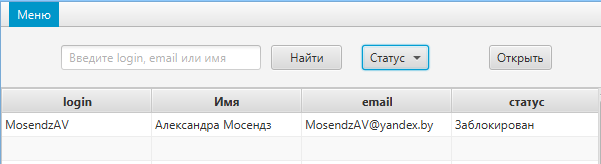


Рисунок 4.38 – Фильтрация пользователей по статусу

Аналогично журналу событий, можно нажать на любого пользователя, далее нажать кнопку «Открыть» и откроется страница пользователя.

Администратор также может изменить пароль, процесс происходит аналогично изменению пароля пользователя.

Таким образом, выполнены проектирование и разработка программного средства составления выкроек для вязания. 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. Береговых Ю.В. Алгоритм составления расписания занятий/ Ю.В. Береговых, Б.А. Васильев, Н.А. Володин, 2009. – 7 с.
2. Галузин К.С. разработка модуля для автоматизации составления оптимального учебного расписания в рамках единой информационной системы образовательного учреждения/ К.С. Галузин, В.Ю. Столбов. - Известия Белорусской инженерной академии, 2003.
3. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. – Горячая Линия - Телеком, 2007. – 452 с.
4. Гладков Л.А. Генетические алгоритмы: учебное пособие / Л.А. Гладков, В.В. Курейчик, В.М. Курейчик. – М. : Физматлит, 2004. – 407 с.
5. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест // МЦНМО. - Москва, 2000. – 960 с.

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

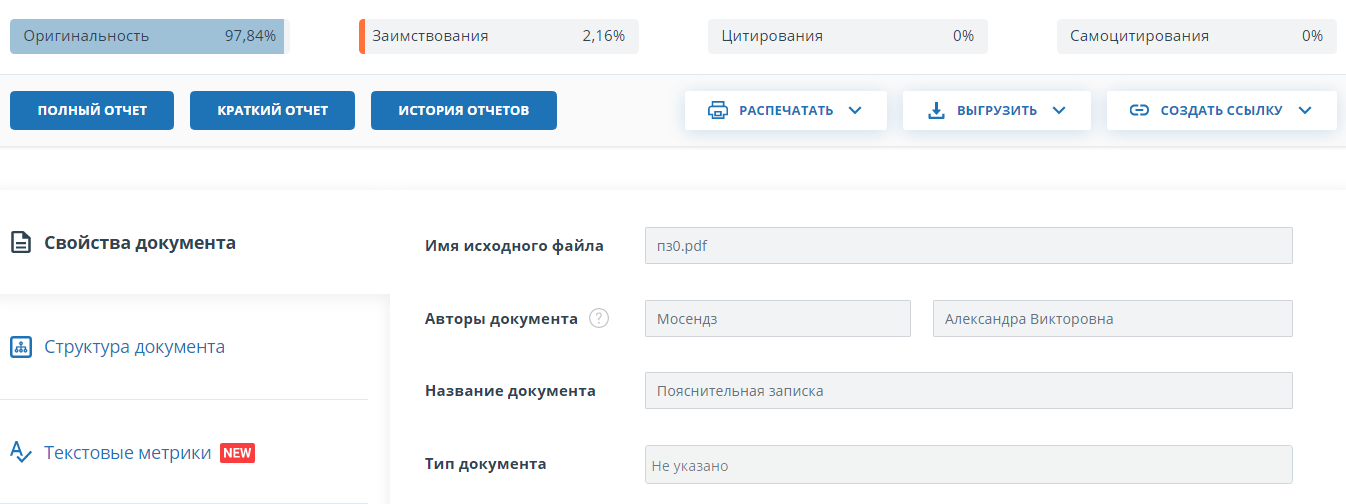


Рисунок А.1 – Отчет о проверке на заимствования в системе «Антиплагиат»

**Приложение Б**(обязательное)  
Листинг кода алгоритмов, реализующих 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);