**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Объектно-ориентированное программирование»

на тему: **«Расчет потребности в сырье и материалах на плановый объем продукции»**

Исполнитель: студент гр. ИП-22

Коваленко А.И.

Руководитель: преподаватель кафедры «Информатика»

В.Н. Шибеко

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2022

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc103027589)

[1 Аналитический обзор 4](#_Toc103027590)

[1.1 Описание объекта 4](#_Toc103027591)

[1.2 Обзор существующих методов решения 4](#_Toc103027592)

[1.3 Используемые средства 5](#_Toc103027593)

[1.4 Постановка задачи 6](#_Toc103027594)

[2 Архитектура программного обеспечения 7](#_Toc103027595)

[2.1 Прецеденты и актеры 7](#_Toc103027596)

[2.2 Преценденты и их описание 10](#_Toc103027597)

[2.3 Описание классов и предметной области 10](#_Toc103027598)

[2.4 Отношения между классами предметной области 11](#_Toc103027599)

[3 Структура и основные алгоритмы созданного программного обеспечения 15](#_Toc103027600)

[3.1 Описание классов реализации пользовательских функций. 15](#_Toc103027601)

[3.2 Описание интерфейсов. 22](#_Toc103027602)

4 [Тестирование 33](#_Toc103027603)

[Заключение 37](#_Toc103027604)

[Список используемых источников 38](#_Toc103027605)

[Приложение А 39](#_Toc103027606)

# **ВВЕДЕНИЕ**

С развитием информационных технологий компьютеры с их расширенными функциональными возможностями активно применяются в различных сферах человеческой деятельности, связанных с обработкой информации, предоставлением данных.

В современном обществе, которое функционирует в рыночных условиях, своевременная обработка информации способствует совершенствованию организации производства, оперативному и досрочному планированию, прогнозированию и анализу хозяйственной деятельности, что позволяет успешно конкурировать на рынке. Каждая организация стремится минимизировать затраты времени, материальных, трудовых ресурсов в ходе своей деятельности и упростить процесс обработки информации. Эти задачи можно решить с использованием автоматизированных информационных систем.

Необходимость автоматизации рабочего процесса возникает, в первую очередь, у тех фирм, которые имеют достаточно солидные обороты и объёмы продаж. Таким компаниям нужна оперативность и управляемость, а они, напротив, сталкиваются с ростом разного рода рутинной деятельности. Требуется, насколько возможно, избавить своих сотрудников от нее и более эффективно их использовать [3].

Целью данной курсовой работы является разработка автоматизированной информационной системы по расчету потребности в сырье и материалах на плановый объем продукции.

# **1 АНАЛИЧЕСКИЙ ОБЗОР**

# **1.1 Описание объекта**

В данной работе главным объектом выступает предприятие по производству продукции. Будет произведена полная обработка данных. Методы оптимизации и автоматизации позволят ускорить решение типовых задач, автоматизировать составление отчетности, спланировать ведение проекта и высвободить драгоценные ресурсы.

Итогом проектирования модели будет приложение под управлением операционной системы *Windows* с подключенным источником данных, а именно базой данных, в которой будет хранится вся необходимая информация.

На основе накопленных данных будет производиться автоматическое формирование статистики, включающих построение графических диаграмм по деятельности предприятия.

# **1.2 Обзор существующих методов решения**

В качестве предметной области автоматизации рассматривается автоматизация расчет потребности в сырье и материала. Данный проект удобен для обработки данных по оказанию услуг в отношении поставщик-покупатель.

Автоматизированная система управления (АСУ) – комплекс аппаратных и программных средств, а также персонала, предназначенный для управления различными процессами в рамках технологического процесса, производства, предприятия.

Современные автоматизированные система улучшает качество обслуживания клиентской базы, снизить напряженность и трудоемкость труда персонала, повышает производительность работы всего персонала, а также уменьшить количество ошибок в его действиях.

Основная функция АСУ – обеспечение руководства информацией.

Важнейшая задача АСУ – повышение эффективности управления объектом на основе роста производительности труда и совершенствования методов планирования процесса управления.

В состав АСУ входят:

* информационные ресурсы, хранящих данные об объектах, связь между которыми задается определенными правилами;
* формальная логико-математическая система, реализованная в виде программных модулей, обеспечивающих ввод, обработку, поиск и вывод необходимой информации;
* интерфейс, обеспечивающий общение пользователя с системой в удобной для него форме, и позволяющий работать с информацией баз данных.

# **1.3 Используемые средства**

Разработка данного программного обеспечения выполнялось при помощи языка *С#.*

*C#* является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и самоописательных пакетов, реализующих отдельные функциональные возможности. Важная особенность таких компонентов – это модель программирования на основе свойств, методов и событий. Каждый компонент имеет атрибуты, предоставляющие декларативные сведения о компоненте, а также встроенные элементы документации. *C#* предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому *C#* отлично подходит для создания и применения программных компонентов [1].

Для разработки графического интерфейса использовалась графическая подсистема, входящая в состав *.NET Framework*, использующая язык *XAML*: *WPF (Windows Presentation Foundation).* Это система следующего поколения для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем.

В основе *WPF* лежит векторная система отрисовки, не зависящая от разрешения и созданная с расчетом на возможности современного графического оборудования. WPF расширяет базовую систему полным набором функций разработки приложений, в том числе язык *XAML* *(Extensible Application Markup Language)*, элементами управления, привязкой данных, макетом, двухмерный- и трехмерный - графикой, анимацией, стилями, шаблонами, документами, мультимедиа, текстом и оформлением. *WPF* входит в состав *Microsoft .NET* *Framework* и позволяет создавать приложения, включающие другие элементы библиотеки классов *.NET Framework*.

Для хранения данных используется база данных *MS SQL Server 2022*. Для удобного управления базой данных было использовано специальное средство администрирования *SQL Server Management Studio*. Для реализации подхода была использована технология *Entity Framework.*

Наиболее качественной средой разработки для языка C# на операционной системе *Windows* является *Visual Studio* – официальный продукт от компании *Microsoft* – разработчика платформы *.NET. Разработка* этого проекта велась в среде *Visual Studio 2019 Community*.

# **1.4 Постановка задачи**

Необходимо разработать программу под платформу *Windows*, которая содержит в себе следующий функционал:

* содержание информации о пользователях;
* осуществление учета производства;
* содержание информации и материалах;
* содержание информации и продукции;
* содержание информации о сырье;
* учет плана производства на заданный период;
* учет нормы производства на заданный период,

На основании произведённого анализа существующих методов, можно сформировать список требований, для разрабатываемой системы:

* приложение не должно иметь лишнего функционала;
* доступ к приложению должен быть удобным и простым;

Программный продукт должен предоставлять:

* справочники, включая сырьё, материалы и продукцию;
* получение статистики в графическом виде, также необходимо разделение доступа к функционалу приложения по ролям: Гость, Администратор, Работник, Клиент.

# **2 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

# **2.1 Прецеденты и актеры**

Предметная область должна описывать все объекты, которые являются частью предметной области, а также любые взаимоотношения между ними. Следовательно, необходимо описать все зависимости между сущностями. Важно знать какие именно объекты попадают в предметную область и какие связи между ними существуют и необходимо помнить цель проектирования данного приложения [4].

Для формирования представления о предметной области используют *UML* диаграммы, которые представлены на рисунках 1, 2, 3, 4.

Актер – множество логически связанных ролей в UML, исполняемых при взаимодействии с прецедентами или сущностями (система, подсистема или класс). Актером может быть человек или другая система, подсистема или класс, которые представляют нечто вне сущности.

Любые (в том числе и программные) системы проектируются с учётом того, что в процессе своей работы они будут использоваться людьми и/или взаимодействовать с другими системами. Сущности, с которыми взаимодействует система в процессе своей работы, называются актерами, причём каждый актер ожидает, что система будет вести себя строго определённым, предсказуемым образом.

Графически актер изображается символом класса с соответствующим стереотипом. Эта форма представления имеет один и тот же смысл и могут использоваться в диаграммах. «Стереотипированная» форма чаще применяется для представления системных актеров или в случаях, когда актер имеет свойства и их нужно отобразить.

Прецеденты представляют действия, выполняемые системой в интересах актеров. Проще говоря, прецедент – это описание последовательности действий (или нескольких последовательностей), которые выполняются системой и производят для отдельного актера видимый результат. Один актер может использовать несколько элементов прецедентов, и наоборот, один прецедент может иметь несколько актеров, использующих его. Каждый прецедент задает определенный путь использования системы. Набор всех прецедентов определяет полные функциональные возможности системы.

На рисунке 1 изображена диаграмма, отображающая возможные ветки управления данным приложением на уровень доступа «Гость»

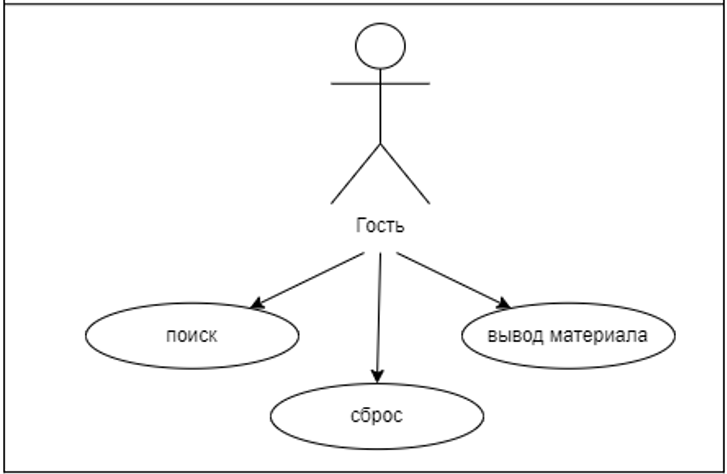


Рисунок 1 – Диаграмма прецедентов роли «Гость»

На рисунке 2 изображена диаграмма, отображающая возможные ветки управления данным приложением на уровень доступа «Клиент»

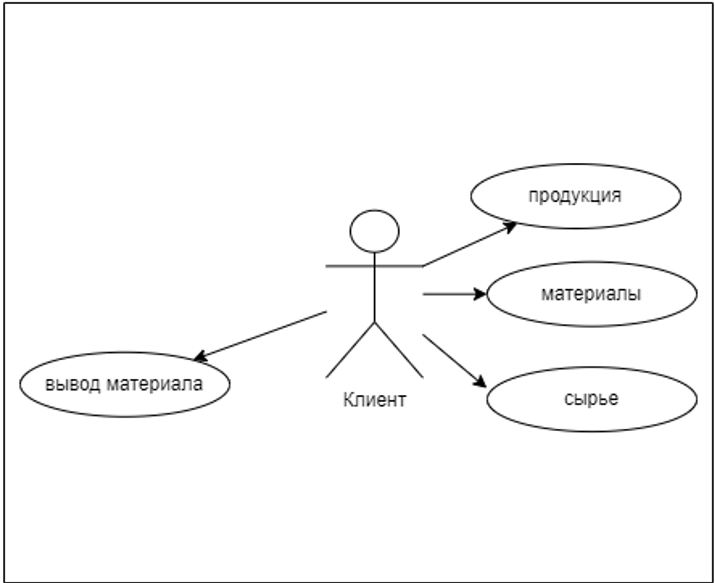


Рисунок 2 – Диаграмма прецедентов роли «Клиент»

На рисунке 3 изображена диаграмма, отображающая возможные ветки управления данным приложением на уровень доступа «Менеджер»

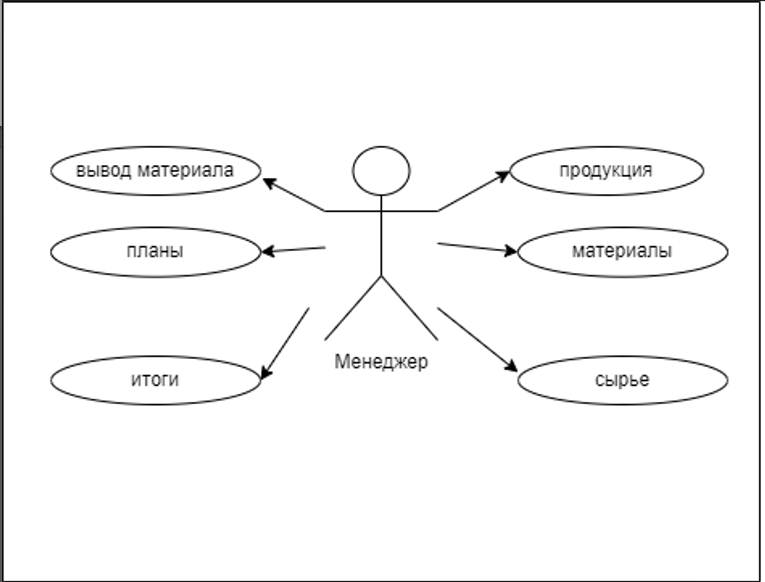


Рисунок 3 – Диаграмма прецедентов роли «Менеджер»

На рисунке 4 изображена диаграмма, отображающая возможные ветки управления данным приложением на уровень доступа «Администратор»

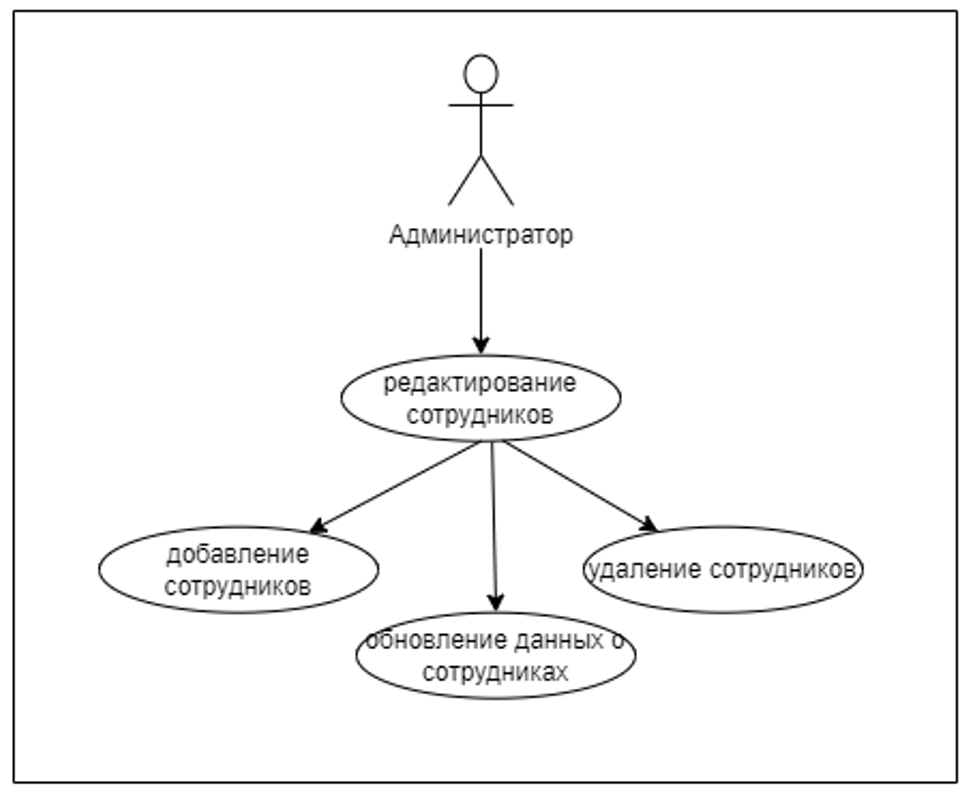


Рисунок 4 – Диаграмма прецедентов роли «Администратор»

# **2.2 Преценденты и их описание**

«Гость» – пользователь, которому не доступны большинство функции. Единственное что он может – это просмотр справочника товаров, о нас, контакты и отзывы наших клиентов.

«Клиент» – это пользователь, задачей которого является оформление заказа. Для удобства просмотра истории заказов данному уровню доступа доступен личный кабинет, где он может посмотреть подробности заказа.

«Менеджер» – это работник, предназначенный для согласования заказа с клиентом, то есть основная задача которого является отказ или принятие заказа.

«Каталог товаров» метод предназначен для ознакомления с материалами, располагающими у фирмы.

«Отзывы» – метод, предназначенный для просмотра отзывов, оставленных клиентами.

«Контакты» – это метод, предназначенный для просмотра контактов для связи с «нами».

«О нас» – это метод, предназначенный для какой-либо справочной информации о компании.

«Сделать заказ» служит для оформления заказа клиентом.

«Изменение статуса заказа» предназначен для предоставления менеджеру возможности изменения состояния заказа: отказать в услуге или принять.

«Статистика» – это окно, предназначенное для отображения статистики, генерируемой автоматически по следующий полям:

* количество заказов за всё время работы;
* количество клиентов за всё время работы;
* количество сотрудников на данный момент времени;
* количество материалов, имеющихся на складе на данный момент

времени;

* средняя стоимость всех имеющихся материалов.

Прецедент «Личный кабинет» служит для просмотра информации о заказах, которые выполняет сотрудник с данным ID.

# **2.3 Описание классов и предметной области**

Реляционная модель – совокупность данных, состоящая из набора двумерных таблиц. В теории множеств таблице соответствует термин отношение (*relation*), физическим представлением которого является таблица, отсюда и название модели – реляционная.

Реляционная модель является удобной и наиболее привычной формой представления данных, так в настоящее время эта модель является фактическим стандартом, на который ориентируются практически все современные СУБД.

На реляционной модели данных строятся реляционные базы данных. Модель данных на рисунке 5 представляет набор таблиц, связанных между собой.

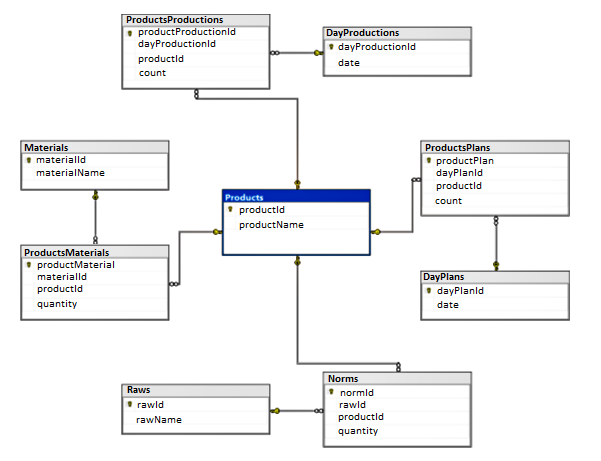


Рисунок 5 – Реляционная модель данных

# **2.4 Отношения между классами предметной области**

Рассмотрим подробно назначения таблиц реляционной модели данных, показанных на рисунке 5.

Таблица «*Materials*» предназначена для хранения дополнительной информации о таблице «*ProductsMaterials*».

Таблица «*Products*» предназначена для хранения в себе всех видов продуктов, имеющихся на складе магазина.

Таблица «*Raws*» имеет схожее назначение с таблицей *Materials* т.к. они оба являются справочниками нашей информационной системы.

Таблица «*Norms*» предназначена для хранения уникального идентификатора определённого вида продукта с дальнейшей обработкой данных

Таблица «*DayPlans*» предназначена для хранения дополнительной информации о таблице *ProductsPlans*.

Таблица «*ProductsPlans*» предназначена для хранения даты запланированного события, продукта со склада, а также количество продукта.

Таблица «*DayProductions*» предназначена для хранения дополнительной информации о таблице *ProductsProductions*.

Таблица «*ProductsProductions*» предназначена для хранений уникальных данных о продуктах с указаний количества.

Между двумя сущностями может быть установлена связь. Отношения между сущностями характеризуются глаголом, который можно применить для взаимодействия между ними. Связь – это некое отношение между двумя типами сущностей.

Между типами сущностей различают следующих 3 типа связей:

* «один-к-одному» –это значит, что одному экземпляру некоторой

сущности может соответствовать только один экземпляр другой сущности;

* «один-ко-многим» – это значит, что одному экземпляру сущности

может соответствовать любое количество экземпляров другой сущности. Если известно значение максимального количества экземпляров, то непосредственно указывается само значение;

* «многие-ко-многим» –это означает, что нескольким экземплярам

одной сущности может соответствовать несколько экземпляров другой сущности.

Между всеми таблицами было установлено обеспечение целостности данных. Целью обеспечения целостности данных является предотвращение появления непарных записей, ссылающихся на несуществующие записи. В результате *SQL* отменяет для этого отношения все действия, которые могут нарушить целостность данных.

Также может потребоваться удаление строки и всех связанных с ней записей – например, записи в таблице «*Clients*» и всех связанных с этим клиентом заказов. Для этого был установлен параметр «Каскадное удаление связанных записей». При удалении записи, содержащей первичный ключ, будут автоматически удалены все записи, связанные с этим первичным ключом. Также стоит заметить, если каскадное удаление не разрешено, невозможно удалить запись в главной таблице, если имеются связанные с ней записи в подчиненной.

Таблица 2.1 – Связи между сущностями

|  |  |  |
| --- | --- | --- |
| Имя таблицы | Тип связи | Имя таблицы |
| *Products* | Один ко многим | *Norms* |
| *ProductsMateruals* |
| *ProductsPlans* |
| *ProductsProductions* |
| *ProductsMaterials* |
| *Materials* | Один ко многим | *ProductsMateruals* |
| *Raws* | Один ко многим | *Norms* |
| *DayPlans* | Один ко многим | *ProductsPlans* |
| *DayProductions* | Один ко многим | *ProductsProductions* |
| *Materials* | Один ко многим | *ProductsMaterials* |

Данная схема отражает модель базы данных, на которой мы можем заметить, что у нас имеется 4 коллекции: *Materials, Products, Raws, Users.*

Коллекция «*Materials*» предназначена для наименования материала. Информация о полях представлена в таблице 2.2.

Таблица 2.2 – Информация о полях коллекции *Materials*

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение |
| *id* | *ObjectId* | Хранение уникального идентификатора |
| *Name* | *string* | Хранение уникального имени материала |

Коллекция «*Products*» предназначена для хранения данных о продукции. Информация о полях представлена в таблице 2.3.

Таблица 2.3 – Информация о полях коллекции *Products*

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение |
| *id* | *ObjectId* | Хранение уникального идентификатора |
| *Name* | *string* | Хранение уникального имени продукта |

Коллекция «*Raws*» предназначена для хранения данных сырье. Информация о полях представлена в таблице 2.4.

Таблица 2.4 – Информация о полях коллекции *Raws*

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение |
| *id* | *ObjectId* | Хранение уникального идентификатора |
| *Name* | *string* | Хранение уникального имени сырья |

Коллекция «*Users*» предназначена для хранения уникальной информации присущей каждому пользователю информационной системы, такой как логин, пароль и уровень доступа в систему. Информация о полях представлена в таблице 2.5.

Таблица 2.5 – Информация о полях коллекции *Users*

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип атрибута | Назначение |
| *id* | *ObjectId* | Хранение уникального номера. |
| *Level* | *String* | Уровень доступа в систему |
| *Login* | *String* | Хранение логина пользователя |
| *Password* | *String* | Хранение пароля пользователя |

# **3 СТРУКТУРА И ОСНОВНЫЕ АЛГОРИТМЫ СОЗДАННОГО ПРОГРАМНОГО ОБЕСПЕЧЕНИЯ**

# **3.1 Описание классов реализации пользовательских функций.**

Описание ООП-интерфейса, если отвлечься от деталей синтаксиса конкретных языков, состоит из двух частей: имени и методов интерфейса.

Имя интерфейсa строится по тем же правилам, что и другие идентификаторы используемого языка программирования. Разные языки и среды разработки имеют различные соглашения по оформлению кода, в соответствии с которыми имена интерфейсов могут формироваться по некоторым правилам, которые помогают отличать имя интерфейса от имён других элементов программы.

Методы интерфейса. В описании интерфейса определяются имена и сигнатуры входящих в него методов, то есть процедур или функций класса.

При непосредственной интеграции базы данных в приложение понятие «сущность» заменяется на «доменный класс».

Доменные классы – это классы в объектно-ориентированных компьютерных программах, выражающие сущности из модели предметной области, относящейся к программе, и реализующие бизнес-логику программы.

Задачи доменных классов представлена в таблице 3.1.

Таблица 3.1 – Назначения доменных классов

|  |  |
| --- | --- |
| Исходное название доменного класса | Назначение доменного класса |
| 1 | 2 |
| *ProductsProductions* | Хранение данных о продукции |
| *DayProductions* | Хранение дополнительных данных о продукции |
| *ProductsMaterials* | Хранение данных о материалах |
| *Materials* | Хранение дополнительных данных о материалах |

Продолжение таблицы 3.1 – Назначение доменных классов

|  |  |
| --- | --- |
| *1* | 2 |
| *Raws* | Хранение дополнительных данных о сырье |
| *Norms* | Хранение данных о сырье |
| *DayPlans* | Хранение даты о событии |
| *ProductsPlans* | Хранением данных о событиях и материалах с учётом их количества |
| *DayProductions* | Хранение даты о событии с продукцией |
| *ProductsProductions* | Хранением данных о событиях и продукции с учётом их количества |

В таблицах 3.2 – 3.14 представлено описание полей классов.

Таблица 3.2 – Описание полей доменного класса *Raws*

|  |  |  |
| --- | --- | --- |
| Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
| *int* | *rawid* | Необходим для хранения уникального номера сырья |
| *string* | *rawName* | Необходим для хранения имени сырья |

Таблица 3.3 – Описание полей доменного класса *Norms*

|  |  |  |  |
| --- | --- | --- | --- |
| Тип данных свойства | | Свойство доменного класса | Предназначение свойства доменного класса |
| *Int* | | *normId* | Необходим для хранения уникального номера сырья |
| *Int* | | *rawId* | Необходим для хранения уникальной связи с таблицей *Raws* |
| *Int* | | *productId* | Необходим для хранения уникального номера продукта |
| Int | quantity | | Необходим для хранения количества сырья |

Таблица 3.4 – Описание полей доменного класса *DayPlans*

|  |  |  |
| --- | --- | --- |
| Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
| *int* | *dayPlanid* | Для хранения уникального номера события |
| *string* | *date* | Необходим для хранения даты к событию |

Таблица 3.5 – Описание полей доменного класса *ProductsPlans*

|  |  |  |
| --- | --- | --- |
| Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
| *int* | *produtPlan* | Необходим для хранения уникального номера события |
| *Int* | *dayPlanid* | Для хранения уникального номера события |
| *int* | *productId* | Необходим для хранения уникального номера продукта |
| *Int* | *count* | Необходим для хранения количества |

Таблица 3.6 – Описание полей доменного класса *DayProductions*

|  |  |  |
| --- | --- | --- |
| Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
| *int* | *dayProductionsId* | Необходим для хранения уникального номера продукта |
| *String* | *date* | Необходим для хранения даты к событию |

Таблица 3.7 – Описание полей доменного класса *ProductsProductions*

|  |  |  |
| --- | --- | --- |
| Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
| *int* | *productProductionid* | Необходим для хранения уникального номера продукции |
| *string* | *dayproductionsid* | Необходим для хранения уникального номера продукта из таблицы *DayProductions* |
| *Int* | *productid* | Необходим для хранения уникального номера |
| *Int* | *count* | Необходим для хранения количества |

Таблица 3.8 – Описание полей доменного класса *Materials*

|  |  |  |
| --- | --- | --- |
| Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
| *int* | *materialId* | Необходим для хранения уникального номера материала |
| *String* | *materialName* | Необходим для хранения названия материала |

Таблица 3.9 – Описание полей доменного класса *ProdutsMaterials*

|  |  |  |
| --- | --- | --- |
| Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
| *int* | *productMaterials* | Необходим для хранения уникального номера материала |
| *Int* | *materialId* | Необходим для хранения идентификатора из таблицы *Materials* |
| *Int* | *productId* | Необходим для хранения идентификатора из таблицы Products |
| *Int* | *quanity* | Необходим для хранения количества материала по текущей таблице |

Таблица 3.10 – Основные классы

|  |  |
| --- | --- |
| Исходное название класса | Назначение класса |
| 1 | 2 |
| *Materials* | Необходим для хранения данных о материалах |

Продолжение таблицы 3.10 – Основные классы

|  |  |
| --- | --- |
| 1 | 2 |
| *Products* | Необходим для хранения данных о продукции |
| *Raws* | Необходим для хранения данных о сырье |
| *Users* | Необходим для хранения данных о пользователях нашей информационной системы |

Таблица 3.11 – Описание свойств класса *Materials*

|  |  |  |
| --- | --- | --- |
| Тип атрибута | Имя атрибута | Назначение |
| *ObjectId* | *Id* | Необходим для хранения уникального номера объекта Materials |
| *String* | *Name* | Необходим для уникального имени материала |

Таблица 3.12 – Описание свойств класса *Products*

|  |  |  |
| --- | --- | --- |
| Тип атрибута | Имя атрибута | Назначение |
| 1 | 2 | 3 |
| *ObjectId* | *Id* | Необходим для хранения уникального идентификатора продута |
| *string* | *Name* | Необходим для хранения уникального имени продукта |
| *Object* | *Plans* | Необходим для хранения уникальной информации о событии с указанием даты и количества необходимой продукции |
| *Object* | Productions | Необходим для хранения уникальной информации о продукции с указанием даты и количества необходимой продукции |

Продолжение таблицы 3.12 – Основные классы

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| *Object* | Norms | Необходим для хранения уникальной информации о сырье с указанием названия сырья и количества |
| *Object* | Materials | Необходим для хранения уникальной информации о материале с указанием названия материала и количества |

Таблица 3.13 – Описание свойств класса *Raws*

|  |  |  |
| --- | --- | --- |
| Тип атрибута | Имя атрибута | Назначение |
| *ObjectId* | *Id* | Необходим для хранения уникального идентификатора сырья |
| *String* | *Name* | Необходим для хранения имени сырья |

Таблица 3.14 – Описание свойств класса *Users*

|  |  |  |
| --- | --- | --- |
| Тип атрибута | Имя атрибута | Назначение |
| *ObjectId* | *\_id* | Необходим для хранения уникального идентификатора пользователя информационной системы |
| *String* | Level | Необходим для хранения уникального уровня доступа для каждого пользователя информационной системы |
| *String* | Login | Необходим для хранения уникального логина для каждого пользователя информационной системы |
| *String* | Password | Необходим для хранения уникального пароля для каждого пользователя информационной системы |

# **3.2 Описание интерфейсов.**

Неотъемлемой частью программы должен быть интерфейс. Потому что именно интерфейс является тем связующим компонентом, с помощью которого пользователь посылает компьютеру очередную инструкцию для её выполнения.

Интерфейс разрабатываемой программы был разработан в достаточно простой и, в то же время, понятной пользователю форме.

Классы окон и их назначение:

* *MainWindow* (авторизация);
* *AddUserWindow*(добавление пользователя согласно роли);
* *AdminWindow*(окно администратора);
* *EditUserWindow*(редактирование пользователей);
* *ManagerWindow* (окно менеджера);
* *ProductDetailWindow* (окно нормы материалов);
* *SearchItemWindow* (вывод материалов или сырья);
* *AddNormWindow* (добавление норм производства);
* *AddPlanWindow* (добавление плана производства);
* *AddProductionWindow* (добавление производства);
* *AddProductWindow* (добавление данных о продукте);
* *InputStringWindow* (ввод текстовых данных);
* *UserWindow* (окно пользователя);
* *ProductDetailsWindow* (информация о продуктах);
* *GuestWindow* (окно гостя);

При запуске программы пользователь видит окно, изображенное на рисунке 6.

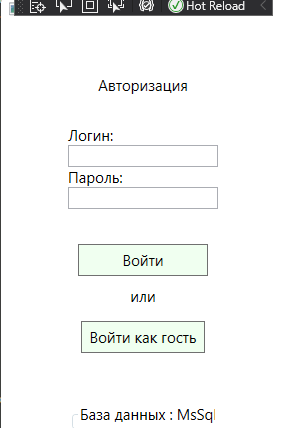


Рисунок 6 – Авторизация

Пользователь должен авторизоваться. Для авторизации нужно ввести логин и пароль аккаунта. Если были введены неверные данные, программа оповестит пользователя об этом. Если данные были введены верно, пользователю откроется меню, согласно его должности.

Осуществив вход, Администратор увидит окно, изображенное на рисунке 7.

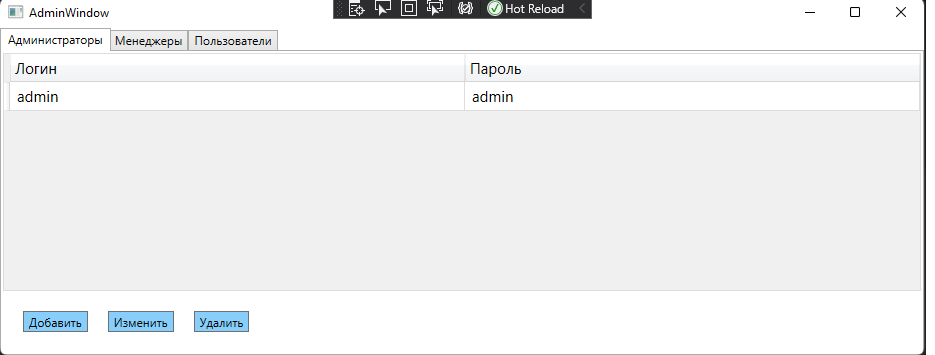


Рисунок 7 – Окно администратора

Здесь администратор может работать со справочником пользователей.

Нажав на кнопку «Администраторы», администратор увидит окно, изображенное на рисунке 8.

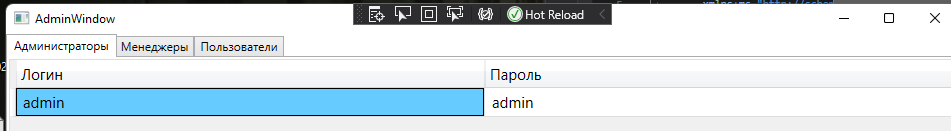


Рисунок 8 – Список администраторов

Нажав на кнопку «Менеджеры», администратор увидит окно, изображенное на рисунке 9.

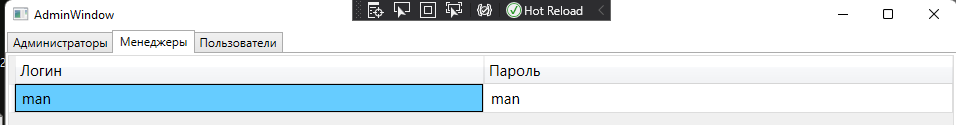


Рисунок 9 – Список менеджеров

Нажав на кнопку «Пользователи», администратор увидит окно, изображенное на рисунке 10.

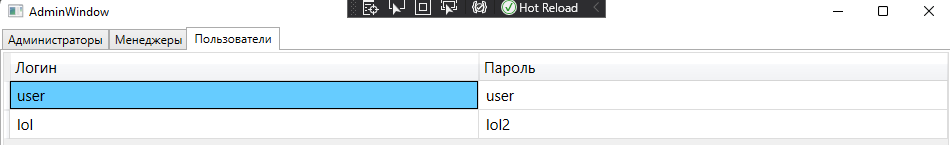


Рисунок 10 – Список пользователей

Находясь в любом из списков, можно добавить нового пользователя, согласно роли списка. Для этого необходимо нажать кнопку «Добавить» и откроется новое окно, изображенное на рисунке 11. Результат добавления изображён на рисунке 12.

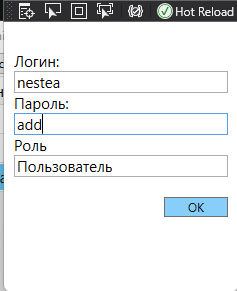


Рисунок 11 – Окно добавления

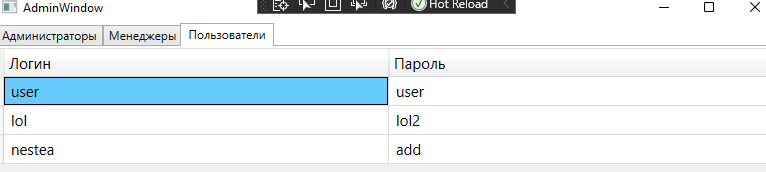


Рисунок 12 – Результат добавления

Также мы можем изменять информацию о пользователях. Для этого необходимо выбрать пользователя из списка и нажать кнопку «Изменить», после этого откроется новое окно, изображенное на рисунке 13. В этом окне мы можем выбрать роль из выпадающего списка.

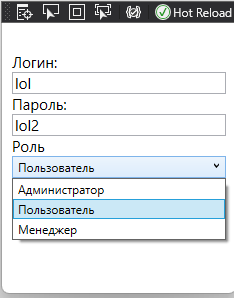


Рисунок 13 – Изменение данных о пользователях

Кнопка «Удалить» удаляет выбранного пользователя из списка безвозвратно.

Осуществив вход, менеджер увидит окно, изображенное на рисунке 14.

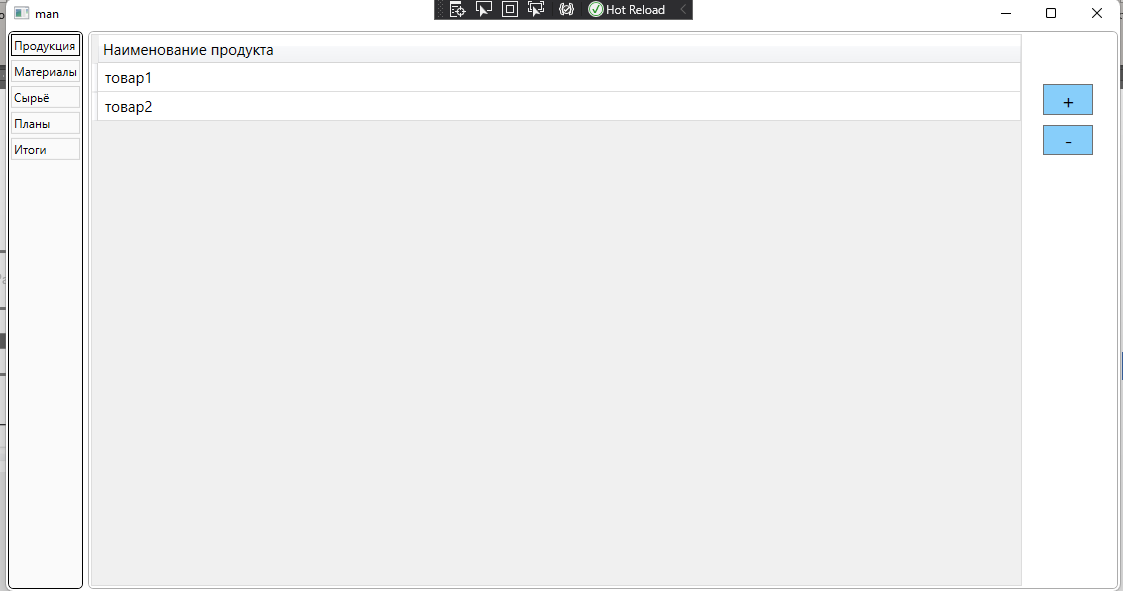


Рисунок 14 – Окно менеджера

Здесь менеджер может работать со справочником продукции, материалов и сырья. А также задавать планы и итоги.

Нажав на кнопку «Продукция», менеджер увидит окно, изображенное на рисунке 15.

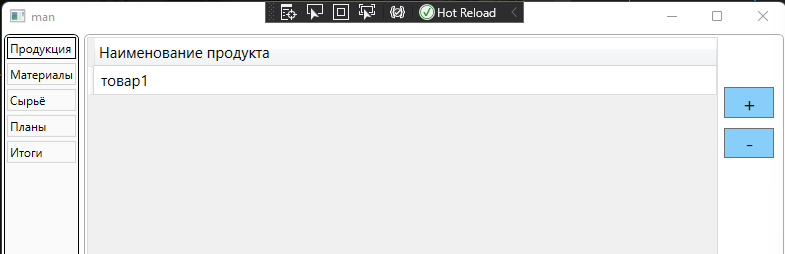
****

Рисунок 15 – Список продукции

Нажав на кнопку «Материалы», менеджер увидит окно, изображенное на рисунке 16.

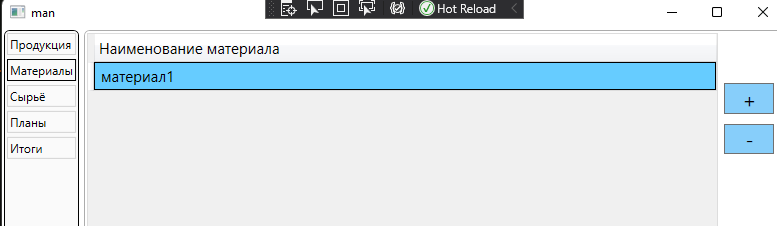
****

Рисунок 16 – Список материалов

Нажав на кнопку «Сырьё», менеджер увидит окно, изображенное на рисунке 17.

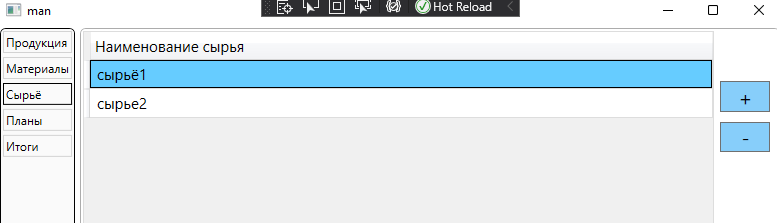
****

Рисунок 17 – Список сырья

Нажав на кнопку «Планы», менеджер увидит окно, изображенное на рисунке 18.

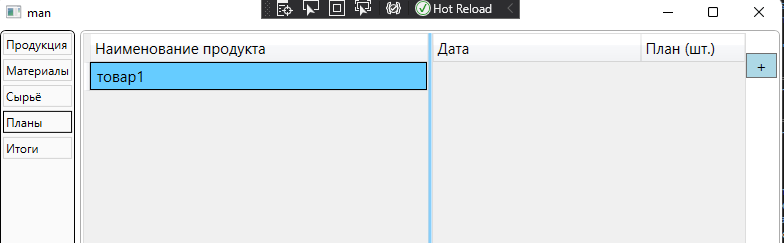
****

Рисунок 18 – Список планов

Нажав на кнопку «Итоги», менеджер увидит окно, изображенное на рисунке 19.

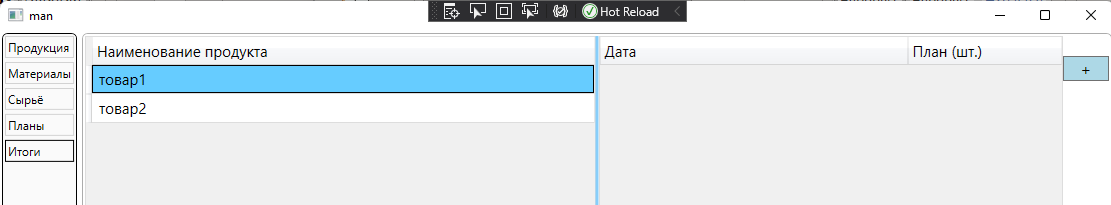


Рисунок 19 – Список итогов

Находясь в любом из списков, можно добавить новый элемент. Для этого необходимо нажать кнопку «+» и откроется новое окно, изображенное на рисунке 20. Результат добавления изображён на рисунке 21. Добавление в списки «Продукция», «Материалы» и «Сырьё» происходит аналогично.

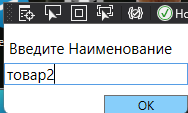
****

Рисунок 20 – Окно добавления

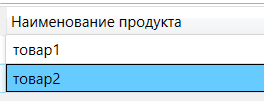
****

Рисунок 21 – Результат добавления

Кнопка «-» удаляет выбранный элемент из списка безвозвратно. Изображение кнопок предоставлено на рисунке 22.

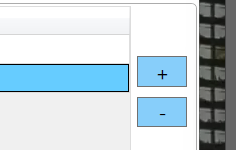
****

Рисунок 22 – Кнопки воздействия

Добавленный нами товар пока не содержит даты и количества, но нажав на него, а затем на кнопку «+», мы можем дополнить информацию. Менеджер увидит окно, изображенное на рисунке 23. Добавление в списки «Итоги» и «Планы» происходит аналогично.

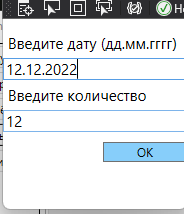
****

Рисунок 23 – Результат добавления

Программа содержит всевозможные проверки на ошибки, поэтому после корректного ввода информации результат добавления будет изображен на рисунке 24.

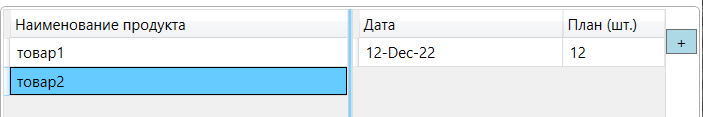
****

Рисунок 24 – Результат добавления

Осуществив вход, клиент увидит окно, изображенное на рисунке 25.

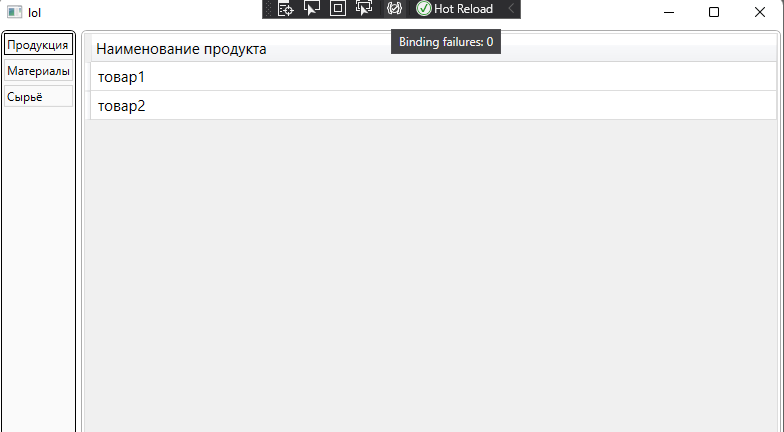


Рисунок 25 – Окно клиента

Здесь клиент может просматривать справочники продукции, материалов и сырья.

Нажав на кнопку «Продукция», клиент увидит окно, изображенное на рисунке 26.

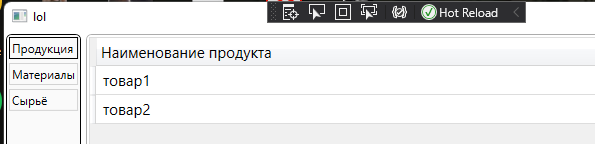
****

Рисунок 26 – Список продукции

Нажав на кнопку «Материалы», клиент увидит окно, изображенное на рисунке 27.

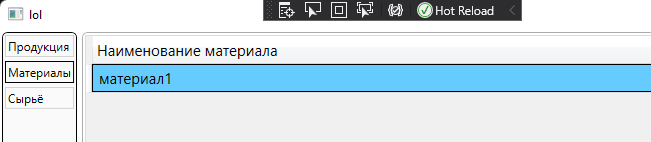
****

Рисунок 27 – Список материалов

Нажав на кнопку «Сырьё», клиент увидит окно, изображенное на рисунке 28.

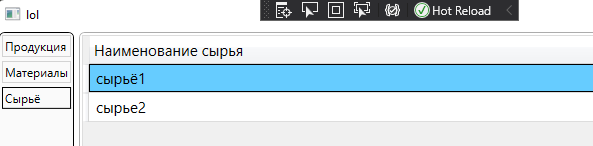
****

Рисунок 28 – Список сырья

Возвращаясь к началу и авторизации, при нажатии кнопки «Войти как гость» нам не понадобится аккаунт для входа, однако функционал будет максимально ограничен. Гость увидит окно, изображенное на рисунке 29.

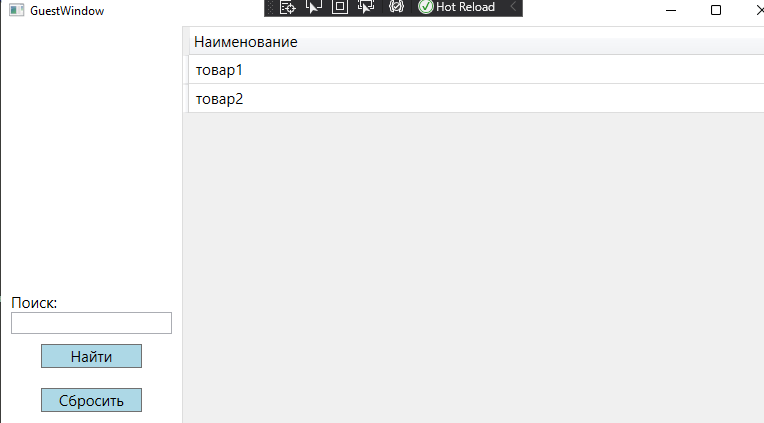
****

Рисунок 29 – Окно гостя

Здесь гость может просматривать справочники продукции.

Введя информацию в поле поиска и нажав на кнопку «Найти», клиент увидит отфильтрованные товары, соответствующие запросу. Результат изображен на рисунке 30.

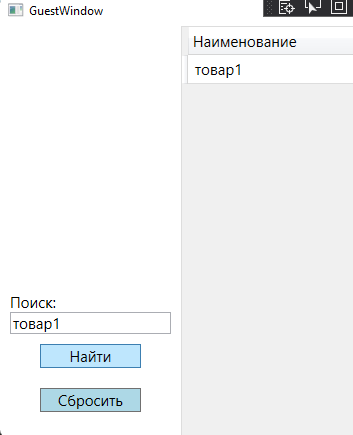
****

Рисунок 30 – Поиск продукции

Нажав на кнопку «Сброс», все товары снова показываются в списке.

На рисунке 31, предоставлена статистика по заполненным товарам.

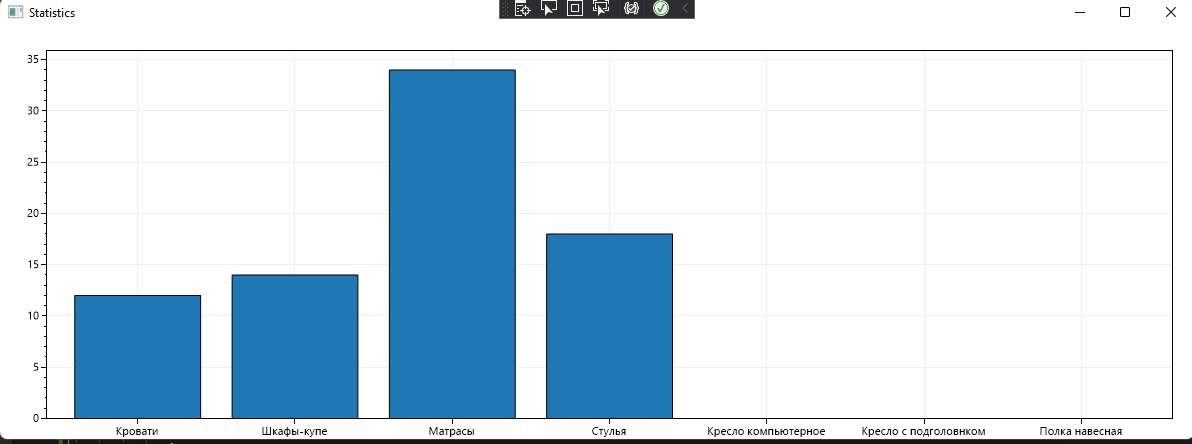


Рисунок 31 – Статистика по заполненным товарам

# **4 ТЕСТИРОВАНИЕ**

Программа была протестирована на наличие ошибок, недоработок, а также на стабильность работы. Программа содержит всевозможные проверки на ошибки, такие как:

* проверка на правильность ввода пароля и логина;
* в окне администратора существуют проверки на добавление новых пользователей, во имя избежание конфликтов имен, наличия пробелов и несоответствию размера;
* проверка на валидность информации;
* проверка на соответствие введенной информации требуемому объекту;
* проверка на язык введенной информации. Интерфейс разработан на русском языке, значит и данные должны быть введены в соответствии.

На рисунках 32-37– предоставлены примеры ошибок.

На рисунке 32 предоставлена ошибка изменения собственной информации.

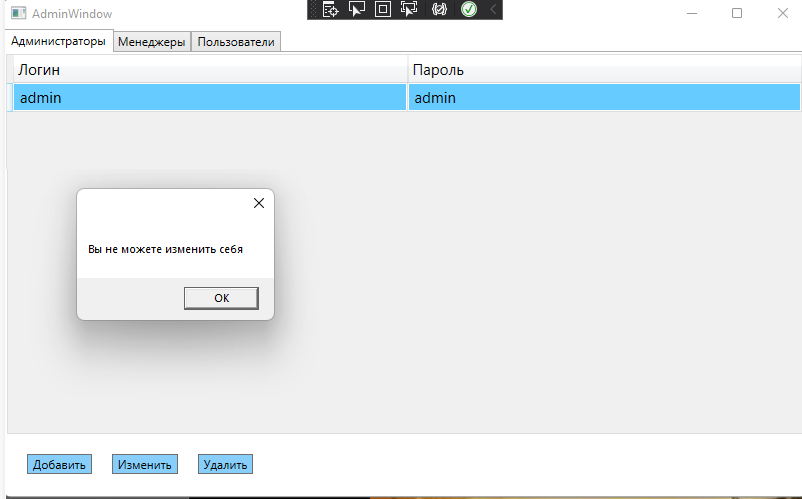


Рисунок 32 – Ошибка изменения пользователя

На рисунке 33 предоставлена ошибка регистрации пользователей под уже существующими логинами.

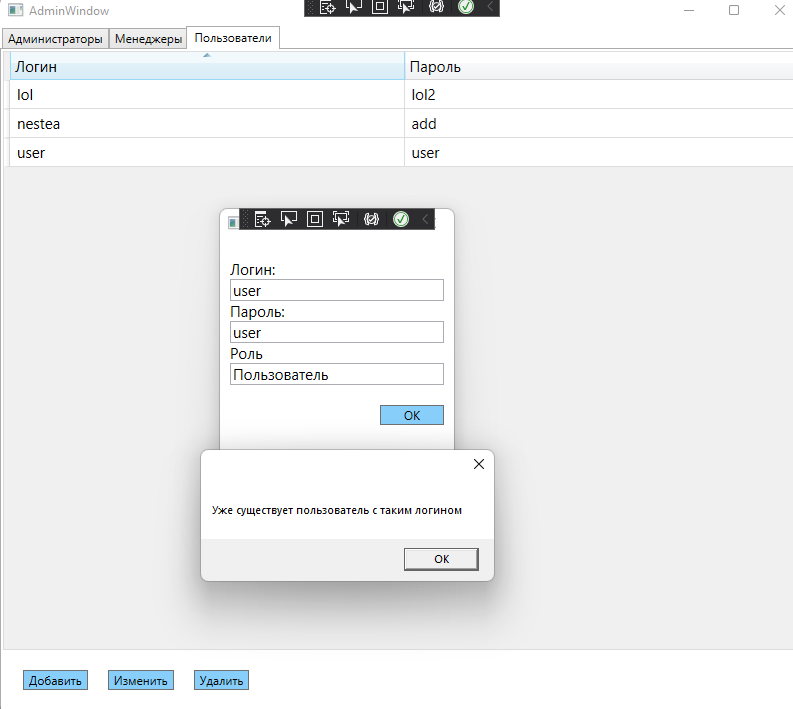


Рисунок 33 – Ошибка регистрации под занятым именем

На рисунке 34 предоставлена ошибка регистрации пользователей c некорректными данными.

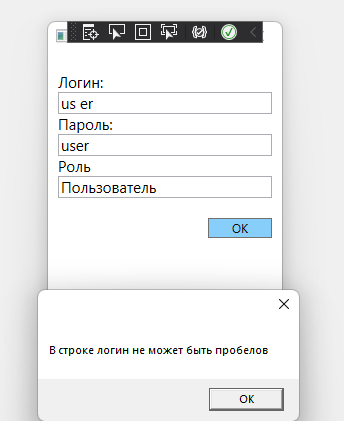


Рисунок 34 – Ошибка правописания

На рисунке 35 предоставлена ошибка написания даты в неверном формате.

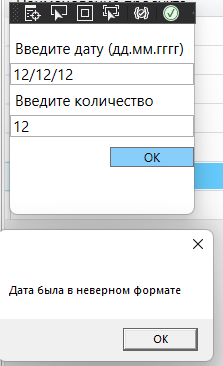


Рисунок 35 – Ошибка написания даты

На рисунке 36 предоставлена ошибка написания количества в неверном формате.

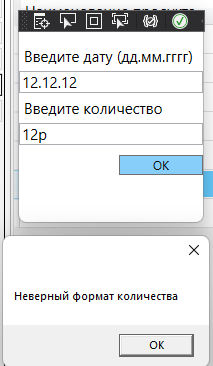


Рисунок 36 – Ошибка написания количества

На рисунке 37 предоставлена ошибка авторизации.

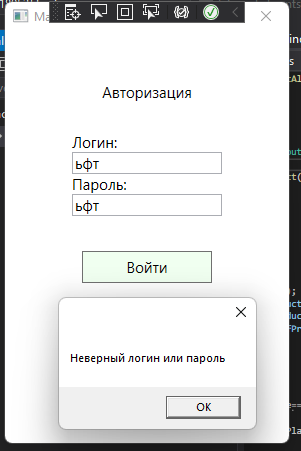


Рисунок 37 – Ошибка авторизации

Разработанное приложение содержит необходимый функционал для выполнения поставленной задачи. Поставленные задачи были решены и описаны в разделе 3, их выполнение показано в таблице 4.1.

Таблица 4.1

|  |  |
| --- | --- |
| **Поставленные задачи** | **Ссылка на интерфейс** |
| План производства | Таблица 3.5, рисунок 18 |
| Расчет потребности в сырье | Таблица 3.2, рисунок 17 |
| Расчет потребности в материалах | Таблица 3.3, рисунок 16 |
| Месячные отчеты(итоги) | Таблица 3.7, рисунок 19 |
| Статистика | Рисунок 31 |

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения, данной курсовой работы, было спроектировано и разработано приложение. Для этого было проведено изучение предметной области и выявление необходимых данных. Для хранения данных спроектирована и разработана база данных. Разработан пользовательский интерфейс с использованием технологии *WPF*. Также реализована ролевая политика для установки ограничения редактирования и просмотра данных пользователям для обеспечения защиты хранимых данных. Разработанные приложения содержат необходимый функционал для выполнения поставленной задачи.

В процессе создания приложения были пройдены такие этапы разработки, как постановка задачи, составление логической модели предметной области, создание и реализация алгоритма работы программы с последующим тестированием.

Преимуществом разработанной нами системы является ее относительно низкие требования к установке и использования, а также простота и удобство в эксплуатации.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Джозеф Альбахари и Бен Альбахари. C# 7.0. Карманный справочник. – «O’Reilly Media, Inc», 2017. – 156 с.
2. Рихтер, Джеффри CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Джеффри Рихтер. - М.: Питер, 2013. - 928 c
3. Когаловский М. Р. Энциклопедия технологий баз данных. — М.: Финансы и статистика, 2002. — 800 с.
4. Актеры и прецеденты [Электронный ресурс]. – URL: <https://studopedia.su/6_42762_akteri-i-pretsedenti.html>., свободный. – Загл. с экрана.
5. Введение в WPF [Электронный ресурс]. – URL: <https://studfile.net/preview/4532318/>., свободный. – Загл. с экрана.

# **ПРИЛОЖЕНИЕ А**

**Листинг программы**

**Folder Data.Models:**

**Material.cs**

namespace Factory.Data.Models

{

public class Material

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**Norm.cs**

namespace Factory.Data.Models

{

public class Norm

{

public int Id { get; set; }

public Raw Raw { get; set; }

public double Quantity { get; set; }

}

}

**Product.cs**

using System.Collections.Generic;

namespace Factory.Data.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public List<ProductPlan> Plans { get; set; } = new();

public List<ProductProduction> Productions { get; set; } = new();

public List<Norm> Norms { get; set; } = new();

public List<ProductMaterial> Materials { get; set; } = new();

}

}

**ProductMaterial.cs**

namespace Factory.Data.Models

{

public class ProductMaterial

{

public int Id { get; set; }

public double Quantity { get; set; }

public Material Material { get; set; }

}

}

**ProductPlan.cs**

using System;

namespace Factory.Data.Models

{

public class ProductPlan

{

public int Id { get; set; }

public DateTime Date { get; set; }

public int Count { get; set; }

public string DateStr => Date.ToString("d");

}

}

**ProductProduction.cs**

using System;

namespace Factory.Data.Models

{

public class ProductProduction

{

public int Id { get; set; }

public DateTime Date { get; set; }

public int Count { get; set; }

}

}

**Raw.cs**

namespace Factory.Data.Models

{

public class Raw

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**User.cs**

using System.Collections.Generic;

namespace Factory.Data.Models

{

public class User

{

public int Id { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public int Level { get; set; }

public string Access => Accesses[Level];

public static readonly Dictionary<int, string> Accesses = new()

{

[1] = "Администратор",

[2] = "Менеджер",

[3] = "Пользователь"

};

}

}

**Validator.cs**

using System;

using System.Linq;

using System.Text.RegularExpressions;

namespace Factory.Data.Models

{

public static class Validator

{

private static bool IsRussian(this string str)

{

return Regex.IsMatch(str, @"\p{IsCyrillic}");

}

private static bool CheckLength(this string str, int leftBorder = 3, int rightBorder = 100)

{

return (str.Length >= 3) && (str.Length <= 100);

}

public static User Auth(DataLayer data, string login, string password)

{

return data.Users.GetAll().FirstOrDefault(x => (x.Login == login) && (x.Password == password));

}

public static bool CompareDate(DateTime first, DateTime second)

{

return (first.Day == second.Day) && (first.Month == second.Month) && (first.Year == second.Year);

}

public static string Valid(ProductProduction material, Product product)

{

if ((material.Count <= 0) || (material.Count >= 100000))

{

return "Неверное количество";

}

if ((material.Date <= DateTime.Now.AddYears(-50)) || (material.Date >= DateTime.Now.AddYears(50)))

{

return "Неверная дата";

}

if (product.Plans.Any(x => CompareDate(x.Date, material.Date)))

{

return "На этот день уже назначен результат";

}

return null;

}

public static string Valid(ProductPlan material, Product product)

{

if ((material.Count <= 0) || (material.Count >= 100000))

{

return "Неверное количество";

}

if ((material.Date <= DateTime.Now.AddYears(-50)) || (material.Date >= DateTime.Now.AddYears(50)))

{

return "Неверная дата";

}

if (product.Plans.Any(x => CompareDate(x.Date, material.Date)))

{

return "На этот день уже назначен план";

}

return null;

}

public static string Valid(Norm material)

{

if (material.Raw == null)

{

return "Не выбрано сырьё";

}

if (material.Quantity is < 0.01 or > 10000)

{

return "Неверное количество";

}

return null;

}

public static string Valid(ProductMaterial material)

{

if (material.Material == null)

{

return "Не выбран материал";

}

if (material.Quantity is < 0.01 or > 10000)

{

return "Неверное количество";

}

return null;

}

public static string Valid(Product raw)

{

if (!raw.Name.CheckLength())

{

return "Неверный формат наименования";

}

if (!raw.Name.IsRussian())

{

return "Название должно быть на кириллице";

}

return null;

}

public static string Valid(Material raw)

{

if (!raw.Name.CheckLength())

{

return "Неверный формат наименования";

}

if (!raw.Name.IsRussian())

{

return "Название должно быть на кириллице";

}

return null;

}

public static string Valid(Raw raw)

{

if (!raw.Name.CheckLength())

{

return "Неверный формат наименования";

}

if (!raw.Name.IsRussian())

{

return "Название должно быть на кириллице";

}

return null;

}

public static string Valid(User user, IRepository<User> data, bool ignoreLogin = false)

{

if (!ignoreLogin)

{

if (data.GetAll().Any(x => x.Login == user.Login))

{

return "Уже существует пользователь с таким логином";

}

}

if (user.Login.Contains(' '))

{

return "В строке логин не может быть пробелов";

}

if (!user.Login.CheckLength())

{

return "Строка логин некорректна";

}

if (!user.Password.CheckLength())

{

return "Строка пароль неккоректна";

}

return null;

}

}

}

**Folder Data.Sql.Entities:**

**Data.cs**

using Factory.Data.Sql.Repositories;

namespace Factory.Data.Sql.Entities

{

public static class Data

{

public static DataLayer Get()

{

return new(new Materials(), new Products(), new Raws(), new Users());

}

}

}

**Material.cs**

namespace Factory.Data.Sql.Entities

{

public class Material

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**Norm.cs**

namespace Factory.Data.Sql.Entities

{

public class Norm

{

public int Id { get; set; }

public int RawId { get; set; }

public int ProductId { get; set; }

public double quantity { get; set; }

}

}

**Product.cs**

namespace Factory.Data.Sql.Entities

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**ProductMaterial.cs**

namespace Factory.Data.Sql.Entities

{

public class ProductMaterial

{

public int Id { get; set; }

public int MaterialId { get; set; }

public int ProductId { get; set; }

public double Quantity { get; set; }

}

}

**ProductPlan.cs**

using System;

namespace Factory.Data.Sql.Entities

{

public class ProductPlan

{

public int Id { get; set; }

public int ProductId { get; set; }

public DateTime Date { get; set; }

public int Count { get; set; }

}

}

**ProductProduction.cs**

using System;

namespace Factory.Data.Sql.Entities

{

public class ProductProduction

{

public int Id { get; set; }

public DateTime DayProduction { get; set; }

public int ProductId { get; set; }

public int Count { get; set; }

}

}

**Raw.cs**

namespace Factory.Data.Sql.Entities

{

public class Raw

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**User.cs**

namespace Factory.Data.Sql.Entities

{

public class User

{

public int Id { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public int Level { get; set; }

}

}

**Folder Data.Sql.Repositories:**

**Materials.cs**

using Factory.Data.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace Factory.Data.Sql.Repositories

{

public class Materials : IRepository<Material>

{

private readonly FactoryContext db = FactoryContext.Get();

public bool Add(Material itemToAdd)

{

try

{

db.Materials.Add(new()

{

Id = GetFreeId(),

Name = itemToAdd.Name

});

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public bool Remove(int id)

{

try

{

db.ProductsMaterials.RemoveRange(db.ProductsMaterials.Where(x => x.MaterialId == id));

db.SaveChanges();

db.Materials.Remove(db.Materials.First(x => x.Id == id));

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public bool Update(Material itemToUpdate)

{

try

{

var item = db.Materials.FirstOrDefault(x => x.Id == itemToUpdate.Id);

item.Name = itemToUpdate.Name;

db.Materials.Update(item);

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public IEnumerable<Material> GetAll()

{

return db.Materials.ToList().Select(x => new Material()

{

Id = x.Id,

Name = x.Name

});

}

public int GetFreeId()

{

return db.Materials.Any() ? db.Materials.Max(x => x.Id) + 1 : 1;

}

public Material Find(int id)

{

return GetAll().FirstOrDefault(x => x.Id == id);

}

}

}

**Products.cs**

using Factory.Data.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace Factory.Data.Sql.Repositories

{

public class Products : IRepository<Product>

{

private readonly FactoryContext db = FactoryContext.Get();

public bool Add(Product itemToAdd)

{

try

{

db.Products.Add(new()

{

Id = GetFreeId(),

Name = itemToAdd.Name

});

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

private int GetFreeProductMaterialId()

{

return db.ProductsMaterials.Any() ? db.ProductsMaterials.Max(x => x.Id) + 1 : 1;

}

public bool Remove(int id)

{

try

{

db.Norms.RemoveRange(db.Norms.Where(x => x.ProductId == id));

db.SaveChanges();

db.ProductsMaterials.RemoveRange(db.ProductsMaterials.Where(x => x.ProductId == id));

db.SaveChanges();

db.ProductsProductions.RemoveRange(db.ProductsProductions.Where(x => x.ProductId == id));

db.SaveChanges();

db.ProductsPlans.RemoveRange(db.ProductsPlans.Where(x => x.ProductId == id));

db.SaveChanges();

db.Products.Remove(db.Products.First(x => x.Id == id));

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

private int GetFreeProductionId()

{

return db.ProductsProductions.Any() ? db.ProductsProductions.Max(x => x.Id) + 1 : 1;

}

public int GerFreePlanId()

{

return db.ProductsPlans.Any() ? db.ProductsPlans.Max(x => x.Id) + 1 : 1;

}

private int GetFreeNormId()

{

return db.Norms.Any() ? db.Norms.Max(x => x.Id) + 1 : 1;

}

public bool Update(Product itemToUpdate)

{

try

{

var item = db.Products.FirstOrDefault(x => x.Id == itemToUpdate.Id);

item.Name = itemToUpdate.Name;

db.Norms.RemoveRange(db.Norms.Where(x => x.RawId == item.Id));

db.SaveChanges();

db.Norms.AddRange(itemToUpdate.Norms.Select(x => new Entities.Norm()

{

Id = GetFreeNormId(),

ProductId = itemToUpdate.Id,

quantity = x.Quantity,

RawId = itemToUpdate.Id

}));

db.SaveChanges();

db.ProductsProductions.RemoveRange(db.ProductsProductions.Where(x => x.ProductId == item.Id));

db.SaveChanges();

db.ProductsProductions.AddRange(itemToUpdate.Productions.Select(x => new Entities.ProductProduction()

{

Id = GetFreeProductionId(),

DayProduction = x.Date,

Count = x.Count,

ProductId = itemToUpdate.Id

}));

db.SaveChanges();

db.ProductsPlans.RemoveRange(db.ProductsPlans.Where(x => x.ProductId == item.Id));

db.SaveChanges();

db.ProductsPlans.AddRange(itemToUpdate.Plans.Select(x => new Entities.ProductPlan()

{

Id = GerFreePlanId(),

Date = x.Date,

Count = x.Count,

ProductId = itemToUpdate.Id

}));

db.SaveChanges();

db.ProductsMaterials.RemoveRange(db.ProductsMaterials.Where(x => x.MaterialId == item.Id));

db.SaveChanges();

db.ProductsMaterials.AddRange(itemToUpdate.Materials.Select(x => new Entities.ProductMaterial()

{

Id = GetFreeProductMaterialId(),

MaterialId = itemToUpdate.Id,

Quantity = x.Quantity,

ProductId = x.Id

}));

db.SaveChanges();

db.Products.Update(item);

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public IEnumerable<Product> GetAll()

{

return db.Products.ToList().Select(x => new Product()

{

Id = x.Id,

Name = x.Name,

Plans = db.ProductsPlans.Where(p => p.ProductId == x.Id).ToList().Select(p => new ProductPlan()

{

Id = p.Id,

Count = p.Count,

Date = p.Date

}).ToList(),

Productions = db.ProductsProductions.Where(p => p.ProductId == x.Id).ToList().Select(p => new ProductProduction()

{

Id = p.Id,

Count = p.Count,

Date = p.DayProduction

}).ToList(),

Materials = db.ProductsMaterials.Where(m => m.MaterialId == x.Id)

.Select(m => new ProductMaterial()

{

Id = m.Id,

Material = new Materials().Find(m.MaterialId),

Quantity = m.Quantity

}).ToList(),

Norms = db.Norms.Where(n => n.RawId == x.Id).ToList().Select(s => new Norm()

{

Id = s.Id,

Raw = new Raws().Find(s.RawId),

Quantity = s.quantity

}).ToList()

});

}

public int GetFreeId()

{

return db.Products.Any() ? db.Products.Max(x => x.Id) + 1 : 1;

}

public Product Find(int id)

{

return GetAll().FirstOrDefault(x => x.Id == id);

}

}

}

**Raws.cs**

using Factory.Data.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace Factory.Data.Sql.Repositories

{

public class Raws : IRepository<Raw>

{

private readonly FactoryContext db = FactoryContext.Get();

public bool Add(Raw itemToAdd)

{

try

{

db.Raws.Add(new()

{

Id = GetFreeId(),

Name = itemToAdd.Name

});

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public bool Remove(int id)

{

try

{

db.Norms.RemoveRange(db.Norms.Where(x => x.RawId == id));

db.SaveChanges();

db.Raws.Remove(db.Raws.First(x => x.Id == id));

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public bool Update(Raw itemToUpdate)

{

try

{

var item = db.Raws.FirstOrDefault(x => x.Id == itemToUpdate.Id);

item.Name = itemToUpdate.Name;

db.Raws.Update(item);

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public IEnumerable<Raw> GetAll()

{

return db.Raws.ToList().Select(x => new Raw()

{

Id = x.Id,

Name = x.Name

});

}

public int GetFreeId()

{

return db.Raws.Any() ? db.Raws.Max(x => x.Id) + 1 : 1;

}

public Raw Find(int id)

{

return GetAll().FirstOrDefault(x => x.Id == id);

}

}

}

**Users.cs**

using Factory.Data.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace Factory.Data.Sql.Repositories

{

public class Users : IRepository<User>

{

private readonly FactoryContext db = FactoryContext.Get();

public bool Add(User itemToAdd)

{

try

{

db.Users.Add(new()

{

Id = GetFreeId(),

Level = itemToAdd.Level,

Login = itemToAdd.Login,

Password = itemToAdd.Password

});

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public bool Remove(int id)

{

try

{

db.Users.Remove(db.Users.First(x => x.Id == id));

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public bool Update(User itemToUpdate)

{

try

{

var item = db.Users.FirstOrDefault(x => x.Id == itemToUpdate.Id);

item.Login = itemToUpdate.Login;

item.Password = itemToUpdate.Password;

item.Level = itemToUpdate.Level;

db.Users.Update(item);

db.SaveChanges();

}

catch (Exception)

{

db.RollBack();

return false;

}

return true;

}

public IEnumerable<User> GetAll()

{

return db.Users.ToList().Select(x => new User()

{

Id = x.Id,

Level = x.Level,

Login = x.Login,

Password = x.Password

});

}

public int GetFreeId()

{

return db.Users.Any() ? db.Users.Max(x => x.Id) + 1 : 1;

}

public User Find(int id)

{

return GetAll().FirstOrDefault(x => x.Id == id);

}

}

}

**FactoryContext.cs**

using Factory.Data.Sql.Entities;

using Microsoft.EntityFrameworkCore;

namespace Factory.Data.Sql

{

public class FactoryContext : DbContext

{

private static readonly string conStr =

"Server=DESKTOP-AIDC8ES\\SQLEXPRESS;" +

"Database=ll2;" +

"Trusted\_Connection=True;" +

"MultipleActiveResultSets=true;";

public DbSet<Material> Materials { get; set; }

public DbSet<User> Users { get; set; }

public DbSet<Norm> Norms { get; set; }

public DbSet<Product> Products { get; set; }

public DbSet<ProductMaterial> ProductsMaterials { get; set; }

public DbSet<ProductPlan> ProductsPlans { get; set; }

public DbSet<ProductProduction> ProductsProductions { get; set; }

public DbSet<Raw> Raws { get; set; }

public FactoryContext(DbContextOptions<FactoryContext> options)

: base(options) { }

private static readonly FactoryContext context;

static FactoryContext()

{

DbContextOptionsBuilder<FactoryContext> optionsBuilder = new();

optionsBuilder.UseSqlServer(conStr);

context = new(optionsBuilder.Options);

}

public void RollBack()

{

foreach (var entry in ChangeTracker.Entries())

{

switch (entry.State)

{

case EntityState.Modified:

entry.State = EntityState.Unchanged;

break;

case EntityState.Added:

entry.State = EntityState.Detached;

break;

case EntityState.Deleted:

entry.Reload();

break;

}

}

}

public static FactoryContext Get()

{

return context;

}

}

}

**DataLayer.cs**

using Factory.Data.Models;

using System;

namespace Factory.Data

{

public class DataLayer

{

public DataLayer(IRepository<Material> materials, IRepository<Product> products, IRepository<Raw> raws, IRepository<User> users)

{

Materials = materials ?? throw new ArgumentNullException(nameof(materials));

Products = products ?? throw new ArgumentNullException(nameof(products));

Raws = raws ?? throw new ArgumentNullException(nameof(raws));

Users = users ?? throw new ArgumentNullException(nameof(users));

}

public static DataLayer Sql() => Data.Sql.Entities.Data.Get();

public IRepository<Material> Materials { get; set; }

public IRepository<Product> Products { get; set; }

public IRepository<Raw> Raws { get; set; }

public IRepository<User> Users { get; set; }

}

}

**IRepository.cs**

using System.Collections.Generic;

namespace Factory.Data

{

public interface IRepository<T>

{

public bool Add(T itemToAdd);

public bool Remove(int id);

public bool Update(T itemToUpdate);

public IEnumerable<T> GetAll();

public int GetFreeId();

public T Find(int id);

}

}

**Folder AdminWindows**

**AddUserWindow.xaml.cs**

using Factory.Data;

using Factory.Data.Models;

using System.Windows;

namespace Factory.AdminWindows

{

/// <summary>

/// Логика взаимодействия для AddUserWindow.xaml

/// </summary>

public partial class AddUserWindow : Window

{

public User Result { get; set; }

private readonly DataLayer data;

public AddUserWindow(int access, DataLayer data)

{

this.data = data;

InitializeComponent();

Result = new()

{

Level = access

};

LevelTextBox.Text = User.Accesses[access];

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

Result.Login = LoginTextBox.Text;

Result.Password = PasswordTextBox.Text;

var validRes = Validator.Valid(Result, data.Users);

if (validRes != null)

{

MessageBox.Show(validRes);

return;

}

DialogResult = true;

}

}

}

**AdminWindow.xaml.cs**

using Factory.Data;

using Factory.Data.Models;

using System.Linq;

using System.Windows;

namespace Factory.AdminWindows

{

/// <summary>

/// Логика взаимодействия для AdminWindow.xaml

/// </summary>

public partial class AdminWindow : Window

{

private readonly User user;

private readonly DataLayer data;

public AdminWindow(User user, DataLayer data)

{

this.data = data;

this.user = user;

InitializeComponent();

UpdateData();

}

private void UpdateData()

{

var users = data.Users.GetAll();

AdminDataGrid.ItemsSource = users.Where(x => x.Level == 1);

ManagerDataGrid.ItemsSource = users.Where(x => x.Level == 2);

UsersDataGrid.ItemsSource = users.Where(x => x.Level == 3);

}

private void AddAdminButton\_Click(object sender, RoutedEventArgs e)

{

var window = new AddUserWindow(1, data);

if (window.ShowDialog() == true)

{

data.Users.Add(window.Result);

UpdateData();

}

}

private void EditAdminButton\_Click(object sender, RoutedEventArgs e)

{

if (AdminDataGrid.SelectedItem is not null and User user)

{

if (user.Login == this.user.Login)

{

MessageBox.Show("Вы не можете изменить себя");

}

else

{

var window = new EditUserWindow(user, data);

if (window.ShowDialog() == true)

{

data.Users.Update(user);

UpdateData();

}

}

}

else

{

MessageBox.Show("Сначала сделайте выбор");

}

}

private void DeleteAdminButton\_Click(object sender, RoutedEventArgs e)

{

if (AdminDataGrid.SelectedItem is not null and User user)

{

if (user.Login == this.user.Login)

{

MessageBox.Show("Вы не можете удалить себя");

}

else

{

data.Users.Remove(user.Id);

UpdateData();

}

}

else

{

MessageBox.Show("Сначала сделайте выбор");

}

}

private void AddManagerButton\_Click(object sender, RoutedEventArgs e)

{

var window = new AddUserWindow(2, data);

if (window.ShowDialog() == true)

{

data.Users.Add(window.Result);

UpdateData();

}

}

private void EditManagerButton\_Click(object sender, RoutedEventArgs e)

{

if (ManagerDataGrid.SelectedItem is not null and User user)

{

if (user.Login == this.user.Login)

{

MessageBox.Show("Вы не можете изменить себя");

}

else

{

var window = new EditUserWindow(user, data);

if (window.ShowDialog() == true)

{

data.Users.Update(user);

UpdateData();

}

}

}

else

{

MessageBox.Show("Сначала сделайте выбор");

}

}

private void DeleteManagerButton\_Click(object sender, RoutedEventArgs e)

{

if (ManagerDataGrid.SelectedItem is not null and User user)

{

if (user.Login == this.user.Login)

{

MessageBox.Show("Вы не можете удалить себя");

}

else

{

data.Users.Remove(user.Id);

UpdateData();

}

}

else

{

MessageBox.Show("Сначала сделайте выбор");

}

}

private void AddUserButton\_Click(object sender, RoutedEventArgs e)

{

var window = new AddUserWindow(3, data);

if (window.ShowDialog() == true)

{

data.Users.Add(window.Result);

UpdateData();

}

}

private void EditUserButton\_Click(object sender, RoutedEventArgs e)

{

if (UsersDataGrid.SelectedItem is not null and User user)

{

if (user.Login == this.user.Login)

{

MessageBox.Show("Вы не можете изменить себя");

}

else

{

var window = new EditUserWindow(user, data);

if (window.ShowDialog() == true)

{

data.Users.Update(user);

UpdateData();

}

}

}

else

{

MessageBox.Show("Сначала сделайте выбор");

}

}

private void DeleteUserButton\_Click(object sender, RoutedEventArgs e)

{

if (UsersDataGrid.SelectedItem is not null and User user)

{

if (user.Login == this.user.Login)

{

MessageBox.Show("Вы не можете удалить себя");

}

else

{

data.Users.Remove(user.Id);

UpdateData();

}

}

else

{

MessageBox.Show("Сначала сделайте выбор");

}

}

}

}

**EditUserWindow.xaml.cs**

using Factory.Data;

using Factory.Data.Models;

using System.Collections.Generic;

using System.Windows;

namespace Factory.AdminWindows

{

/// <summary>

/// Логика взаимодействия для EditUserWindow.xaml

/// </summary>

public partial class EditUserWindow : Window

{

public User Result { get; set; }

private readonly DataLayer data;

private readonly Dictionary<string, int> accesses = new()

{

["Администратор"] = 1,

["Пользователь"] = 3,

["Менеджер"] = 2,

};

public EditUserWindow(User user, DataLayer data)

{

this.data = data;

InitializeComponent();

LevelTextBox.ItemsSource = new List<string>()

{

"Администратор",

"Пользователь",

"Менеджер"

};

LevelTextBox.Text = user.Access;

LoginTextBox.Text = user.Login;

PasswordTextBox.Text = user.Password;

Result = user;

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

Result.Password = PasswordTextBox.Text;

Result.Level = accesses[LevelTextBox.Text];

var validRes = Validator.Valid(Result, data.Users, true);

if (validRes != null)

{

MessageBox.Show(validRes);

return;

}

DialogResult = true;

}

}

}

**Folder GuestWindows**

**GuestWindow.xaml.cs**

using Factory.Data;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

namespace Factory.GuestWindows

{

/// <summary>

/// Логика взаимодействия для GuestWindow.xaml

/// </summary>

public partial class GuestWindow : Window

{

private readonly IEnumerable<ProductItem> products;

private class ProductItem

{

public string Name { get; set; }

public string Materials { get; set; }

}

public GuestWindow(DataLayer data)

{

products = data.Products.GetAll().Select(x => new ProductItem()

{

Name = x.Name,

Materials = string.Join(", ", x.Materials.Select(m => m.Material.Name))

});

InitializeComponent();

RefreshSearch();

}

private void RefreshSearch(string name = null)

{

if (name != null)

{

ProductDataGrid.ItemsSource = products.Where(x => (x.Name == name) || x.Name.ToLower().Contains(name.Trim().ToLower()));

}

else

{

ProductDataGrid.ItemsSource = products;

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

RefreshSearch(SearchTextBox.Text);

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

RefreshSearch();

}

}

}

**Folder ManagerWindows**

**AddNormWindow.xaml.cs**

using Factory.Data;

using Factory.Data.Models;

using System.Windows;

namespace Factory.ManagerWindows

{

/// <summary>

/// Логика взаимодействия для AddNormWindow.xaml

/// </summary>

public partial class AddNormWindow : Window

{

private readonly DataLayer data;

public Product Product { get; set; }

private Raw norm;

public AddNormWindow(DataLayer data, Product product)

{

this.data = data;

Product = product;

InitializeComponent();

}

private void SearchMaterialButton\_Click(object sender, RoutedEventArgs e)

{

var window = new SearchItemWindow(data.Raws.GetAll());

if (window.ShowDialog() == true)

{

norm = window.Result as Raw;

MaterialTextBox.Text = norm.Name;

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (!double.TryParse(QuantityTextBox.Text, out double res))

{

MessageBox.Show("В строке количество должно быть число");

return;

}

Norm productMaterial = new()

{

Raw = norm,

Quantity = res

};

var validRes = Validator.Valid(productMaterial);

if (validRes is not null)

{

MessageBox.Show(validRes);

return;

}

Product.Norms.Add(productMaterial);

DialogResult = true;

}

}

}

**AddPlanWindow.xaml.cs**

using Factory.Data.Models;

using System;

using System.Linq;

using System.Windows;

namespace Factory.ManagerWindows

{

/// <summary>

/// Логика взаимодействия для AddPlanWindow.xaml

/// </summary>

public partial class AddPlanWindow : Window

{

private readonly Product product;

public AddPlanWindow(Product product)

{

this.product = product;

InitializeComponent();

}

private static DateTime? Parse(string str)

{

string[] values = str.Split('.');

if (values.Length != 3)

{

return null;

}

try

{

var ints = values.Select(x => int.Parse(x)).ToArray();

var date = new DateTime(ints[2], ints[1], ints[0]);

return date;

}

catch (Exception)

{

return null;

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

var date = Parse(DateTextBox.Text);

if (date == null)

{

MessageBox.Show("Дата была в неверном формате");

return;

}

if (!int.TryParse(QuantityTextBox.Text, out int res))

{

MessageBox.Show("Неверный формат количества");

return;

}

ProductPlan plan = new()

{

Date = (DateTime)date,

Count = res

};

var validRes = Validator.Valid(plan, product);

if (validRes is not null)

{

MessageBox.Show(validRes);

return;

}

product.Plans.Add(plan);

DialogResult = true;

}

}

}

**AddProductionWindow.xaml.cs**

using Factory.Data.Models;

using System;

using System.Linq;

using System.Windows;

namespace Factory.ManagerWindows

{

/// <summary>

/// Логика взаимодействия для AddProductionWindow.xaml

/// </summary>

public partial class AddProductionWindow : Window

{

private readonly Product product;

public AddProductionWindow(Product product)

{

this.product = product;

InitializeComponent();

}

private static DateTime? Parse(string str)

{

string[] values = str.Split('.');

if (values.Length != 3)

{

return null;

}

try

{

var ints = values.Select(x => int.Parse(x)).ToArray();

var date = new DateTime(ints[2], ints[1], ints[0]);

return date;

}

catch (Exception)

{

return null;

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

var date = Parse(DateTextBox.Text);

if (date == null)

{

MessageBox.Show("Дата была в неверном формате");

return;

}

if (!int.TryParse(QuantityTextBox.Text, out int res))

{

MessageBox.Show("Неверный формат количества");

return;

}

ProductProduction plan = new()

{

Date = (DateTime)date,

Count = res

};

var validRes = Validator.Valid(plan, product);

if (validRes is not null)

{

MessageBox.Show(validRes);

return;

}

product.Productions.Add(plan);

DialogResult = true;

}

}

}

**AddProductWindow.xaml.cs**

using Factory.Data;

using Factory.Data.Models;

using System.Windows;

namespace Factory.ManagerWindows

{

/// <summary>

/// Логика взаимодействия для AddProductWindow.xaml

/// </summary>

public partial class AddProductMaterialWindow : Window

{

private readonly DataLayer data;

public Product Product { get; set; }

private Material material;

public AddProductMaterialWindow(Product product, DataLayer data)

{

this.data = data;

Product = product;

InitializeComponent();

}

private void SearchMaterialButton\_Click(object sender, RoutedEventArgs e)

{

var window = new SearchItemWindow(data.Materials.GetAll());

if (window.ShowDialog() == true)

{

material = window.Result as Material;

MaterialTextBox.Text = material.Name;

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (!double.TryParse(QuantityTextBox.Text, out double res))

{

MessageBox.Show("В строке количество должно быть число");

return;

}

ProductMaterial productMaterial = new()

{

Material = material,

Quantity = res,

};

var validRes = Validator.Valid(productMaterial);

if (validRes is not null)

{

MessageBox.Show(validRes);

return;

}

Product.Materials.Add(productMaterial);

DialogResult = true;

}

}

}

**InputStringWindow.xaml.cs**

using System.Windows;

namespace Factory.ManagerWindows

{

/// <summary>

/// Логика взаимодействия для InputStringWindow.xaml

/// </summary>

public partial class InputStringWindow : Window

{

public string Result { get; set; }

public InputStringWindow(string fieldName)

{

InitializeComponent();

NameLabel.Content = ((string)NameLabel.Content) + fieldName;

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

Result = ValueTextBox.Text;

DialogResult = true;

}

}

}

**ManagerWindow.xaml.cs**

using Factory.Data;

using Factory.Data.Models;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

namespace Factory.ManagerWindows

{

/// <summary>

/// Логика взаимодействия для ManagerWindow.xaml

/// </summary>

public partial class ManagerWindow : Window

{

private readonly DataLayer data;

public ManagerWindow(DataLayer data, User user)

{

this.data = data;

InitializeComponent();

Title = user.Login;

UpdateData();

}

private void UpdateData()

{

ProductsDataGrid.ItemsSource = data.Products.GetAll();

MaterialsDataGrid.ItemsSource = data.Materials.GetAll();

RawDataGrid.ItemsSource = data.Raws.GetAll();

ProductProductionDataGrid.ItemsSource = data.Products.GetAll();

ProductPlanDataGrid.ItemsSource = data.Products.GetAll();

}

private void ProductsDataGrid\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

if (ProductsDataGrid.SelectedItem is not null and Product product)

{

new ProductDetailWindow(data, product).ShowDialog();

}

}

private void DeleteCatalogRawWindow\_Click(object sender, RoutedEventArgs e)

{

if (RawDataGrid.SelectedItem is not null and Raw raw)

{

data.Raws.Remove(raw.Id);

UpdateData();

}

}

private void AddCatalogRawWindow\_Click(object sender, RoutedEventArgs e)

{

var name = new InputStringWindow("Наименование");

if (name.ShowDialog() == true)

{

Raw raw = new()

{

Name = name.Result

};

var validRes = Validator.Valid(raw);

if (validRes != null)

{

MessageBox.Show(validRes);

return;

}

data.Raws.Add(raw);

UpdateData();

}

}

private void AddCatalogMaterialWindow\_Click(object sender, RoutedEventArgs e)

{

var name = new InputStringWindow("Наименование");

if (name.ShowDialog() == true)

{

Material raw = new()

{

Name = name.Result

};

var validRes = Validator.Valid(raw);

if (validRes != null)

{

MessageBox.Show(validRes);

return;

}

data.Materials.Add(raw);

UpdateData();

}

}

private void DeleteCatalogMaterialWindow\_Click(object sender, RoutedEventArgs e)

{

if (MaterialsDataGrid.SelectedItem is not null and Material raw)

{

data.Materials.Remove(raw.Id);

UpdateData();

}

}

private void AddCatalogProductWindow\_Click(object sender, RoutedEventArgs e)

{

var name = new InputStringWindow("Наименование");

if (name.ShowDialog() == true)

{

Product raw = new()

{

Name = name.Result

};

var validRes = Validator.Valid(raw);

if (validRes != null)

{

MessageBox.Show(validRes);

return;

}

data.Products.Add(raw);

UpdateData();

}

}

private void DeleteCatalogProductWindow\_Click(object sender, RoutedEventArgs e)

{

if (ProductsDataGrid.SelectedItem is not null and Product raw)

{

data.Products.Remove(raw.Id);

UpdateData();

}

}

private void ProductPlanDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (ProductPlanDataGrid.SelectedItem is not null and Product product)

{

PlansDataGrid.ItemsSource = product.Plans;

}

else

{

PlansDataGrid.ItemsSource = null;

}

}

private void AddPlanProductButton\_Click(object sender, RoutedEventArgs e)

{

if (ProductPlanDataGrid.SelectedItem is not null and Product product)

{

var window = new AddPlanWindow(product);

if (window.ShowDialog() == true)

{

data.Products.Update(product);

PlansDataGrid.ItemsSource = null;

PlansDataGrid.ItemsSource = product.Plans;

}

}

}

private void ProductProductionDataGrid\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (ProductProductionDataGrid.SelectedItem is not null and Product product)

{

ProductionDataGrid.ItemsSource = product.Plans;

}

else

{

ProductionDataGrid.ItemsSource = null;

}

}

private void AddProductionProductButton\_Click(object sender, RoutedEventArgs e)

{

if (ProductProductionDataGrid.SelectedItem is not null and Product product)

{

var window = new AddPlanWindow(product);

if (window.ShowDialog() == true)

{

data.Products.Update(product);

ProductionDataGrid.ItemsSource = null;

ProductionDataGrid.ItemsSource = product.Plans;

}

}

}

}

}

**ProductDetailWindow.xaml.cs**

using Factory.Data;

using Factory.Data.Models;

using System.Windows;

namespace Factory.ManagerWindows

{

/// <summary>

/// Логика взаимодействия для ProductDetailWindow.xaml

/// </summary>

public partial class ProductDetailWindow : Window

{

private readonly Product product;

private DataLayer data;

public ProductDetailWindow(DataLayer data, Product product)

{

this.data = data;

this.product = product;

InitializeComponent();

UpdateData();

}

private void UpdateData()

{

MaterialsDataGrid.ItemsSource = null;

NormsDataGrid.ItemsSource = null;

MaterialsDataGrid.ItemsSource = product.Materials;

NormsDataGrid.ItemsSource = product.Norms;

}

private void AddMaterialWindow\_Click(object sender, RoutedEventArgs e)

{

var window = new AddProductMaterialWindow(product, data);

if (window.ShowDialog() == true)

{

data.Products.Update(product);

UpdateData();

}

}

private void AddNormButton\_Click(object sender, RoutedEventArgs e)

{

var window = new AddNormWindow(data, product);

if (window.ShowDialog() == true)

{

data.Products.Update(product);

UpdateData();

}

}

}

}

**SearchItemWindow.xaml.cs**

using System.Collections.Generic;

using System.Windows;

using System.Windows.Input;

namespace Factory.ManagerWindows

{

/// <summary>

/// Логика взаимодействия для SearchItemWindow.xaml

/// </summary>

public partial class SearchItemWindow : Window

{

public object Result { get; set; }

public SearchItemWindow(IEnumerable<object> values)

{

InitializeComponent();

ItemsDataGrid.ItemsSource = values;

}

private void ItemsDataGrid\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

if (ItemsDataGrid.SelectedItem is not null)

{

Result = ItemsDataGrid.SelectedItem;

DialogResult = true;

}

}

}

}

**Folder UserWindows**

**ProductDetailsWindow.xaml.cs**

using Factory.Data.Models;

using System.Windows;

namespace Factory.UserWindows

{

/// <summary>

/// Логика взаимодействия для ProductDetailsWindow.xaml

/// </summary>

public partial class ProductDetailsWindow : Window

{

public ProductDetailsWindow(Product product)

{

InitializeComponent();

Title = product.Name;

MaterialsDataGrid.ItemsSource = product.Materials;

NormsDataGrid.ItemsSource = product.Norms;

}

}

}

**UserWindow.xaml.cs**

using Factory.Data;

using Factory.Data.Models;

using System.Windows;

using System.Windows.Input;

namespace Factory.UserWindows

{

/// <summary>

/// Логика взаимодействия для UserWindow.xaml

/// </summary>

public partial class UserWindow : Window

{

public UserWindow(DataLayer data, User user)

{

InitializeComponent();

Title = user.Login;

ProductsDataGrid.ItemsSource = data.Products.GetAll();

MaterialsDataGrid.ItemsSource = data.Materials.GetAll();

RawDataGrid.ItemsSource = data.Raws.GetAll();

}

private void ProductsDataGrid\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

if (ProductsDataGrid.SelectedItem is not null and Product product)

{

new ProductDetailsWindow(product).ShowDialog();

}

}

}

}

**MainWindow.xaml.cs**

using Factory.Data;

using Factory.Data.Models;

using System;

using System.Windows;

namespace Factory

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

public DataLayer GetDb()

{

return DataLayer.Sql();

}

private void AuthButton\_Click(object sender, RoutedEventArgs e)

{

var user = Validator.Auth(GetDb(), LoginTextBox.Text, PasswordTextBox.Text);

if (user is not null)

{

Window window = null;

switch (user.Level)

{

case 1:

window = new AdminWindows.AdminWindow(user, GetDb());

break;

case 3:

window = new UserWindows.UserWindow(GetDb(), user);

break;

case 2:

window = new ManagerWindows.ManagerWindow(GetDb(), user);

break;

default:

return;

}

InvokeWindow(window);

}

else

{

MessageBox.Show("Неверный логин или пароль");

}

}

private void GuestButton\_Click(object sender, RoutedEventArgs e)

{

var window = new GuestWindows.GuestWindow(GetDb());

InvokeWindow(window);

}

private void InvokeWindow(Window window)

{

window.Closed += Window\_Closed;

Hide();

window.Show();

}

private void Window\_Closed(object sender, EventArgs e)

{

PasswordTextBox.Text = "";

LoginTextBox.Text = "";

Show();

}

private void Statictic\_Click(object sender, RoutedEventArgs e)

{

var window = new Statistics();

InvokeWindow(window);

}

}

}

**Statistics.xaml.cs**

using Factory.Data;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using Factory.Data.Models;

namespace Factory

{

public partial class Statistics : Window

{

private readonly DataLayer data;

private IEnumerable<ProductItem> products;

public Statistics()

{

data = GetDb();

InitializeComponent();

}

public DataLayer GetDb()

{

return DataLayer.Sql();

}

private class ProductItem

{

public string Name { get; set; }

public int ID { get; set; }

public List<ProductPlan> Plans { get; set; }

}

private void DataGrid\_Loaded(object sender, RoutedEventArgs e)

{

var titleOfProducts = data.Products.GetAll();

}

private void Chart\_Loaded(object sender, RoutedEventArgs e)

{

products = data.Products.GetAll().Select(x => new ProductItem()

{

Name = x.Name,

ID=x.Id,

Plans=x.Plans

});

var titleOfProducts = products.ToList();

string[] test = new string[titleOfProducts.Count];

double[] count = new double[titleOfProducts.Count];

double[] positions = new double[titleOfProducts.Count];

for (int i = 0; i < test.Length; i++)

{

test[i] = titleOfProducts[i].Name;

positions[i] = i;

if (titleOfProducts.First(x=>x.Name==test[i]).Plans.Count != 0)

{

count[i] = titleOfProducts[i].Plans[0].Count;

}

}

double[] values = count;

string[] labels = test;

Chart.Plot.AddBar(values, positions);

Chart.Plot.XTicks(positions, labels);

Chart.Plot.SetAxisLimits(yMin: 0);

Chart.Plot.SaveFig("bar\_labels.png");

Chart.Refresh();

}

}

}