|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Утверждаю | | | | | | | | |
| Зав. каф. | | | | | САУ | | | |
|  | | | |  | наименование | | | |
|  | | | |  | Ланских Ю.В. | | | |
| подпись | | | | | Ф.И.О. | | | |
| « |  | » |  | | | 20 | 22 | г. |

ЗАДАНИЕ

на курсовую работу

|  |  |
| --- | --- |
| по дисциплине | Объектно-ориентированное программирование |
|  | полное название дисциплины |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студентам | Овчарук А.И. | | | , обучающимся по образовательной |
| программе | 09.03.02 – Информационные системы и технологии | | | |
| полное название направления подготовки (специальности) | | | | |
| второй | |  | очная | |
| курс обучения | |  | форма обучения | |

|  |
| --- |
| Задание на курсовую работу: |
| 1. Выполнить анализ заданной предметной области и выявить имеющиеся в ней классы. Определить свойства и методы выявленных классов. Обнаружить существующие между выявленными классами отношения. Построить объектную модель предметной области. |
| 1. Реализовать полученную модель на языке программирования C# для платформы .Net Framework или .Net Core. |
| 1. Используя библиотеку Entity Framework, сгенерировать из полученного набора классов базу данных с использованием СУБД Microsoft SQL Server. |
| 1. Разработать приложение, реализующее задачи из заданной предметной области и использующее созданные классы и базу данных. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. Исходные данные: | Предметная область для курсовой работы: «ИС | | | |
| производства одежды» | | | | |
|  | | | | |
| 2. Основные разделы: |  | | | |
| 1. Построение объектной модели предметной области | | | | |
| 1. Реализация классов и базы данных | | | | |
| 1. Разработка прикладного приложения | | | | |
| 3. График выполнения: |  | | | |
| Разработка модели предметной области: | | 15.10.2021 г. | | |
| Реализация классов и базы данных: г. | | 15.11.2021 | | |
| Разработка прикладного приложения: | | 15.12.2021 г. | | |
| Подготовка пояснительной записки: г. | | 26.01.2022 | | |
| Представить выполненную курсовую работу на проверку не позднее: | | |  | 28.01.2022 |
|  | | |  | Дата |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Руководитель работы |  |  |  | Кузьминых М. М. |  |  |
|  |  | Подпись руководителя |  | Ф.И.О. руководителя |  | Дата |
| Задание принял |  |  |  |  |  |  |
|  |  | Подпись обучающегося |  | Ф.И.О. обучающегося |  | Дата |

Реферат

Овчарук А.И.Реализация информационной системы производства одежды: ТПЖА 090302.951 ПЗ, ТПЖА 090302.783 ПЗ: Отчет по курсовой работе / ВятГУ, каф. САУ; рук. М.М. Кузьминых. – Киров, 2022. ПЗ 39 с., 21 рис., 1 табл., 4 источников, 6 прил.

ПЕРЕЧЕНЬ КЛЮЧЕВЫХ СЛОВ: КЛАСС, ОТНОШЕНИЯ МЕЖДУ КЛАССАМИ, C#, ОБЪЕКТНО-ОРИЕНТИРОВННОЕ ПРОГРАММИРОВАНИЕ, БАЗА ДАННЫХ.

Объектом исследования является высокоуровневый язык программирования C#, разработка с использованием современных информационных технологий.

Цель курсовой работы – закрепить теоретические знания, полученные в ходе освоения курса и приобрести практические навыки проектирования и программирования с использованием объектно-ориентированного подхода.

Разработано приложение для производства одежды по принципу ООП, с помощью которого сотрудник сможет контролировать ход работы на производстве, просмотреть свои данные, и если новый сотрудник, то зарегистрироваться.

|  |  |
| --- | --- |
| **Содержание**  Изм.  Лист  № докум.  Подпись  Дата  Лист  1  ТПЖА 090302.951 ПЗ ТПЖА 090302.783 ПЗ  Разраб.  *Овчарук А.И. Балахнина Е.П.*  Провер.  Кузьминых М.М.  Реценз.  Н. Контр.  Утверд.  Литер  Листов  2  *Кафедра САУ,*  *гр. ИТб-2301-01-00* |  |
| Введение……………………………………………………………………….. | 3 |
| 1. Обзор использованной литературы……………………………………….. | 5 |
| 1. Анализ предметной области.....................................................…………… | 5 |
| 2.1 Процессы, протекающие в предметной области …………….............. | 5 |
| 2.2 Классы предметной области...................……………………………… | 5 |
| 1. Реализация классов на языке программирования C#..................……........ | 7 |
| 3.1 Отношение между классами...........................................……………… | 7 |
|  |  |
| 1. Создание базы данных с использованием Entity Framework..................... | 8 |
| 4.1 Создание контекста для базы данных………...…………………… | 8 |
|  |  |
| 4.2 Создание БД................................................................……………... | 8 |
| 1. Разработка приложения, реализующего базовую функциональность. | 9 |
| 5.1 Вход в приложение………...…………………..............................…..... | 9 |
| 5.2 Форма директора...................................……...………………………… | 9 |
| 5.2.1 Форма информации о сотрудниках……..……………………… | 10 |
| 5.2.1.1 Форма редактирования информации о сотруднике…. | 11 |
| 5.2.2 Форма добавления цвета ……….................................................. | 11 |
| 5.2.3 Форма добавления материала ...................................................... | 12 |
| 5.2.4 Форма производства....................................................................... | 13 |
| 5.2.4.1 Отдел художественного эскиза………………………… | 14 |
| 5.2.4.2 Отдел технического эскиза……………………………... | 15 |
| 5.2.4.3 Отдел раскройки………………………………………… | 16 |
| 5.2.4.4 Отдел пошива……………………………………………. | 17 |
| 5.2.4.5 Контроль качества………………………………………. | 17 |
|  |  |
| 5.3 Форма дизайнера..............................……...…………………….......... | 18 |
| 5.4 Форма раскройщика и швеи…………………………………………. | 19 |
| Заключение………………………………………………………………..…… | 20 |
| Приложение А (обязательное) Ключевые классы……………………...........  Изм.  Лист  № докум.  Подпись  Дата  Лист  2  Разраб.  Провер.  Кузьминых М.М.  Реценз.  Н. Контр.  Утверд.  Литер  Листов  2 | 21 |
| Приложение Б (обязательное) Вид отношения ассоциация.......................… | 23 |
| Приложение В (обязательное) Вид отношения композиция......................... | 24 |
| Приложение Г (обязательное) Вид отношения агрегация............................. | 25 |
| Приложение Д (обязательное) Контекст БД………………………………… | 26 |
| Приложение Е (обязательное) Листинг программного кода реализации работы с библиотекой классов.......................................................................... | 27 |
| Приложение Ж (справочное) Библиографический список……………..... | 39 |

Реализация информационной системы производства одежды

*Овчарук А.И..*

# Введение

Объектно-ориентированное программирование (ООП) — это методология программирования, которая основана на представлении программы в виде совокупности ряда объектов, причём каждый из этих объектов — это экземпляр определённого класса. ООП — это шаблон проектирования ПО, позволяющий решать задачи разработчика с точки зрения взаимодействия объектов.

Цель курсовой работы по дисциплине «Объектно-ориентированное программирование» - закрепить теоретические знания, полученные в ходе освоения курса и приобрести практические навыки проектирования и программирования с использованием объектно-ориентированного подхода.

Задачи, которые должны быть решены в рамках написания курсовой работы:

**·** Овладение основами объектно-ориентированного анализа предметной области и выявления классов и отношений между ними;

**·** Освоение приемов объектно-ориентированного программирования в ходе реализации выявленных классов на выбранном языке программирования;

**·** Знакомство с библиотечными классами выбранной платформы программирования;

**·** Знакомство с приемами работы с базами данных с использованием библиотек, реализующих объектно-реляционное отображение (ORM).

Основная целевая платформа для разработки – Microsoft .Net Framework, язык программирования – C#, библиотека ORM – Entity Framework. Допускается выбор другого объектно-ориентированного языка и платформы, но он должен быть согласован с преподавателем.

# **Обзор использованной литературы**

Главной целью данного обзора является изучение инструментов для разработки информационной системы с помощью библиотеки Entity Framework.

Для работы необходимо было овладеть основами объектно-ориентированного анализа предметной области и выявить классы и отношения между ними, изучить технологию программирования на базе фреймворка .NET – Entity Framework для работы с реляционными данными. Для этого обратимся к сайтам в интернете [1-2, приложение И]. Данные статьи демонстрируют основные принципы работы с данными при использовании Entity Framework.

Также необходимо изучить виды отношений между классами и объектами. Для этого обратимся к статье электронного ресурса [3-4, приложение К]. Из статьи можно узнать о трех основных видах связей: ассоциация, композиция и агрегация. А также рекомендации по их применению.

# Анализ предметной области

Производство одежды – отрасль лёгкой промышленности, разрабатывающая текстильные изделия. Оно не занимается никакой коммерческой деятельностью, а только осуществляет изготовление продукции.

На производстве есть сотрудники, за каждым из которых закреплена особая часть работы над товаром, соответствующая его должности. Производство должно выполнять следующие функции:

1. Регистрация сотрудников.
2. Добавление нового изделия.
3. Возможность отслеживания этапов разработки изделия.
4. Возможность редактирования информации об изделии.
5. Добавление и редактирование информации об используемом материале.
6. Добавление и редактирование информации о цветах изделия.
7. Привилегированный доступ к информации.
8. Удаление неактуальной продукции.

Технологический процесс предприятия предусматривает последовательность выполнения шагов на различных стадиях изготовления заказанной продукции.

* 1. Процессы, протекающие в предметной области

Основным процессом является предоставление информации об этапах разработки одежды. Для реализации данного процесса в первую очередь директору необходимо быть зарегистрированным и иметь доступ во все разделы приложения. После чего он может добавить информацию о сотрудниках, их должностях, используемом в работе материале и цвет продукции.

Все пользователи могут добавлять, изменять и удалять информацию об актуальных разработках производства. Немаловажным фактором является ограничение доступа для некоторых должностей в соответствующие разделы приложения. Это сделано для избежания незапланированного удаления информации и конфиденциальности сотрудников.

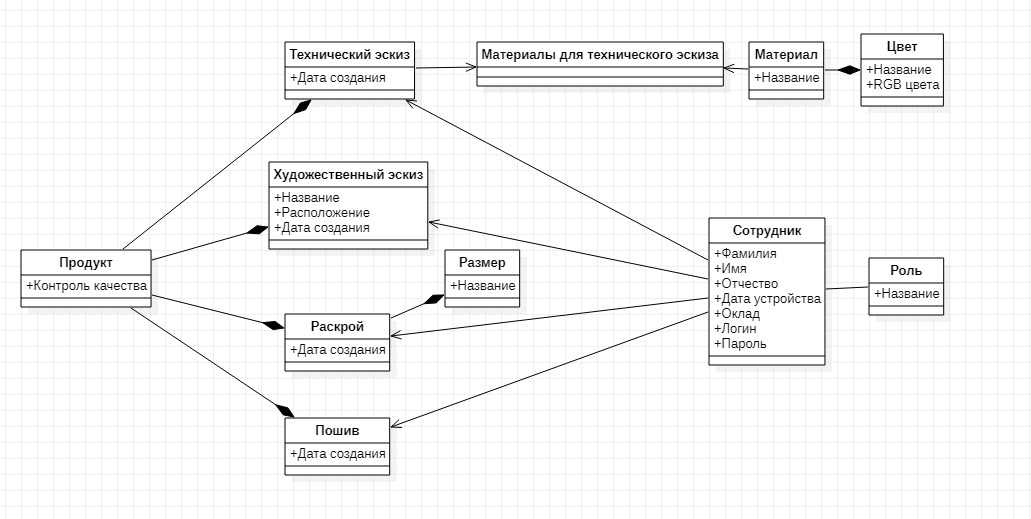
* 1. Классы предметной области

Выявленные классы предметной области указаны в таблице 1.

Таблица 1 – классы предметной области

|  |  |  |
| --- | --- | --- |
| № п/п | Категория | Классы |
| 1 | Физические или материальные объекты | Товар, материал |
| 2 | Спецификации, описания объектов | Характеристики товара |
| 3 | Места | Производство |
| 4 | Транзакции | Удаление готового изделия |
| 5 | Элементы транзакций | – |
| 6 | Роли людей | Директор, швея, раскройщик, дизайнер |
| 7 | Контейнеры других объектов | Размер, цвет |
| 8 | Содержимое контейнеров | Единицы измерения |
| 9 | Организации | Производство |
| 10 | События | Разработка товара |
| 11 | Процессы | Художественный эскиз, технический эскиз, раскрой, пошив, контроль качества |
| 12 | Каталоги | Справочник единиц измерения, должность, размер и цвет |
| 13 | Записи о деятельности | Стадия разработки, редактирование, удаление |

Выявленные классы, их отношения и свойства, отраженные на диаграмме классов UML, изображены на рисунке 1.

Рисунок 1 – Диаграмма классов UML

В данном разделе проанализирована предметная область, определены протекающие в ней процессы. Выявлены классы при помощи метода анализа по категориям. Создана диаграмма классов UML, отображающая выявленные классы с атрибутами и связями между ними.

# Реализация классов на языке программирования C#

В предметной области был выделен ряд ключевых классов: Employee, Material, ModelColor, Sketch, Blueprint, Cut, Sewing, Product. Код ключевых классов представлен в приложении А. Для всех классов реализованы конструкторы, задающие корректное начальное состояние объектов.

* 1. **Отношения между классами**

Ассоциация — это отношение, при котором объекты одного типа неким образом связаны с объектами другого типа. Между классами Employee и Sewing реализован вид отношения ассоциация. Данное отношение продемонстрировано в приложении Б.

Композиция — разновидность отношения агрегации, при которой составные части целого имеют такое же время жизни, что и само целое. Между классами Product и Sketch реализован вид отношения композиция. Реализация данных классов представлена в приложении В. Композиция определяет отношение "имеет". Класс заявка содержит объект класса контракт.

Агрегация – разновидность ассоциации, представляющая структурную связь целого с его частями. Как тип ассоциации, агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов. Между классами ModelColor и Material реализован вид отношения агрегация.

Таким образом, в 3 разделе представлена реализация выявленных классов и их конструкторов. Осуществлены описанные отношения между классами предметной области.

# Создание базы данных с использованием Entity Framework

Работа написана с помощью библиотеки ORM(Entity Framework). Entity Framework — это набор технологий в ADO.NET, которые поддерживают разработку программных приложений, ориентированных на данные. Архитекторам и разработчикам приложений приходится учитывать необходимость достижения двух совершенно различных целей. Они должны моделировать сущности, связи и логику решаемых бизнес-задач, а также работать с ядрами СУБД. Например, работая с базами данных напрямую, разработчик должен беспокоиться о подключении, подготовке SQL и параметров, отправке запросов и транзакций. На Entity Framework всё это делается автоматически — программист же работает непосредственно с сущностями и только говорит EF, что нужно сохранить изменения.

* 1. Создание контекста для базы данных

Для взаимодействия с базой данных был создан класс AppDbContext. Он является наследником DbContext. Код данного типа предоставлен в приложении Г.

Для определения структуры БД используются аннотации: Required, MinLength. Перечисленные аннотации используются во многих классах, в том числе в уже представленных в приложении А.

4.2 Создание БД

Диаграмма базы данных предоставлена на рисунке 2.

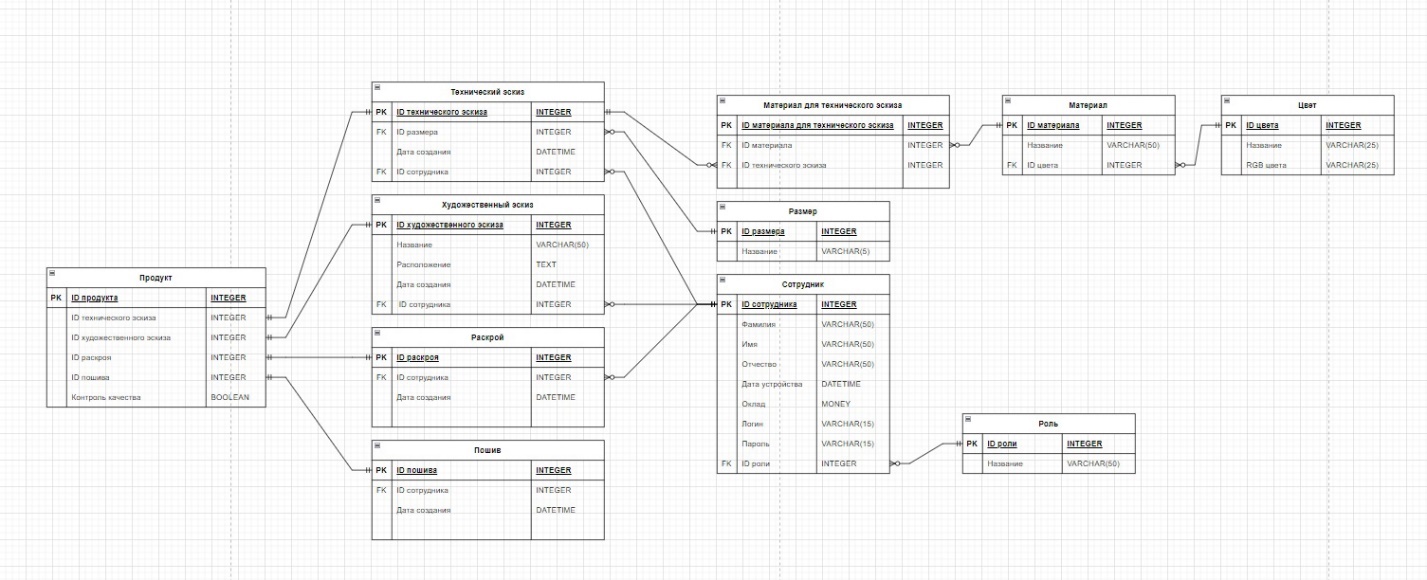


Рисунок 2 – Схема базы данных

На схемы отображены классы с их атрибутами, типами данных и ограничениями целостности. Между классами расставлены связи.

В данном разделе был создан контекст для работы с базой данных.

# Разработка приложения, реализующего базовую функциональность

В данном разделе приведена демонстрация работы программы. Ключевые фрагменты кода, реализующие работу с созданными классами представлены в приложении К.

В приложении разработан пользовательский индивидуальный интерфейс, для каждой из ролей: директор, дизайнер, швея и раскройщик. Интерфейс для данных ролей разработан с учетом задач, которые должны решаться в выбранной предметной области для конкретной роли.

* 1. **Вход в приложение**

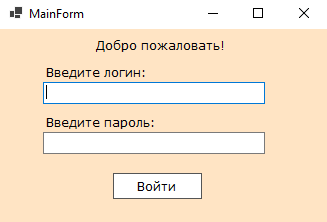
Форма для входа в приложение под определенной ролью представлена на рисунке 3.

Рисунок 3 – Вход в приложение

Каждый сотрудник имеет свой уникальный логин и пароль для входа в приложение.

**5.2 Форма директора**

На форме представлены 4 основные навигационные кнопки: сотрудники, производство, редактирование материала и цвета. Возможности директора: добавление нового сотрудника в базу данных, редактирование и удаление информации о сотрудниках. Также он может добавлять, редактировать, удалять информацию о материалах и цвете изделия, контролировать этапы разработки создания продукции. Реализация формы для директора представлена на рисунке 4.

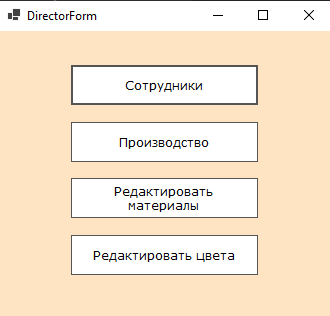


Рисунок 4 – Форма директора

**5.2.1 Форма информации о сотрудниках**

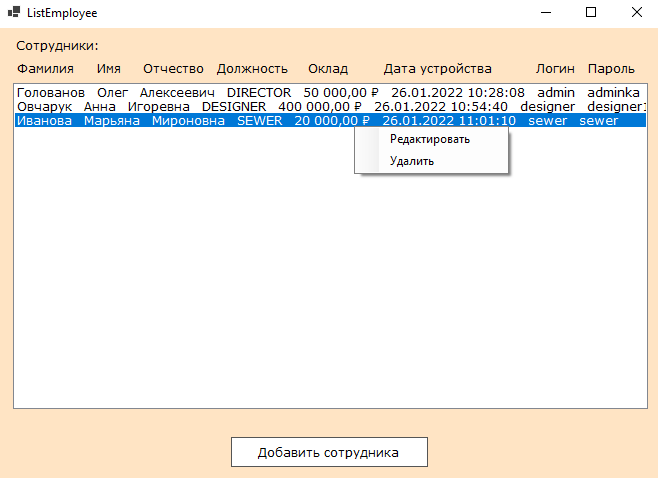
На данной форме представлен список сотрудников, их ФИО, оклад, должность, дата устройства пароль и логин. Здесь можно добавить нового сотрудника, а при нажатии правой кнопки мышки появляется меню редактирования и удаления информацию. Данная форма представлена на рисунке 5.

Рисунок 5 – Форма информации о сотрудниках

**5.2.1.1 Форма редактирования информации о сотруднике**

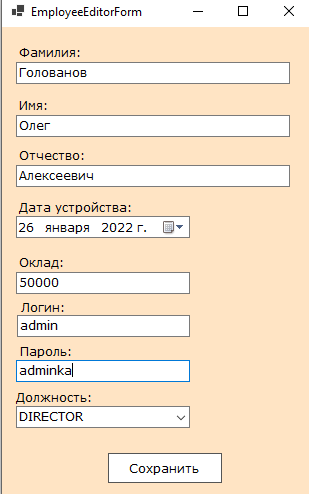
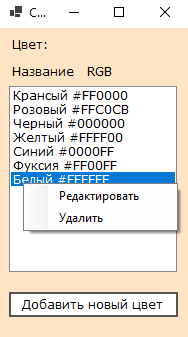
Форма поваляется при первом использовании приложения. На этом этапе необходимо внести информацию о директоре, после чего данное окно можно будет вызвать только при редактировании уже существующего сотрудника или через ответствующую кнопку на форме информации о сотрудниках. Форма добавления и редактирования сотрудника изображена на рисунке 6.

Рисунок 6 – Форма добавления и редактирования информации о сотруднике

**5.2.2 Форма добавления цвета**

Форма цвета создана для удобства в использовании. Т.к. продукция состоит из разного материала, цвет конечного изделия будет смешанным. Именно для уточнения информации о товаре добавлена рассматриваемая форма. На окне можно увидеть список уже созданных цветов и добавить новый. Элементы из списка можно редактировать и удалять. Реализация представлена на рисунке 7.

При нажатии на кнопку добавления нового цвета или при редактировании уже имеющегося цвета поваляется форма, в которую необходимо ввести текстовое и RGB название. Реализация представлена на рисунке 8.

Рисунок 7 – Форма цвета

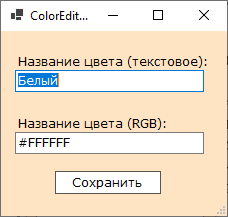


Рисунок 8 – Форма добавления цвета

**5.2.3 Форма добавления материала**

Даннаяформа предназначена для просмотра материала. Предусмотрена возможность добавления, редактирования и удаления информации. Реализация представлена на рисунке 9.

При нажатии на кнопку добавления нового материала или при редактировании уже имеющегося поваляется форма, в которую необходимо ввести название и цвет материала. Реализация представлена на рисунке 10.

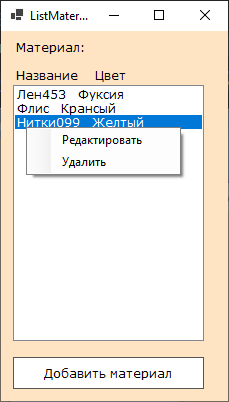
****

Рисунок 9 – Форма материала

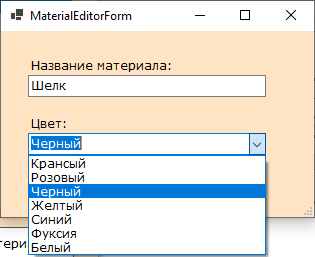


Рисунок 10 – Форма добавления и редактирования материала

Стоит отметить, что цвет материала задается уже из готового списка цветов, которые были внесены ранее.

**5.2.4 Форма производства**

Форма предназначена для контроля этапов разработки продукции. На ней можно увидеть список всех изделий и их уровень готовности. Интерфейс данной формы представлен на рисунке 11.

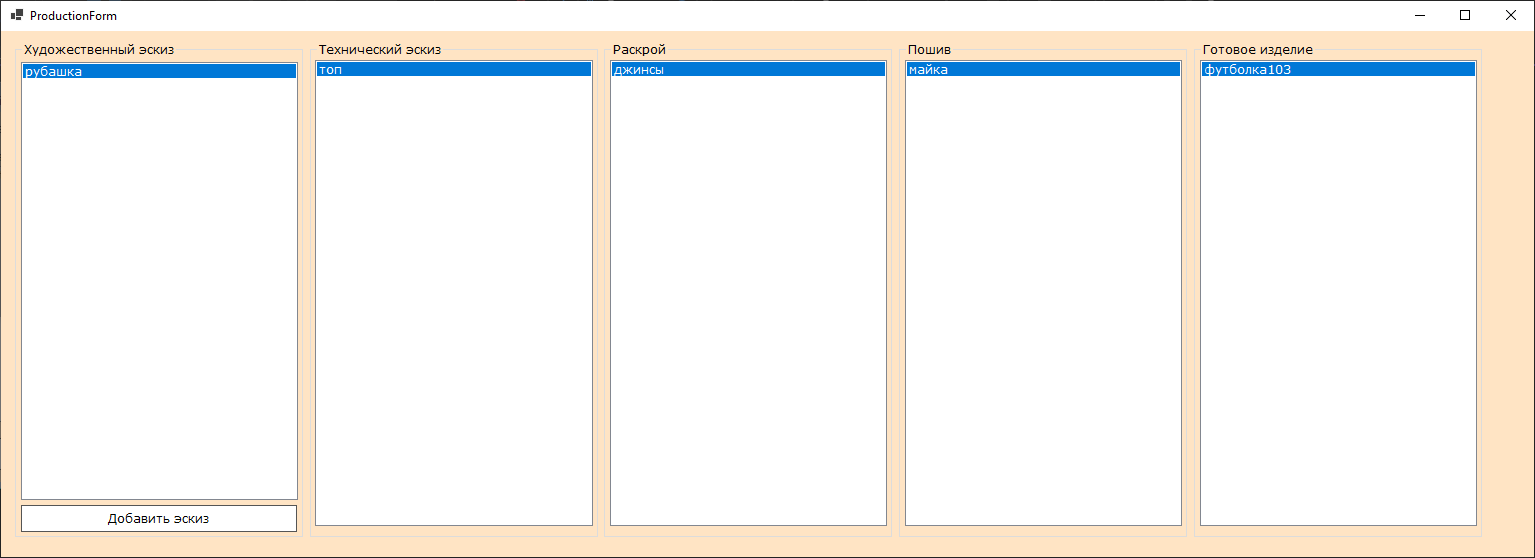


Рисунок 11 – Форма производства

На рисунке 11 видно, что форма разделена на такие части, как художественный эскиз, технический эскиз, раскрой, пошив и готовое изделие. Каждая из частей содержит в себе информацию о продукте. Информацию можно добавлять, редактировать и изменять.

**5.2.4.1 Отдел художественного эскиза**

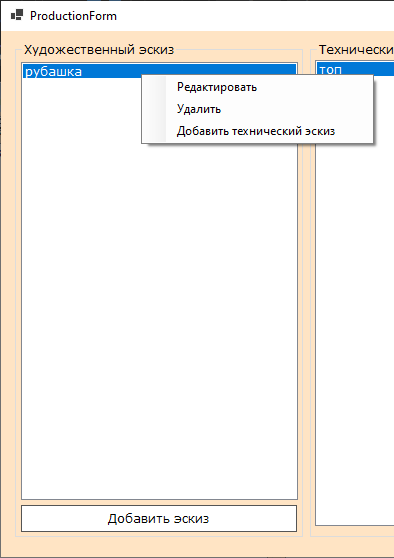
****Рассмотрим отделение «Художественный эскиз» - рисунок 12.

Рисунок 12 – Художественный эскиз

Здесь можно добавить новый эскиз, редактировать, удалить и переместить в следующий раздел. При нажатии на кнопку добавления эскиза или при редактировании уже имеющегося поваляется форма, в которую необходимо ввести название, расположение, автора и дату создания. Реализация представлена на рисунке 13.

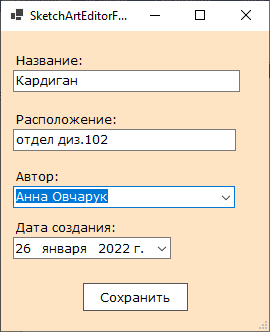


Рисунок 13 – Добавление эскиза

**5.2.4.2 Отдел технического эскиза**

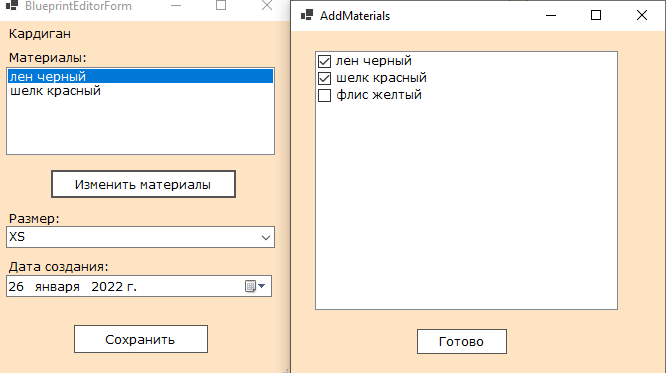
**** Далее изделие попадает в отдел технического эскиза. Для этого необходимо нажать правой кнопкой мыши на изделие и в сплывающем меню выбрать «Добавить технический эскиз». Появляется форма для добавления материла, размера и даты создания. Реализация представлена на рисунке 14.

Рисунок 14 – Переход к техническому эскизу

В разделе «Технический эскиз» можно редактировать, удалить и отправить изделие на раскрой – рисунок 15.

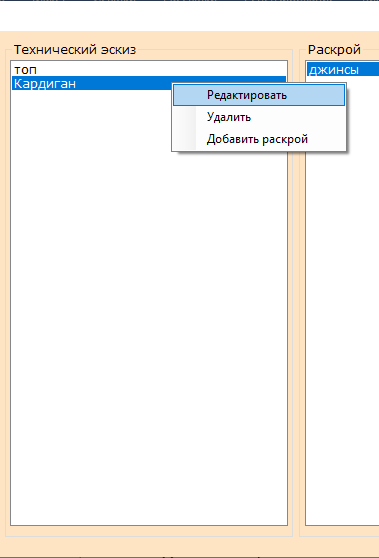


Рисунок 15 – Технический эскиз

**5.2.4.3 Отдел раскройки**

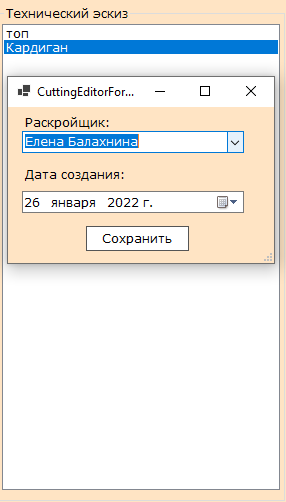
После технического эскиза продукция проходит этап раскройки. Для этого необходимо нажать правой кнопкой мыши на изделие и в сплывающем меню выбрать «Добавить раскрой». Появляется форма для добавления информации о сотруднике и даты создания. Реализация представлена на рисунке 16.

Рисунок 16 – Переход к раскрою изделия

В разделе «Раскрой» сохраняется функция редактирования и удаления неактуальной информации.

**5.2.4.4 Отдел пошива**

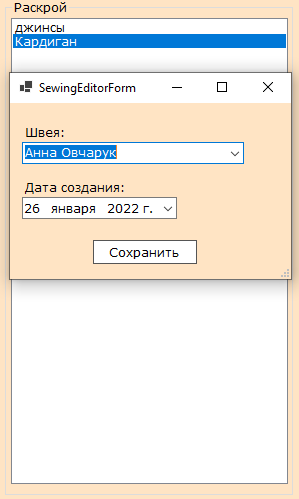
 Пройдя все вышеперечисленные этапы, изделие отправляется на пошив. Для перемещения в следующий раздел необходимо нажать правой кнопкой мыши на изделие и в сплывающем меню выбрать «Добавить пошив». Появляется форма для добавления информации о сотруднике и даты создания. Реализация представлена на рисунке 17.

Рисунок 17 – Переход к пошиву изделия

**5.2.4.5 Контроль качества**

В рассматриваемом разделе, как и в предыдущих, есть возможность редактирования и удаления неактуальной информации. После пошива продукт обязан пройти контроль качества, в результате которого изделие отправляется в соответствующий отдел. Для этого необходимо нажать правой кнопкой мыши на изделие и в сплывающем меню выбрать «Контроль качества». Если изделие с незначительным браком, то оно возвращается на производство. Если брак невозможно исправить, то изделие утилизируется. В лучшем случае, продукция проходит контроль качества и переходит в раздел готовой продукции. Реализация представлена на рисунке 18.

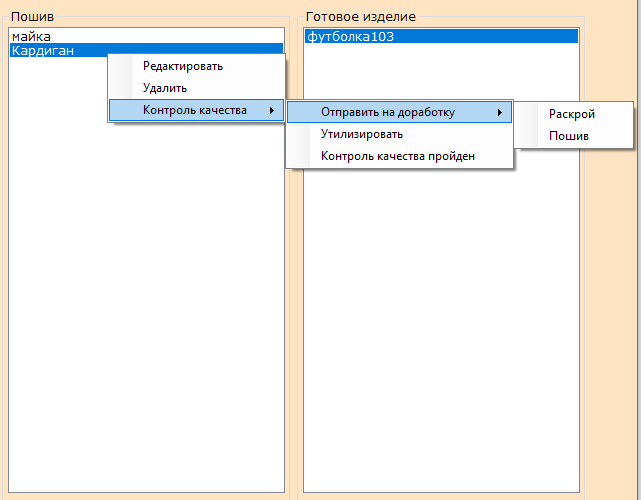


Рисунок 18 – Контроль качества

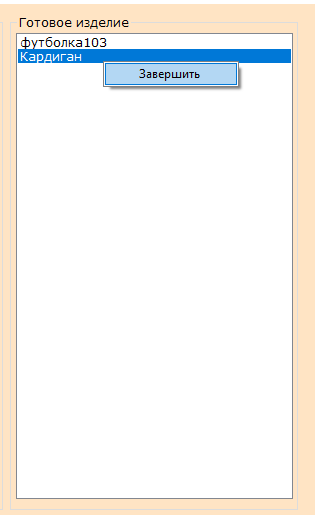
В последнем отделе сохраняется список всех изделий. Нажав кнопку «завершить» изделие удаляется из списка – рисунок 19.

Рисунок 19 – Готовое изделие

* 1. **Форма дизайнера**

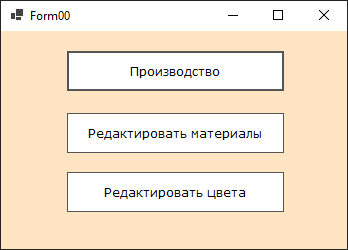
Так как дизайнеру не обязательно знать информацию о других сотрудниках, на его первой форме после авторизации отсутствует кнопка «Сотрудники». Он не может добавлять, изменять и удалять элементы из списка сотрудников. Реализация представлена на рисунке 20.

Рисунок 20 – Форма дизайнера

К остальным разделам приложения у него полный доступ, как и у директора.

* 1. **Форма раскройщика и швеи**

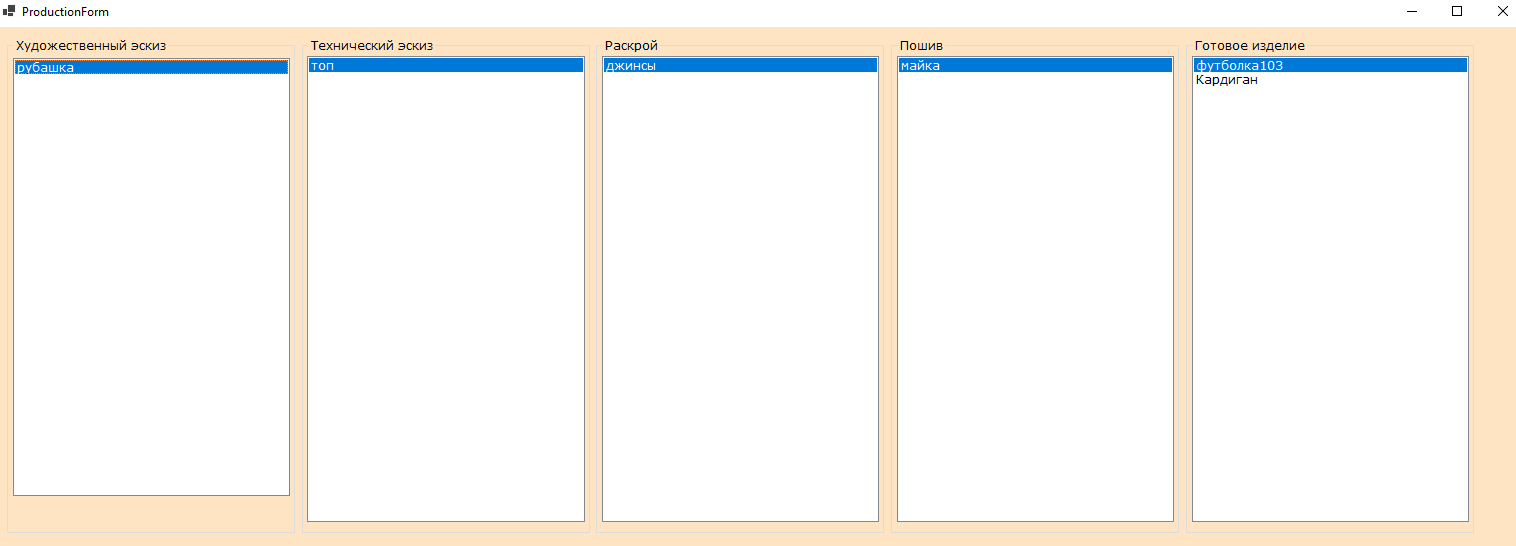
****Раскройщику, как швее, нет необходимости вносить изменения в такие разделы, как сотрудники, добавления материала и цвета. Люди на данных должностях уже работают по готовому эскизу и выбранному материалу. На их форме после авторизации можно увидеть только этапы производства изделия. Раскройщик и швея не могут вносить изменения в какой-либо отдел. Реализация формы представлена на рисунке 21.

Рисунок 21 – Форма раскройщика и швеи

В данном разделе представлена реализация ключевых фрагментов кода, реализующих работу с библиотекой классов. Также были проиллюстрированы и подробно описаны все окна и страницы пользовательского приложения для всех ролей. Описаны возможности приложения для реализации перечня задач для каждой роли.

**Заключение**

В ходе выполнения курсовой работы был выполнен анализ заданной предметной области и выявлены имеющиеся в ней классы. Определены свойства и методы выявленных классов. Обнаружены существующие между выявленными классами отношения. Построена объектная модель предметной области. Реализовали полученную модель на языке программирования C# для платформы .Net Framework или .Net Core. Используя библиотеку Entity Framework, сгенерировали из полученного набора классов базу данных с использованием СУБД Microsoft SQL Server. Разработали приложение, реализующее задачи из заданной предметной области и использующее созданные классы и базу данных.

**Приложение А**

(обязательное)

**Ключевые классы**

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace ProjectOop

{

namespace Entities // сущности

{

public class Employee // сотрудник

{

public int ID { get; set; }

[Required] [MaxLength(50)] public string Surname { get; set; }

[Required] [MaxLength(50)] public string Name { get; set; }

[MaxLength(50)] public string Patronymic { get; set; }

[Required] public DateTime DeviceDate { get; set; }

[Required] public string Salary { get; set; }

[Required] [MaxLength(15)] public string Login { get; set; }

[Required] [MaxLength(15)] public string Password { get; set; }

[Required] public Role Role { get; set; }

}

public class Material // материал

{

public int ID { get; set; }

[Required] public string Name { get; set; }

public ModelColor Сolor { get; set; }

public ICollection<Blueprint> RelatedBlueprints { get; set; }

}

public class ModelColor

{

public int ID { get; set; }

[Required] public string RgbValue { get; set; }

[Required] public string TextName { get; set; }

}

public class Sketch// художественный эскиз

{

public int ID { get; set; }

[Required] public string Name { get; set; }

[Required] public string FileLocation { get; set; }

[Required] public Employee Author { get; set; }

[Required] public DateTime CreationDate { get; set; }

}

public class Blueprint// технический эскиз

{

public int ID { get; set; }

[Required] public WearSize Size { get; set; }

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

[Required] public DateTime CreationDate { get; set; }

}

public class Cut // раскрой

{

public int ID { get; set; }

[Required] public Employee Author { get; set; }

[Required] public DateTime CreationDate { get; set; }

}

public class Sewing // пошив

{

public int ID { get; set; }

[Required] public Employee Author { get; set; }

[Required] public DateTime CreationDate { get; set; }

}

public class Product // продукт

{

public int ID { get; set; }

public Sketch Sketch { get; set; }

public Blueprint Blueprint { get; set; }

public Cut Cut { get; set; }

public Sewing Sewing { get; set; }

public bool QaPassed { get; set; }

}

**Приложение Б**

(обязательное)

**Вид отношений ассоциация**

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace ProjectOop

{

namespace Entities

{

public class Employee // сотрудник

{

public int ID { get; set; }

[Required] [MaxLength(50)] public string Surname { get; set; }

[Required] [MaxLength(50)] public string Name { get; set; }

[MaxLength(50)] public string Patronymic { get; set; }

[Required] public DateTime DeviceDate { get; set; }

[Required] public string Salary { get; set; }

[Required] [MaxLength(15)] public string Login { get; set; }

[Required] [MaxLength(15)] public string Password { get; set; }

[Required] public Role Role { get; set; }

}

}

}

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace ProjectOop

{

namespace Entities

{

public class Sewing // пошив

{

public int ID { get; set; }

[Required] public Employee Author { get; set; }

[Required] public DateTime CreationDate { get; set; }

}

}

}

**Приложение В**

(обязательное)

**Вид отношений композиция**

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace ProjectOop

{

namespace Entities // сущности

{

public class Sketch

{

public int ID { get; set; }

[Required] public string Name { get; set; }

[Required] public string FileLocation { get; set; }

[Required] public Employee Author { get; set; }

[Required] public DateTime CreationDate { get; set; }

}

public class Product

{

public int ID { get; set; }

public Sketch Sketch { get; set; }

public Blueprint Blueprint { get; set; }

public Cut Cut { get; set; }

public Sewing Sewing { get; set; }

public bool QaPassed { get; set; }

}

**Приложение Г**

(обязательное)

**Вид отношений агрегация**

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace ProjectOop

{

namespace Entities // сущности

{

public class Material // материал

{

public int ID { get; set; }

[Required] public string Name { get; set; }

public ModelColor Сolor { get; set; }

public ICollection<Blueprint> RelatedBlueprints { get; set; }

}

public class ModelColor

{

public int ID { get; set; }

[Required] public string RgbValue { get; set; }

[Required] public string TextName { get; set; }

}

**Приложение Д**

(обязательное)

**Контекст БД**

using Microsoft.EntityFrameworkCore;

using ProjectOop.Entities;

using System.Threading.Tasks;

public class AppDbContext : DbContext

{

public DbSet<Employee> Employees { get; set; }

public DbSet<ModelColor> Colors { get; set; }

public DbSet<Sketch> Sketches { get; set; }

public DbSet<Blueprint> Blueprints { get; set; }

public DbSet<Cut> Cuts { get; set; }

public DbSet<Sewing> Sewings { get; set; }

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

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

public AppDbContext()

{

Database.EnsureCreated();

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

var host = "127.0.0.1";

var port = "5432";

var databaseName = "ProjectOOP";

var username = "postgres";

var password = "4545";

optionsBuilder.UseNpgsql($"Host={host};Port={port};Database={databaseName};Username={username};Password={password}");

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Employee>().HasIndex(e => e.ID).IsUnique();

modelBuilder.Entity<ModelColor>().HasIndex(e => e.ID).IsUnique();

modelBuilder.Entity<Material>().HasIndex(e => e.ID).IsUnique();

modelBuilder.Entity<Sketch>().HasIndex(e => e.ID).IsUnique();

modelBuilder.Entity<Blueprint>().HasIndex(e => e.ID).IsUnique();

modelBuilder.Entity<Cut>().HasIndex(e => e.ID).IsUnique();

modelBuilder.Entity<Sewing>().HasIndex(e => e.ID).IsUnique();

modelBuilder.Entity<Product>().HasIndex(e => e.ID).IsUnique();

}

}

**Приложение Е**

(обязательное)

**Листинг программного кода реализации работы с библиотекой классов**

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Project.Forms;

using ProjectOop.Entities;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Project

{

/// <summary>

/// Класс хранит в себе информацию о состоянии приложения, такую как список открытых форм, текущий пользователь и тд.

/// </summary>

public class ProgramContext : ApplicationContext

{

/// <summary>

/// IHost - контейнер зависимостей.

/// Инкапсулирует внутри себя логику создания и хранения экземпляров классов.

/// </summary>

private readonly IHost host;

private readonly List<Form> formsRegistry = new();

/// <summary>

/// Флаг того, что в данный момент можно выполнять выход из программы

/// </summary>

private bool exitAllowed = true;

/// <summary>

/// Флаг того, что в данный момент выполняется завершение программы

/// </summary>

private bool disposing = false;

public ProgramContext()

{

// Настраиваем контейнер зависимостей

host = Host.CreateDefaultBuilder()

.ConfigureServices((\_, services) =>

services

.AddSingleton<AppDbContext>() // AddSingleton говорит контейнеру, что экземпляр AppDbContext должен быть создан только один раз

.AddSingleton(this) // добавляем текущий экземпляр ProgramContext в контейнер в виде синглтона, чтобы его можно было подставить в нужные места

// (например в конструкторы, если там указан ProgramContext)

.AddTransient<LoginForm>() // AddTransient говорит контейнеру, что на каждый запрос зависимости указанного типа нужно создавать новый экземпляр

.AddTransient<ProductionForm>()

.AddTransient<EmployeeListForm>()

.AddTransient<MaterialListForm>()

.AddTransient<MaterialEditorForm>()

.AddTransient<ColorListForm>()

.AddTransient<ColorEditor>()

.AddTransient<DirectorForm>()

.AddTransient<EmployeeEditorForm>()

.AddTransient<DesignerForm>()

.AddTransient<SketchArtEditorForm>()

.AddTransient<SewingEditorForm>()

.AddTransient<CuttingEditorForm>()

.AddTransient<BlueprintEditorForm>()

.AddTransient<MaterialsSelectorForm>()

.AddTransient<DesignerForm>()

).Build();

host.StartAsync();

OnStart();

}

/// <summary>

/// Пользователь, который авторизован на данный момент в программе

/// </summary>

public Employee employee { get; private set; }

// уберите. Про internal лучше рассказать если спросят. То же самое что писать в комментах что такое int и bool. Лучше добавить что именно делает метод EditEmployee

//internal: компоненты класса или структуры доступен из любого места кода в той же сборке, однако он недоступен для других программ и сборок.

internal async Task EditEmployee(Employee employee)

{

var e = await CreateForm<EmployeeEditorForm>().SetEmployee(employee).EmployeeAsync(showModal: true);

if (e == null)

{

Debug.WriteLine("employee edit cancelled");

return;

}

Debug.WriteLine("employee edit complete, saving");

//FindAsync находит сущность с заданными значениями первичного ключа. Если сущность с заданными

//значениями первичного ключа отслеживается контекстом, то она возвращается немедленно,

//без запроса к базе данных. В противном случае делается запрос к базе данных.

var existing = await host.Services.GetRequiredService<AppDbContext>().Employees.FindAsync(e.ID);

if (existing == null)

{

Debug.WriteLine("cannot find employee in host.Services.GetRequiredService<AppDbContext>(), return");

return;

}

//Запись обеспечивает доступ к информации об отслеживании изменений и операциям для объекта.

host.Services.GetRequiredService<AppDbContext>().Entry(existing).CurrentValues.SetValues(e);

Debug.WriteLine("saving changes");

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

Debug.WriteLine("changes saved");

}

internal async Task EditColor(ModelColor color)

{

var c = await CreateForm<ColorEditor>().SetColor(color).ColorAsync(showModal: true);

if (c == null)

{

Debug.WriteLine("color edit cancelled");

return;

}

Debug.WriteLine("color edit complete, saving");

var existing = await host.Services.GetRequiredService<AppDbContext>().Colors.FindAsync(c.ID);

if (existing == null)

{

Debug.WriteLine("cannot find colors in host.Services.GetRequiredService<AppDbContext>(), return");

return;

}

host.Services.GetRequiredService<AppDbContext>().Entry(existing).CurrentValues.SetValues(c);

Debug.WriteLine("saving changes");

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

Debug.WriteLine("changes saved");

}

internal async Task EditMaterial(Material material)

{

var c = await CreateForm<MaterialEditorForm>().SetMaterial(material).MaterialAsync(showModal: true);

if (c == null)

{

Debug.WriteLine("material edit cancelled");

return;

}

Debug.WriteLine("material edit complete, saving");

var existing = await host.Services.GetRequiredService<AppDbContext>().Materials.FindAsync(c.ID);

if (existing == null)

{

Debug.WriteLine("cannot find material in host.Services.GetRequiredService<AppDbContext>(), return");

return;

}

host.Services.GetRequiredService<AppDbContext>().Entry(existing).CurrentValues.SetValues(c);

Debug.WriteLine("saving changes");

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

Debug.WriteLine("changes saved");

}

internal async Task EditSketch(Sketch sketch)

{

var s = await CreateForm<SketchArtEditorForm>().SetSketch(sketch).SketchAsync(showModal: true);

if (s == null)

{

Debug.WriteLine("sketch edit cancelled");

return;

}

Debug.WriteLine("sketch edit complete, saving");

var db = host.Services.GetRequiredService<AppDbContext>();

var existing = await db.Sketches.FindAsync(s.ID);

if (existing == null)

{

Debug.WriteLine("cannot find sketch in db, return");

return;

}

db.Entry(existing).CurrentValues.SetValues(s);

Debug.WriteLine("saving changes");

db.SaveChanges();

Debug.WriteLine("changes saved");

}

internal async Task EditCutting(Product product)

{

var cut = await CreateForm<CuttingEditorForm>().EditCutAsync(product.Cut,showModal: true);

if (cut == null)

{

Debug.WriteLine("cutting edit cancelled");

return;

}

product.Cut = cut;

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

}

internal async Task EditSewing(Product product)

{

var sewing = await CreateForm<SewingEditorForm>().EditSewingAsync(product.Sewing, showModal:true);

if (sewing == null)

{

Debug.WriteLine("sewing edit cancelled");

return;

}

Debug.WriteLine("sewing not null");

product.Sewing = sewing;

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

}

internal void QualityControlPassed(Product product)

{

product.QaPassed = true;

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

}

internal async Task ConvertFromSketchToBlueprint(Product product)

{

if (product.Blueprint != null) return;

var blueprint = await CreateForm<BlueprintEditorForm>().BlueprintAsync(product);

if (blueprint == null) return;

//создался технический эскиз

host.Services.GetRequiredService<AppDbContext>().Blueprints.Add(blueprint);

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

//в продукт записан технический эскиз

host.Services.GetRequiredService<AppDbContext>().Attach(product);

product.Blueprint = blueprint;

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

}

internal async Task AddNewEmployee()

{

var employee = await CreateForm<EmployeeEditorForm>().EmployeeAsync(showModal: true);

if (employee == null) return;

host.Services.GetRequiredService<AppDbContext>().Employees.Add(employee);

await host.Services.GetRequiredService<AppDbContext>().SaveChangesAsync();

}

internal async Task EditBlueprint(Product product)

{

await CreateForm<BlueprintEditorForm>().BlueprintAsync(product);

}

internal async Task AddNewColor()

{

var color = await CreateForm<ColorEditor>().ColorAsync(showModal: true);

if (color == null) return;

host.Services.GetRequiredService<AppDbContext>().Colors.Add(color);

await host.Services.GetRequiredService<AppDbContext>().SaveChangesAsync();

}

internal async Task CreateSketchAndProduct()

{

var sketch = await CreateForm<SketchArtEditorForm>().SketchAsync(showModal: true);

if (sketch == null) return;

host.Services.GetRequiredService<AppDbContext>().Sketches.Add(sketch);

host.Services.GetRequiredService<AppDbContext>().Products.Add(new Product()

{

Sketch = sketch

});

await host.Services.GetRequiredService<AppDbContext>().SaveChangesAsync();

}

internal async Task AddNewMaterial()

{

var material = await CreateForm<MaterialEditorForm>().MaterialAsync(showModal: true);

if (material == null) return;

host.Services.GetRequiredService<AppDbContext>().Materials.Add(material);

await host.Services.GetRequiredService<AppDbContext>().SaveChangesAsync();

}

private void OnStart()

{

// проверяем всех сотрудников в базе данных. Если есть хоть один сотрудник, открываем форму входа. Если нет, показываем форму для добавления информации о сотруднике

if (host.Services.GetRequiredService<AppDbContext>().Employees.Any())

{

ShowForm<LoginForm>();

}

else

{

Debug.WriteLine("users not found, show editor form");

// C помощью булевых флагов обрабатываем сценарий, когда в программе не зарегистрирован ни один пользователь

// а редактор пользователей был закрыт без сохранения.

// Программа реализована таким образом, чтобы можно было открывать сразу несколько окон,

// и чтобы программа завершалась после закрытия последнего открытого окна

// флаг exitAllowed, установленный в false позволяет не выполнять выход из программы после закрытия формы редактора пользователей.

// данный сценарий нужен для первоначального запуска программы

exitAllowed = false;

var editorForm = ShowForm<EmployeeEditorForm>();

var OnEmployeeReadyHasBeenCalled = false;

editorForm.OnEmployeeReady += (sender, employee) =>

{

OnEmployeeReadyHasBeenCalled = true;

exitAllowed = true;

employee.Role = Role.DIRECTOR;

host.Services.GetRequiredService<AppDbContext>().Employees.Add(employee);

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

ShowMainForm(employee);

((Form)sender).Close();

};

editorForm.FormClosed += (\_, args) =>

{

if (!OnEmployeeReadyHasBeenCalled)

{

exitAllowed = true;

MessageBox.Show("Требуется хотя бы один пользователь");

ExitProgram();

}

};

}

}

internal async Task DeleteEmployee(Employee employee)

{

host.Services.GetRequiredService<AppDbContext>().Employees.Remove(employee);

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

}

internal async Task DeleteSketch(Sketch sketch)

{

host.Services.GetRequiredService<AppDbContext>().Sketches.Remove(sketch);

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

}

internal async Task DeleteColor(ModelColor color)

{

host.Services.GetRequiredService<AppDbContext>().Colors.Remove(color);

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

}

internal async Task DeleteMaterial(Material material)

{

host.Services.GetRequiredService<AppDbContext>().Materials.Remove(material);

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

}

private void ExitProgram()

{

if (disposing) return;

disposing = true;

host.Dispose();

ExitThread();

}

internal void SendToCut(Product product)

{

product.Sewing = null;

host.Services.GetRequiredService<AppDbContext>().SaveChanges();

}

#region "forms"

public T CreateForm<T>() where T : Form

{

var form = host.Services.GetRequiredService<T>();

formsRegistry.Add(form);

form.FormClosed += OnFormClosed;

return form;

}

private void OnFormClosed(object sender, EventArgs e)

{

var form = (Form)sender;

if (formsRegistry.Remove(form) && formsRegistry.Count == 0 && exitAllowed)

{

ExitProgram();

}

}

// T - универсальный параметр. Метод создает и показывает форму

public T ShowForm<T>() where T : Form

{

var form = CreateForm<T>();

form.Show();

return form;

}

// Проверка роли и создание формы в зависимости от роли

private Form CreateMainForm(Employee employee)

{

this.employee = employee;

switch (employee.Role)

{

case Role.DIRECTOR: return CreateForm<DirectorForm>();

case Role.DESIGNER: return CreateForm<DesignerForm>();

default: return CreateForm<ProductionForm>();

}

}

public void ShowMainForm(Employee employee)

{

var form = CreateMainForm(employee);

form.Show();

}

public void DoColorEdit()

{

ShowForm<ColorListForm>();

}

public void ShowEmployerList()

{

ShowForm<EmployeeListForm>();

}

public void ShowProductionList()

{

ShowForm<ProductionForm>();

}

public void ShowMaterialList()

{

ShowForm<MaterialListForm>();

}

#endregion

}

}

Приложение Ж

(справочное)

**Библиографический список**

1. Entity Framework [электронный ресурс] // обновлено 02.19.2020 URL: <https://docs.microsoft.com/en-us/aspnet/entity-framework/> (дата обращения 9.01.2022).
2. Руководство в Entity Framework [электронный ресурс] // Последнее обновление 31.10.2015 URL: <https://metanit.com/sharp/efcore/> (дата обращения 9.01.2022).
3. Отношения между классами и объектами [электронный ресурс] // Последнее обновление 31.10.2015 URL: <https://metanit.com/sharp/patterns/1.2.php/> (дата обращения 9.01.2022).
4. Отношения между классами и объектами [электронный ресурс] // Последнее обновление 16.08.2018 URL: <https://flexberry.github.io/ru/gpg_class-diagram.html> (дата обращения 9.01.2022).