МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГ**О»

Кафедра полимеров на базе ООО «АКРИПОЛ»

Интеграция программного обеспечения с целью оптимизации производственной структуры ООО «Акрипол»

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студента 4 курса 441 группы

Направления 20.03.01 «Техносферная безопасность»

Института химии

Шапкина Эдуарда Владимировича

Научный руководитель
к.х.н., доцент _____ И.В.Федусенко

Зав. кафедрой полимеров
на базе ООО «АКРИПОЛ»,
д.х.н., проф. _____ А. Б. Шиповская

Введение

Актуальность данной темы состоит в том, что независимо от отрасли, к которой принадлежит предприятие, вопрос о производственной структуре является одним из ключевых в системе управления. От правильно и четко разработанной структуры зависят результаты хозяйственной деятельности предприятия, а также эффективность всех протекающих процессов.

В настоящее время говоря об оптимизации производственной структуры в целом, экономия затрат всегда была одной из основных задач для любого бизнеса. Экономят, или лучше сказать, оптимизируют компании обычно все: от заработной платы сотрудников до мелких офисных расходов. Для того, чтобы избежать панического неэффективного сокращения затрат, необходимо создать целую систему управления ими, которая включает в себя в том числе и повышение координации деятельности различных структурных подразделений.

Цель работы: Создание программного обеспечения для предприятия ООО «АКРИПОЛ» с целью повышения производительности труда, за счёт оптимизации затрат на «ненужные» операции и производственные приёмы, лишние перемещения людей, а также с целью повышения информативности передаваемой информации из цехов, касающейся израсходованного сырья и коммунальных ресурсов, отделу управления предприятием. Для достижения данной цели нужно поэтапно решить ряд задач.

- Обсуждение функционала приложения.
- Выявление «не выполнимых» требований в рамках ВКР.
- Согласование сроков и конечного функционала приложения и формирования

на основе требований к функционалу приложения технологического задания.

- Систематизация требований и теоретическая разработка архитектуры приложения.
- Теоретическая разработка архитектуры базы данных.
- Подбор современных и удобных методов для реализации проекта.
- Разработка приложения для производственного объекта, исходя из требований заказчика.
- Тестирование приложения на предприятии.

Задачи проекта. Снижение психофизических нагрузок на персонал, при помощи ПО, которое оптимизирует расчёты, связанные с количественными характеристиками продуктов и коммунальных ресурсов, а также, с бухгалтерским расчётом стоимости затрат;

Повышение информативности отчётов, связанных с затраченым сырьём на производство продуктов.

Данная работа содержит: 9 глав:

Введение

Термины и определения

Реализация проекта

Реализация архитектуры проекта и технологий

Входной интерфейс программы

Бизнес логика и обработка результатов

Заключение

Список используемых источников

Приложение 1

Основное содержание работы

Реализация архитектуры проекта и технологий. Для реализации бизнес-логики использовали инструменты [1]-[5] и [52]-[58]. Бизнес-логику можно представить как работу с базой данных, заполнение которой происходит после передачи по TCP-соединению на сервер. Далее на сервере формируется база данных, которая состоит из объектов, внутри каждого из которых содержится своя база данных, реализованная при помощи Java Collection Framework [6] , где производятся необходимые арифметические вычисления с записью результата в базе данных объекта. Далее логика работы сводится к отображению результата арифметических вычислений в графическом интерфейсе (GUI) в профиле «Руководитель».

Далее вся бизнес-логика связана с теми результатами, которые ввёл пользователь. Проводится арифметическая обработка с записью результатов в базу данных с последующим отображением в профиле «Руководитель».

Для реализации GUI была выбрана технология JavaFX (или OpenJFX) [7], так как она обладает отличным графическим фреймворком SceneBuilder [11], который значительно ускоряет скорость разработки, позволяя разделять бизнес-логику работы графического интерфейса и разметки его элементов. Использовали инструменты [11],[12]-[51].

Для реализации стилизации графического интерфейса использовался формальный язык CSS [9], так как он хорошо интегрирован в JavaFX, что существенно ускоряет процесс кастомизации приложения.

В качестве системы контроля версий (VCS) использовался Mercurial [8], так как он имеет интеграцию в Comunity Edition (бесплатная версия IntelliJ_IDEA) [10].

В качестве интегрированной среды разработки программного обеспечения была выбрана среда IntelliJ_IDEA [10], так как в бесплатной версии (Comunity Edition) имеется ряд интеграций со всеми используемыми

(вышеперечисленными) технологиями в проекте, что позволяет многократно сократить время разработки.

Обработка результатов, введённых пользователем начинается с того, что мы высчитываем, то сколько по норме должно быть израсходовано того или иного ресурса, будь он коммунальным или сырьевым.

Под ресурсом в данном контексте следует понимать количество сырья (исчисление в тоннах, не считая ситуации, когда исчисляется антислёживающий агент, необходимый на этапе сушки, который исчисляется в г. и кг. преимущественно), электроэнергию (исчисление в кВт·ч.), воду (исчисление в $м^3$), азот (исчисление в $м^3$) и тепло (исчисление в гкал.).

Расчеты по расходованию ресурсов проводили по формуле (1):

$$Q_{продукта} \cdot Qr_{ресурса} = Q_{по\ норме} \quad (1)$$

где $Q_{продукта}$ (от англ. Quantity) - количество произведенного продукта;

$Qr_{ресурса}$ (от англ. Quantity rate) - расходная норма на соответствующий ресурс, который требуется расчитать;

$Q_{по\ норме}$ (от англ. Quantity) - величина затрат ресурса по норме.

Для коммунальных ресурсов, к которым в контексте данной программы отнесены азот, электроэнергия, вода и тепло применяется логика формулы (1). Далее результат вычисления форматируется классом BigDecimal [54] кодом:

1. valueDiv = valueDiv.replace(',', '.');// Замена запятой на точку введена для того, чтобы в случае если на вход метода от пользователя поступит запятая в числе, это не привело к выбрасыванию исключения java.lang.NumberFormatException и не было необходимости его обрабатывать.

2. return String.valueOf (new BigDecimal (Double. ParseDouble (**valueDiv**)). setScale (2, RoundingMode.CEILING). doubleValue());

где - valueDiv — значение для форматирования.

2.1. В случае, если расходные нормы на данный ресурс отсутствуют, в карточке продукта в GUI будет следующая запись: «" Нет Р.Н."»

На этом математическая обработка коммунальных ресурсов закончена. Далее идёт запись в ячейку внутреннего массива класса Product.

Для сырьевых ресурсов с количественными характеристиками также применяется логика формулы (1), однако, присутствует ещё и вычисление отклонения от нормы, расчёт которого проводится по формуле (2):

$Div_{сырья} = (((Q_{изр.} \cdot 100) / Q_{по\ норме}) - 100)$; (2)

где $Div_{сырья}$ - отклонение реально израсходованого сырья от того количества, которое должно быть израсходовано по норме;

$Q_{изр.}$ - количество реально израсходованого сырья;

$Q_{по\ норме}$ - количество сырья, израсходованного по норме.

В случае если при производстве продукта используется полупродукт, который произведён на данном предприятии (ООО «Акрипол»), то можно вычислить количество сырья, затраченного на производство полупродукта, исходя из отклонения, которое было допущено при его производстве.

Расчёт проводится по формуле (3)

$Q_{по\ норме\ от\ отклонения} = (((((Q_{синтез} \cdot 100) / (Qr_{сырья} \cdot Q_{продукта}))-100)+100)/100) \cdot Q_{полупродукта} \cdot Qr_{сырья}$; (3)

где $Q_{по\ норме\ от\ отклонения}$ – количество затраченного ресурса на синтез полупродукта, исходя из его отклонения, которое было допущено при его синтезе;

$Q_{синтез}$ – количество реально затраченного сырья на производство всего количества полупродукта;

$Qr_{сырья}$ – расходная норма на сырьё, в контексте которого ведётся расчёт;

$Q_{продукта}$ — количество продукта, полученного всего;

$Q_{полупродукта}$ — количество синтезированного полупродукта.

В качестве примера приведены некоторые расчеты.

Пример 1.

Приведен пример расчёта коммунального ресурса воды для «Акриламид

40%» по расходным нормам.

Исходные данные:

1. Произведено «Акриламид 40%» 20тонн;

2. Норма на расход воды для производства «Акриламид 40%» составляет 1.61;

Для расчета количества воды, затрачиваемой на синтез 20 тонн «Акриламид 40%» используем формулу (1). Получаем:

$20 \cdot 1.61 = 32.2$ (куб.м.).

Просмотреть информацию о каждом продукте, а также об отклонениях, которые были допущены при производстве, пользователь может в окне «Информация о продукте», которое вызывается кнопкой «Показать информацию о продукте», которая находится в профиле «Руководитель». Также, был внедрён функционал, позволяющий подсчитать все коммунальные ресурсы, которые затрачивались на производства и пост-обработку полупродуктов (гидрогелей). Это было реализовано посредством внедрения кнопки «Показать общие коммунальные расходы» в профиль «Руководитель»

Поля ввода текста поддерживают форматируемый ввод, т.е. полям присваивается шаблон ввода (по шаблону: мы не можем ввести никакой символ кроме числа или числа содержащего одну точку): «**^[0-9]*\\.{0,1}[0-9]***».

Это сделано для того, чтобы избежать ввода пользователем букв, иных символов (или большего количества символов (например ввод 2 и более точек, вместо одной точки для разделения десятичного разряда)) и избежать исключений и ошибок которые требуют дополнительной обработки и могут привести к непредсказуемому поведению программы в случае если вследствие наличия у разработчика человеческого фактора где-нибудь пропустить обработку исключений.

Заключение

Для решения поставленных задач были использованы современные методы проектирования и реализации данных задач:

1. Методы сбора и обработки данных — прямое общение с заказчиком (и сотрудниками данной организации, которые представляли заказчика на отдельных ступенях реализации проекта), описание, обобщение, шаблонизация.

2. Методы реализации проекта — работа с современными методами и использования всей мощи языка (ООП, framework's, IDE, библиотек (libraries)).

Проведенное исследование показало, что возможно продолжение работы в направлении расширения функционала приложения с целью повышения степени оптимизации всего производства и визуальной информативности предоставления информации. Например, сбор данных о состоянии средств индивидуальной защиты в каждом цеху (срок годности, готовности к применению и др.) поможет систематизировать данные о средств индивидуальной защиты на текущий момент и в случае необходимости позволит принимать те или иные решения, основываясь на достоверных данных. Следует рассмотреть возможность подключения дополнительного функционала, позволяющего хранить данные в течение длительных промежутков времени. Это даст возможность проводить анализ этих данных, отображая в графическом интерфейсе результаты анализа в любых выбранных интервалах времени в виде разных способов представления данных (таблицы, диаграммы, гистограммы и пр.)

ВЫВОДЫ:

1. На предприятии ООО «Акрипол» при текущем функционале внедрен программный продукт, позволяющий выявить отклонение от расходных норм, а также провести расчет затрат сырья и коммунальных ресурсов по расходным нормам.

2. Разработан программный продукт для предприятия ООО «Акрипол», позволяющий вычислять расход сырья по расходным нормам и отклонение от расходных норм при производстве полиакриламидных реагентов.

3. Дополнен набор программного обеспечения ООО «Акрипол» принципиально новым программным продуктом, позволяющим снизить психофизическую нагрузку на персонал, посредством возложения однотипных операций вычисления на компьютер, а также вычислять отклонение от расходных норм.

4. Сформулированы рекомендации по расширению функционала приложения с целью повышения степени оптимизации всего производства и визуальной информативности предоставления информации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html // Класс Object
2. https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html // Интерфейс Collection
3. https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html // Класс Arrays
4. https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html // Класс HashMap <K,V>
5. https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html // Класс ArrayList<E>

6. https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html
7. https://docs.oracle.com/javafx/2/
8. https://www.mercurial-scm.org/
9. https://www.w3.org/Style/CSS/
10. https://www.jetbrains.com/idea/
11. https://gluonhq.com/products/scene-builder/
12. https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html // Класс Properties.html
13. https://docs.oracle.com/javase/8/javafx/api/javafx/collections/ObservableList.html // Класс ObservableList
14. https://docs.oracle.com/javase/8/javafx/api/javafx/collections/FXCollections.html // Класс FXCollections
15. https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html // Класс CopyOnWriteArrayList (пакет inet)
16. https://docs.oracle.com/javase/10/docs/api/javafx/collections/transformation/SortedList.html // Класс SortedList
17. https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html // Класс Stage
18. https://docs.oracle.com/javafx/2/api/javafx/stage/Modality.html // Класс Modality – характеристика Stage, модальность окна, хранится в классе в виде enum представления
19. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Scene.html //Класс Scene
20. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Parent.html // Класс Parent
21. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Group.html // Класс Group
22. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/Pane.html // Класс

Pane

23. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ScrollPane.html // Класс ScrollPane

24. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/StackPane.html // Класс StackPane

25. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/GridPane.html // Класс GridPane

26. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/RowConstraints.html // Класс RowConstraints

27. https://docs.oracle.com/javafx/2/api/javafx/scene/layout/ColumnConstraints.html // Класс ColumnConstraints

28. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Node.html // Класс Node

29. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Button.html // Класс Button

30. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Label.html // Класс Label

31. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ChoiceBox.html // Класс ChoiceBox

32. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TextField.html // Класс TextField

33. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/PasswordField.html // Класс PasswordField

34. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/RadialGradient.html // Класс RadialGradient

35. https://docs.oracle.com/cd/E17802_01/javafx/javafx/1/docs/api/javafx.scene.paint/javafx.scene.paint.Color.html // Класс Color

36. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html // Класс Image

37. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/image/ImageView.html // Класс ImageView
38. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/effect/Blend.html // Класс Blend
39. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Line.html // Класс Line
40. https://docs.oracle.com/javafx/2/api/javafx/scene/text/Font.html // Класс шрифтов — Font
41. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Rectangle.html // Класс Rectangle – прямоугольник
42. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/MenuBar.html // Класс MenuBar
43. https://docs.oracle.com/javafx/2/api/javafx/scene/control/Menu.html // Класс Menu
44. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/MenuItem.html // Класс MenuItem
45. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TableView.html // Класс TableView – класс табличного представления информации
46. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TableColumn.html // Класс TableColumn
47. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TableRow.html // Класс TableRow
48. https://docs.oracle.com/javafx/2/api/javafx/scene/control/cell/PropertyValueFactory.html Класс PropertyValueFactory
49. https://docs.oracle.com/javase/8/javafx/api/javafx/event/Event.html // Класс Event – события, применено в контексте переопределения для создания собственного события в приложении, для принятия данных и создания на основе них табличного представления

50. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/input/MouseEvent.html // Класс MouseEvent
51. https://docs.oracle.com/javase/8/javafx/api/javafx/event/ActionEvent.html // Класс ActionEvent
52. https://docs.oracle.com/javase/7/docs/api/java/lang/Number.html // Класс Number
53. https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html // Класс Math используется для математической обработки, алгебраических вычислений, результатов вычисления, а так-же на промежуточных этапах вычисления для получения результатов округления до необходимого знака после запятой, с целью сохранения точности вычисления.
54. https://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html // Класс BigDecimal
55. https://docs.oracle.com/javase/7/docs/api/java/math/RoundingMode.html // Класс RoundingMode
56. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/DecimalFormat.html // Класс DecimalFormat - позволяет «обрезать» конечный результат вычисления, задавая шаблон отображения, с целью сохранения точности вычисления
57. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Double.html // Класс Double
58. https://docs.oracle.com/javase/9/docs/api/java/lang/String.html // Класс String