МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

Факультет автоматики и вычислительной техники

Кафедра систем автоматизации управления

**РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ**

**СИСТЕМЫ ПТИЦЕФАБРИКИ**

Пояснительная записка

Отчёт по дисциплине

«Объектно-ориентированное программирование»

ТПЖА 090302.361 ПЗ

Разработали студенты гр. ИТб-2301-01-00\_\_\_\_\_\_\_\_\_\_\_\_/ Ласкин М.В./

(подпись)

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_/ Кузьминых М.М./

(подпись)

Работа защищена с оценкой «\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_»\_\_\_\_\_\_\_\_\_\_2022 г.

Киров 2022

Реферат

Ласкин М.В. Морозов И.В. Реализация информационной системы учета работы клининговой компании: ТПЖА. 090302.361 ПЗ: Отчет по курсовой работе / ВятГУ, каф. САУ; рук. М.М. Кузьминых. – Киров, 2022. ПЗ 46 с., 13 рис., 1 табл., 6 источников, 9 прил.

ПЕРЕЧЕНЬ КЛЮЧЕВЫХ СЛОВ: ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ, ИНФОРМАЦИОННАЯ СИСТЕМА, C#, ОБЪЕКТНО-ОРИЕНТИРОВННОЕ ПРОГРАММИРОВАНИЕ, ENTITY FRAMEWORK, 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.

Реализована информационная система с использованием Entity Framework в среде программирования Visual Studio.

Разработано приложение, осуществляющее добавление клиентов в БД, расчет и оформление заявок, формирование отчетов за выбранный период времени.

|  |  |
| --- | --- |
| **Содержание**  Изм.  Лист  № докум.  Подпись  Дата  Лист  1  ТПЖА 090302.361 ПЗ  Разраб.  *Ведерников Д.М.*  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...................………………………………. | 6 |
| 1. Реализация классов на языке программирования C#..................……........ | 8 |
| 3.1 Отношение между классами...........................................……………… | 8 |
| 3.2 Полиморфизм......................................................…………………...….. | 8 |
| 1. Создание базы данных с использованием Entity Framework..................... | 9 |
| 4.1 Создание контекста для базы данных....……...…………………… | 9 |
| 4.2 Миграции.....................................................................……….……... | 9 |
| 4.3 Создание БД................................................................……….……... | 9 |
| 1. Разработка приложения, реализующего базовую функциональность.. | 11 |
| 5.1 Вход в приложение....……...…………………..............................…..... | 11 |
| 5.2 Окно администратора...................................……...…………………… | 11 |
| 5.2.1 Форма выбора клиента и адреса................................................... | 12 |
| 5.2.2 Форма оформления новой заявки.................................................. | 12 |
| 5.2.3 Форма просмотра заявок................................................................ | 14 |
| 5.2.4 Форма отчетности........................................................................... | 16 |
| 5.3 Окно бригадира...................................……...…………………….......... | 16 |
| 5.3.1 Информация о бригаде................................................................... | 17 |
| 5.3.2 Заявки бригады................................................................................ | 17 |
| Заключение………………………………………………………………..…… | 20 |
| Приложение А (обязательное) Ключевые классы……………………...........  Изм.  Лист  № докум.  Подпись  Дата  Лист  2  ТПЖА 090302.361 ПЗ  Разраб.  *Ведерников Д.М.*  Провер.  Кузьминых М.М.  Реценз.  Н. Контр.  Утверд.  Литер  Листов  2  *Кафедра САУ,*  *гр. ИТб-2301-01-00* | 21 |
| Приложение Б (обязательное) Вид отношения ассоциация.......................… | 26 |
| Приложение В (обязательное) Вид отношения агрегация............................. | 29 |
| Приложение Г (обязательное) Вид отношения композиция.......................... | 31 |
| Приложение Д (обязательное) Реализация полиморфизма............................ | 33 |
| Приложение Ж (обязательное) Контекст БД................................................... | 36 |
| Приложение З (обязательное) Миграции......................................................... | 38 |
| Приложение И (обязательное) Листинг программного кода реализации работы с библиотекой классов.......................................................................... | 40 |
| Приложение К (справочное) Библиографический список………………..... | 46 |

# Введение

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Целью курсовой работы является закрепление теоретических знаний об объектно-ориентированном подходе программирования на практике, и приобретение практических навыков проектирования информационной системы.

Основная целевая платформа для разработки – Microsoft.Net Framework, язык программирования – C#, библиотека ORM – Entity Framework.

Задачи 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).

# 

# **Обзор использованной литературы**

Главной целью данного обзора является изучение инструментов для разработки информационной системы таких как: Entity Framework, Windows Presentation Foundation (WPF).

В первую очередь необходимо изучить объектно-ориентированную технологию на базе фреймворка .NET – Entity Framework для работы с реляционными данными. Для этого обратимся к сайтам в интернете [1-3, приложение К]. Данные статьи демонстрируют основные принципы работы с данными при использовании Entity Framework.

Особое внимание было уделено внесению изменений в имеющуюся базу данных при помощи миграций [4, приложение К]. Из статьи можно понять все преимущества миграций по сравнению с ручным изменением данных.

Для построения диаграммы классов необходимо изучить виды отношений между классами и объектами. Для этого обратимся к статье электронного ресурса METANIT.COM [5, приложение К]. Из статьи можно узнать о трех основных видах связей: ассоциация, композиция и агрегация. А также рекомендации по их применению.

Для построения графического интерфейса была выбрана и изучена технология WPF [6, приложение К]. По завершении изучения приобретены навыки использование языка декларативной разметки интерфейса XAML, а также изучены способы обработки событий, вызванных инструментами графического интерфейса.

# Анализ предметной области

Клининговая компания – это организация, представляющая услуги профессиональной очистки помещений. Суть услуги – приведение в порядок помещений, создание чистоты. Это похоже на работу уборщика, но с некоторыми уточнениями.

В отличие от простой уборки, клининг – это сервис, оказывающий комплексную услугу по чистке. Она сочетает уборку, уход за обстановкой, предупреждение поломок и избавление от паразитов, когда они есть.

В клининг-компаниях есть бригады рабочих, арсенал средств для проведения работ (химических, технических и т.д.). Это значит, что клининговые услуги таковы:

1. Тщательная уборка – генеральная или текущая.
2. Очищение окон и фасадов здания.
3. Уборка внешних помещений – террас, балконов, оценка их состояния.
4. Полная уборка после проведенного ремонта.
5. Вывоз мусора.
6. Уход за мебелью, поверхностями, коврами для поддержания их в хорошем состоянии.
7. Дезинфекция, дератизация, устранение очагов плесени.

В основе услуги лежит всесторонний подход к наведению чистоты и порядка. Бригада из нескольких человек учитывает требования по отдельным помещениям от заказчика, подходит к работе с профессионализмом. Деятельность клининга и особенности в проведении самой уборки зависят от специализации. То есть того, для какой цели будет нанята команда.

* 1. Процессы, протекающие в предметной области

Основным процессом является предоставление клиенту услуг по уборке на объекте. Для реализации данного процесса в первую очередь администратору необходимо обговорить с клиентом перечень оказываемых услуг, дату и время выполнения и заключить с ним договор. Затем в указанное в заявке время на объект приезжает бригада с профессиональным оборудованием и совершает уборку в соответствии со списком предоставляемых услуг. По завершении проведения работ, бригадир фиксирует успешное окончание мероприятий по уборке на объекте.

Еще одним немаловажным процессом является закупка, поставка и контроль расходуемого материала. В каждой услуге должна учитываться норма расхода для расходуемого по заявке материала. Сотрудник создает заявку на закупку, содержащую необходимые позиции расходников, и отправляет её поставщику. Поставщик в свою очередь совершает поставку расходников, прописанных в содержимом поставки. Все расходники имеют единицы измерения, прописанные в одноименном справочнике.

Разрабатываемая ИС должна оптимизировать процесс оформления заявки на предоставление услуг. А именно: заявка может быть оформлена на уже обслуживающегося ранее клиента. По выбранным услугам должен производиться автоматический расчет стоимости и примерного времени оказания услуг. Возможность просмотра списка всех заявок и подробной информации о них. Функционал приложения должен предоставлять возможность производить расчет прибыли за выбранный промежуток времени.

* 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 | Расходник, инвентарь, договор |
| 2 | Спецификации, описания объектов | Норма расхода, тип инвентаря, должность |
| 3 | Места | Адрес |
| 4 | Транзакции | Поставка, заявка, договор |
| 5 | Элементы транзакций | Услуга, расходник |
| 6 | Роли людей | Администратор, клиент, бригадир, клинер |
| 7 | Контейнеры других объектов | Справочник единиц измерения, инвентарь бригады |
| 8 | Содержимое контейнеров | Единицы измерения, инвентарь |
| 9 | Организации | Поставщик |
| 10 | События | Поставка |
| 11 | Процессы | Оказываемая услуга |
| 12 | Каталоги | Справочник единиц измерения, услуга, инвентарь, должность, норма расхода |
| 13 | Записи о деятельности | Договор |

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

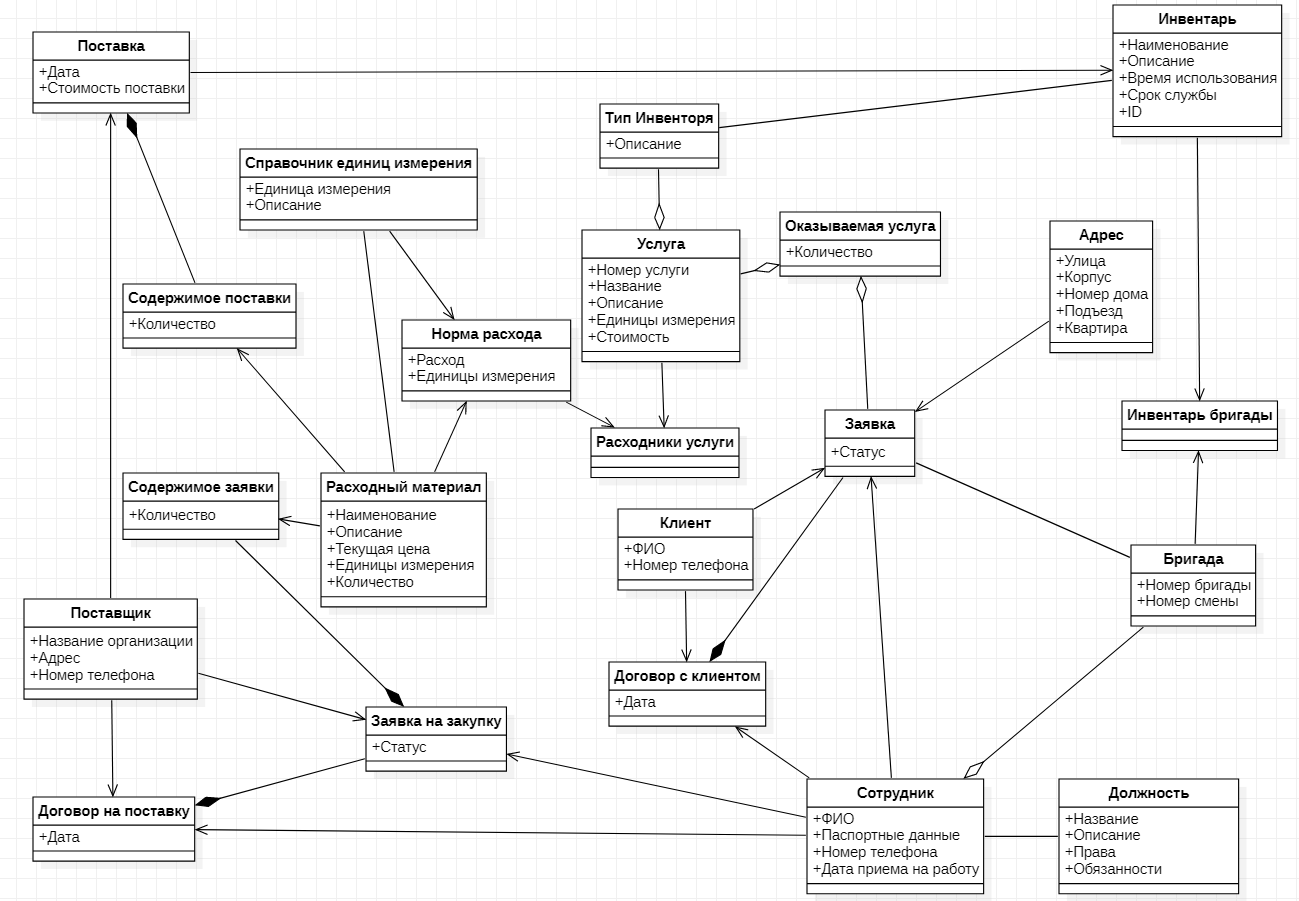


Рисунок 1 – Диаграмма классов UML

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: Client, Employee, Order и Service. Код ключевых классов представлен в приложении А. Для всех классов реализованы конструкторы, задающие корректное начальное состояние объектов.

**3.1 Отношения между классами**

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

Между классами ProvidedService и Order реализован вид отношений агрегация. Листинг данных классов находится в приложении В. При агрегации реализуется слабая связь, то есть в данном случае объекты оказываемая услуга и заявка будут равноправны. В конструктор оказываемая услуга передается ссылка на уже имеющийся объект заявка.

Между классами Contract и Order реализован вид отношения композиция. Реализация данных классов представлена в приложении Г Композиция определяет отношение "имеет". Класс заявка содержит объект класса контракт.

**3.2 Полиморфизм**

Полиморфизм — это способность объекта использовать методы производного класса, который не существует на момент создания базового. В программе реализован полиморфизм на основе наследования. Родительским классом является класс Human, а дочерними Client и Employee. Код классов, реализующих полиморфное поведение, а также код, где выполняется полиморфный вызов представлен в приложении Д.

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

1. **Создание базы данных с использованием Entity Framework**

В рамках курсовой работы в качестве библиотеки ORM использовалась Entity Framework. Entity Framework — это решение для работы с базами данных, которое используется в программировании на языках семейства.NET. Оно позволяет взаимодействовать с СУБД с помощью сущностей (entity), а не таблиц. Также код с использованием EF пишется гораздо быстрее.

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 ApplicationContext. Он является наследником DbContext. Код данного типа предоставлен в приложении З.

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

* 1. **Миграции**

Для изменения моделей в Entity Framework, которые входят в контекст данных, например, добавление или удаление некоторых свойств, необходимо, чтобы база данных также применяла эти изменения. Для этого используется такой инструмент как миграция.

Сгенерированные Entity Framework миграции представлены в приложении И.

4.3 Создание БД

Диаграмма базы данных предоставлена на рисунке 2.

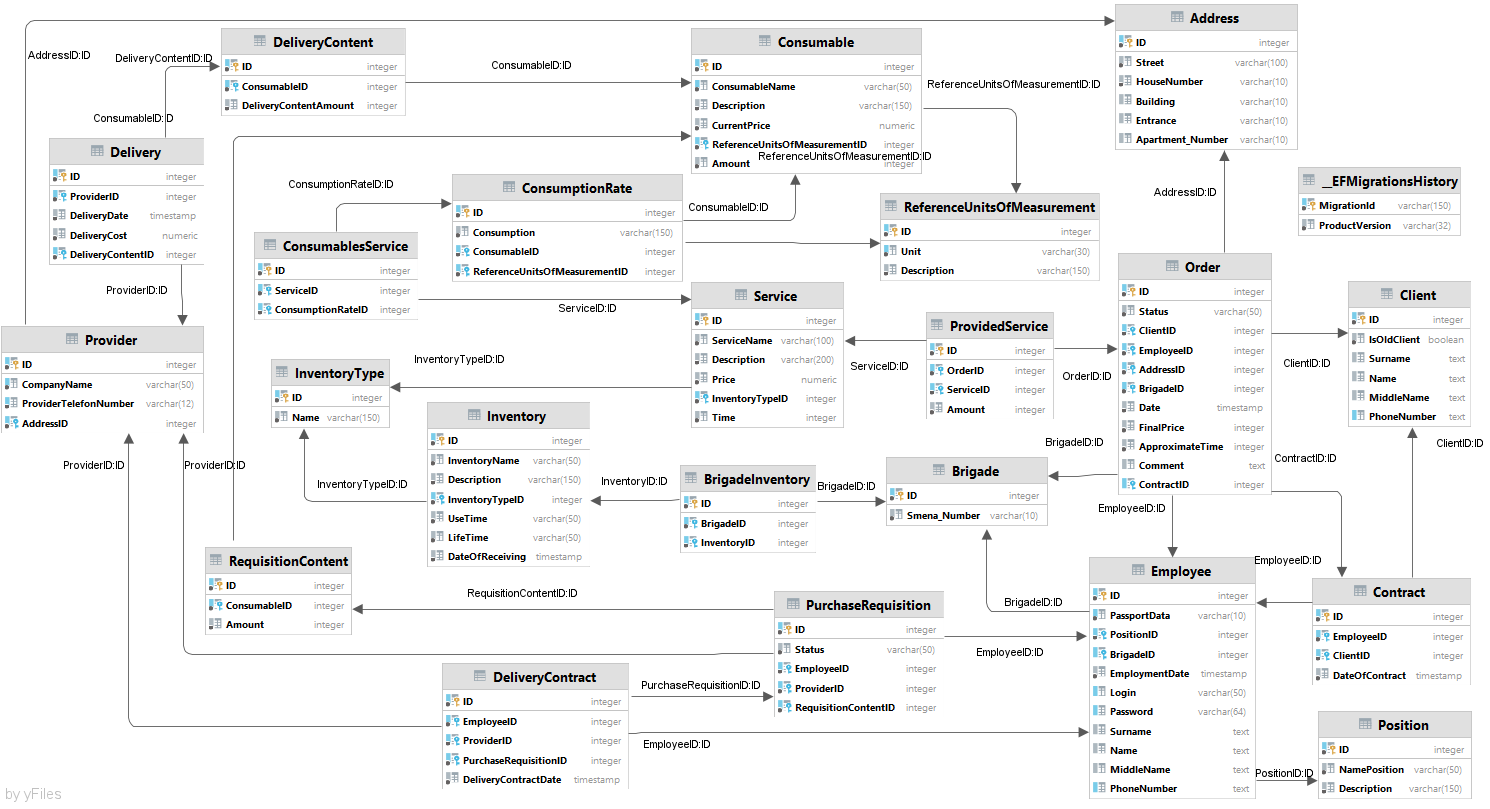


Рисунок 2 – Схема базы данных

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

По итогу в данном разделе был создан контекст для работы с базой данных. Применена миграция для изменения базы данных. Смоделирована физическая модель данных.

1. **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, при помощи технологии построения интерфейса WPF, для двух ролей: администратор и бригадир. 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.

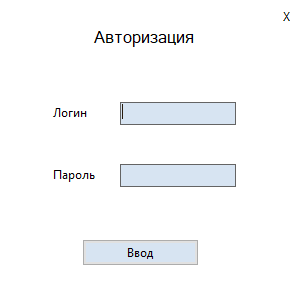


Рисунок 3 – Вход в приложение

Каждый администратор и сотрудник имеет свой уникальный логин и пароль, хранящиеся в hash, для входа в приложение.

**5.2 Окна работников птицефабрики**

Возможности работников: формирование партий с последующим добавлением в базу данных, добавление партий в цеха и отправление партий на следующие этапы.

В каждом окне имеется кнопка “Обновить данные”, она необходима для обновления данных в таблицах с выводом данных о партиях и цехах. Кнопки добавления в цеха позволяют добавлять партии яиц или птиц в цеха для инкубации/взращивания/производства яиц/сортировки яиц/производства фабриката/производства меланжа.

**5.2.1 Форма работника репродуктора**

Данная форма предназначена для формирования партий яиц из репродуктора, либо добавление недостающих особей в репродуктор. Данная форма представлена на рисунке 4.

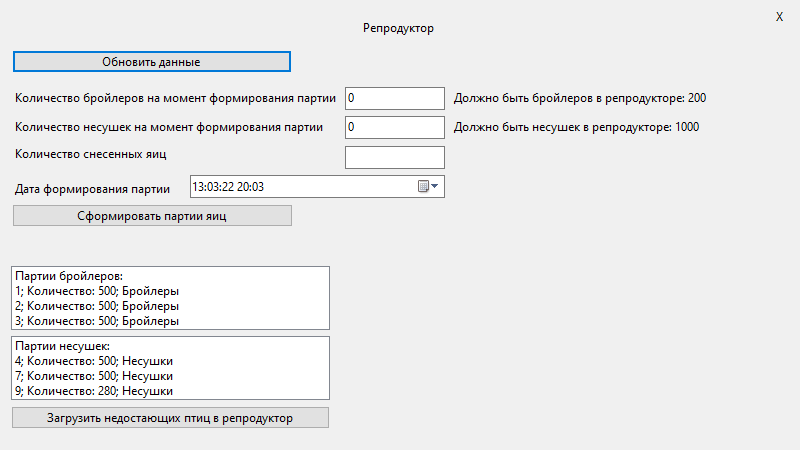


Рисунок 4 – Форма работника репродуктора

После заполнения всех полей и нажатия кнопки “Сформировать партии яиц”, в БД фиксируются количество несушек и бройлеров и партии яиц с количеством менее или равным вместимости инкубатора. Недостающих птиц можно восполнить из партий доступных в списках выбрав их и нажав на кнопку “Загрузить недостающих птиц в репродуктор”. Подсказка о том, сколько птиц должно находится в репродукторе не статична, а настраивается администратором.

**5.2.2 Форма работника инкубатора**

Форма предназначена для занесения партии яиц из репродуктора в инкубаторий и формирования партий молодых бройлеров и несушек. Форма изображена на рисунке 5.

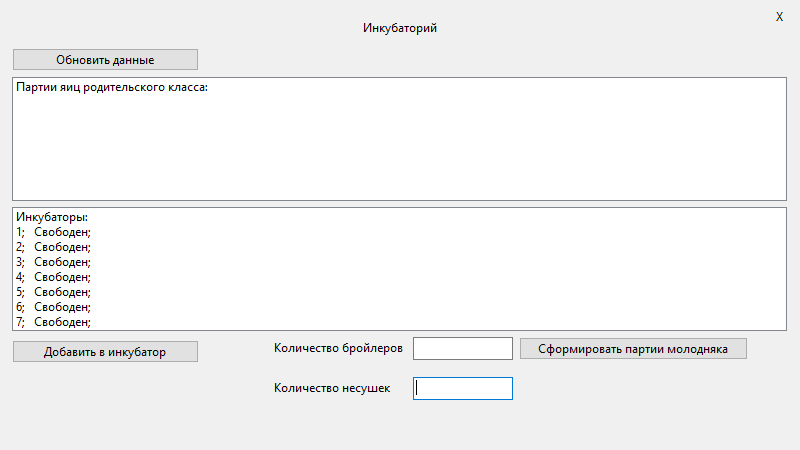


Рисунок 5 – Форма работника инкубатора

После выбора партии и инкубатора из соответствующих списков и по нажатию на кнопку “Добавить в инкубатор” в информация о партии яиц отобразится в инкубаторе, произойдет расчет даты вылупления и поменяется статут инкубатора. После выбора инкубатора, заполнения полей количества бройлеров и несушек и нажатия на кнопку “Сформировать партии молодняка”, в БД поступает информация о партиях при условии окончания периода вылупления и корректных данных о количестве особей.

**5.2.3 Форма работника птичника**

Форма предназначена для занесения партии молодняка в птичник и формирования партий бройлеров и несушек. Форма изображена на рисунке 6.

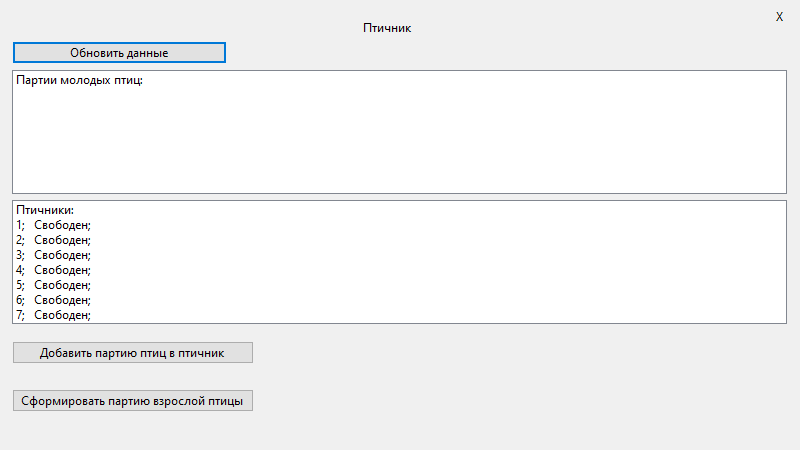


Рисунок 6 – Форма работника птичника

Для занесения партии в птичник работник выбирает нужную ему партию и птичник и нажимает кнопку “Добавить партию птиц в птичник”. После этого в поле птичника отобразится информация о загруженной партии, расчетная дата выроста и статус птичника. После выбора птичника и нажатия на кнопку “Сформировать партию взрослой птицы”, в БД поступает информация о партии при условии наступления расчетной даты выроста.

**5.2.4 Форма работника комплекса производства яиц**

Даннаяформа предназначена для внесения партии несушек в комплекс производства яиц и формирования партий яиц. Ознакомиться с данной формой можно на рисунке 7.

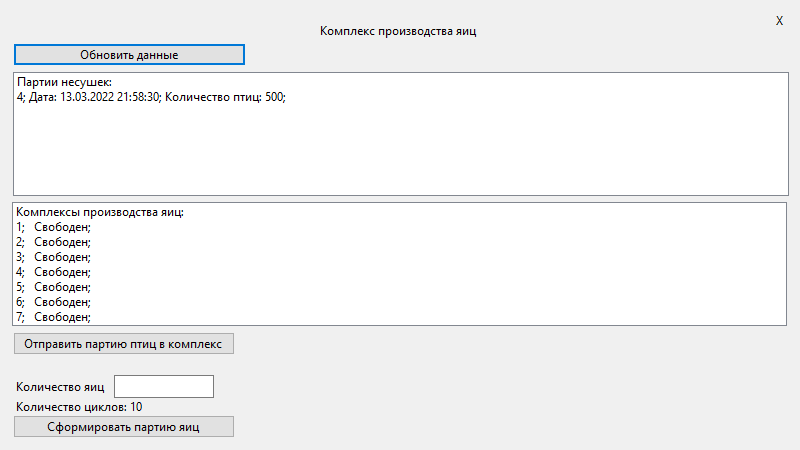


Рисунок 7 – Форма работника комплекса производства яиц

После выбора партии и комплекса из соответствующих списков и по нажатию на кнопку “Отправить партию птиц в комплекс” в комплексе отобразится информация о партии несушек и количество циклов сбора яиц, пройденных ей. После выбора птичника, заполненияполя количества яиц и нажатия на кнопку “Сформировать партию яиц”, в БД поступает информация о партии при условии, что количество пройденных циклов у партии, располагающейся в комплексе, не превышает порогового значения, установленного администратором. Если пороговое значение достигнуто, то птичник освобождается.

**5.2.5 Форма работника цеха сортировки яиц**

Форма предназначена для сортировки партии яиц по категориям. Форма изображена на рисунке 8.

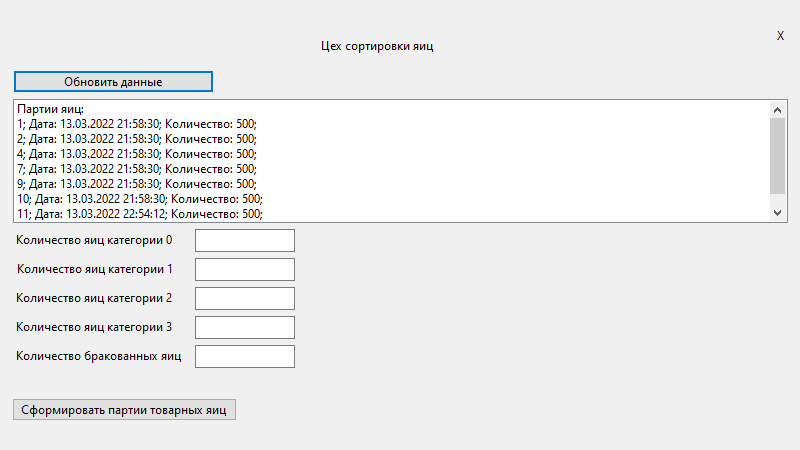


Рисунок 8 – Форма работника цеха сортировки яиц

После выбора партии яиц, заполненияполей количества яиц по категориям и нажатия на кнопку “Сформировать партии товарных яиц”, в БД поступает информация о партиях при условии, что общее указанное количество яиц не превышает изначальное.

**5.2.6 Форма работника убойно перерабатывающего комплекса**

Форма предназначена для преобразования партий бройлеров в партии фабрикатов. Форма изображена на рисунке 9.

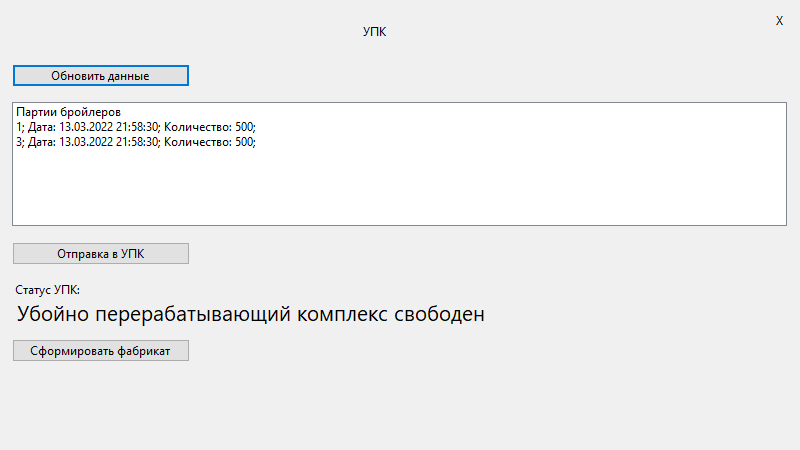


Рисунок 9 – Форма работника УПК

Для занесения партии в УПК работник выбирает нужную ему партию и нажимает кнопку “Отправить в УПК”. После этого статус птичника поменяется. После нажатия на кнопку “Сформировать фабрикат”, в БД поступает информация о партии фабриката.

**5.2.7 Форма работника цеха производства меланжа**

Форма предназначена для преобразования поврежденных яиц в меланж. Интерфейс данной формы представлен на рисунке 10.

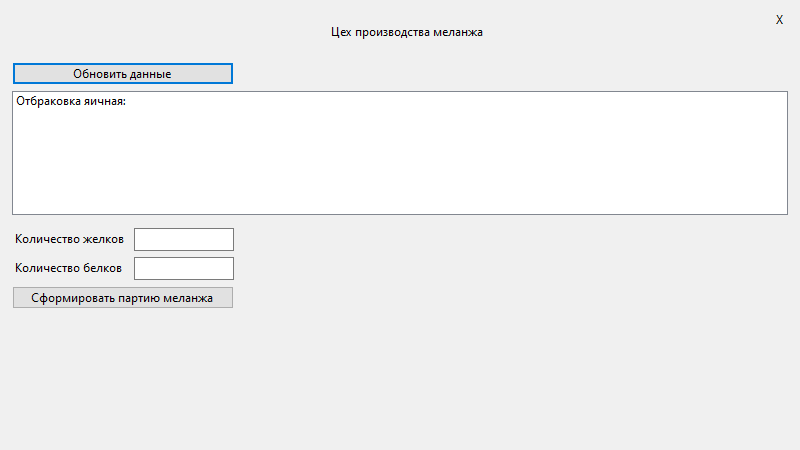


Рисунок 10 – Форма работника цеха производства меланжа

Для формирования меланжа работник выбирает нужную ему партию, заполняет формы и нажимает кнопку “Сформировать партию меланжа”. После нажатия в БД поступает информация о партиях меланжа.

* 1. **Форма руководителя**

Возможности руководителя: просмотр состояния всех цехов, комплексов, партий яиц, молодняка, несушек, бройлеров, готовой продукции. Удаление партий яиц, молодняка, несушек, бройлеров, готовой продукции и обнуление цехов и комплексов. Формирование квартального отчета по готовой продукции в виде exel документов. Изменение необходимого количества бройлеров и несушек в репродукторе, количества циклов несения у несушек в комплексе производства яиц.

В верхней части формы находится навигация в виде вкладок, в правом углу находится кнопка выхода из окна в окно авторизации. Реализацию можно увидеть на рисунке 11.

**5.3.1 Вкладка с состоянием цехов и комплексов**

Форма предназначена для вывода состояния цехов и комплексов с возможностью очистки (обнуления цеха). Описанная форма располагается на рисунке 11.

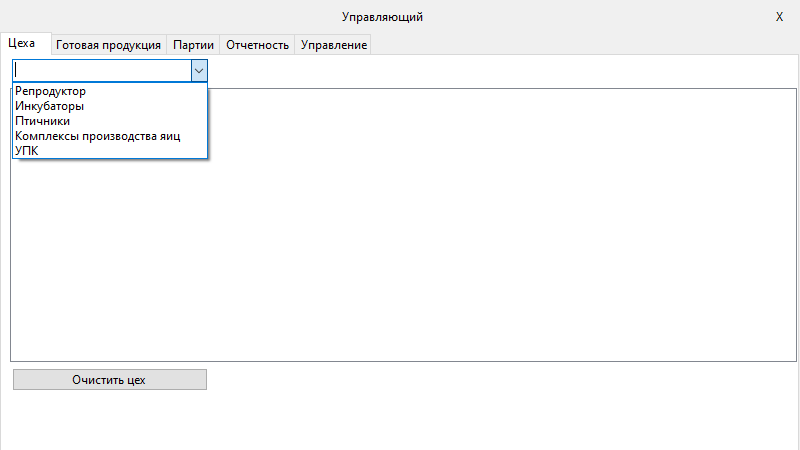


Рисунок 11 – Вкладка с состоянием цехов и комплексов

Функционал: выбор типа вывода цехов и комплексов с возможностью очистки цеха.

**5.3.2 Вкладка с готовой продукцией**

Форма предназначена для вывода готовой продукции с возможностью её удаления. Данная форма представлена на рисунке 12.

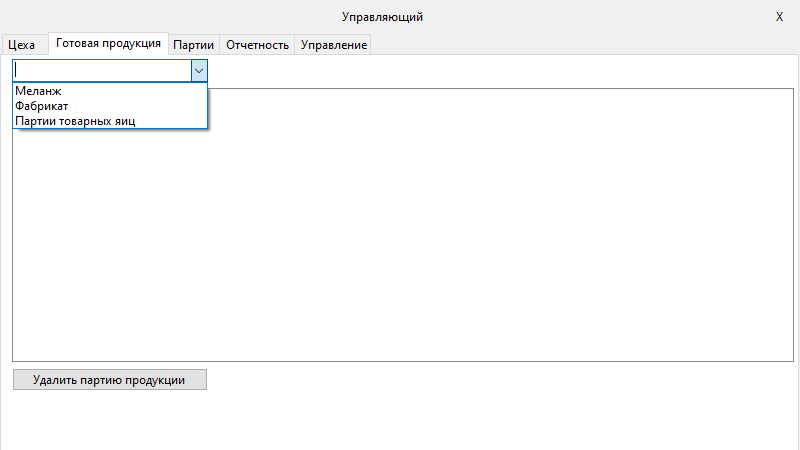


Рисунок 12 – Заявки бригады

Функционал: выбор типа вывода готовой продукции с возможностью её удаления.

**5.3.3 Вкладка с партиями яиц/куриц**

Форма предназначена для вывода партий с возможностью их удаления. Данная форма представлена на рисунке 12.

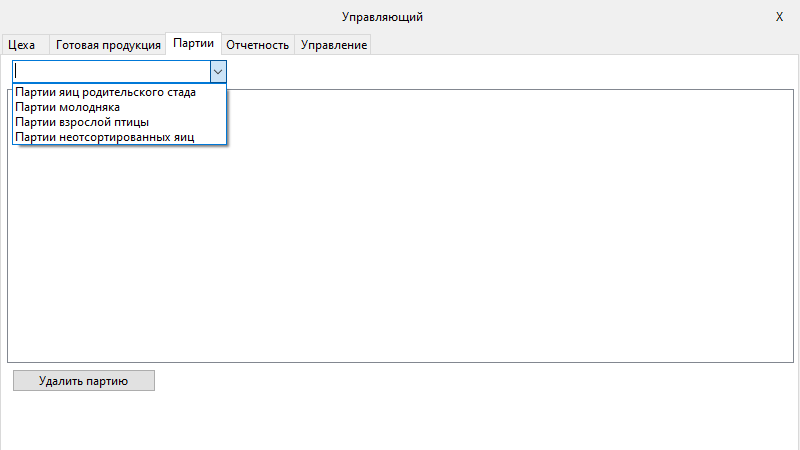


Рисунок 13 – Вкладка с партиями яиц/куриц

Функционал: выбор типа вывода партий с возможностью их удаления.

**5.3.4 Вкладка с созданием поквартального отчета**

Форма предназначена для создания поквартальных отчетов по готовой продукции в виде exel документов. Данная форма представлена на рисунке 12.

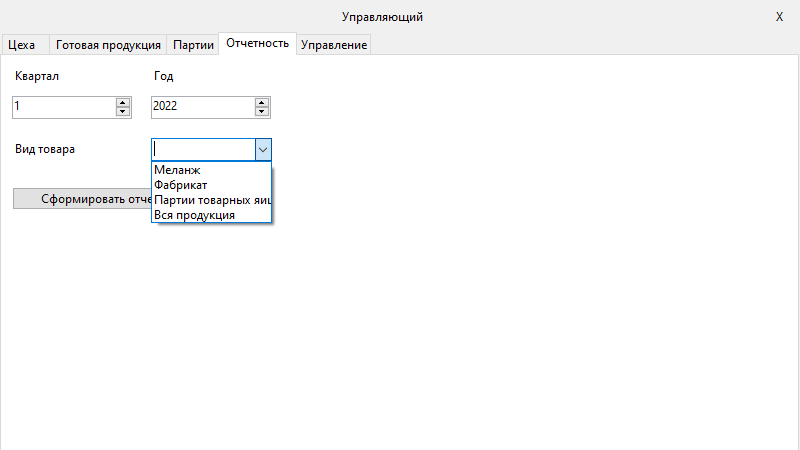


Рисунок 13 – Вкладка с партиями яиц/куриц

Функционал: выбор года, квартала, типа продукции в отчете. Создание отчета в виде exel документа по вышеперечисленным критериям.

В отличие от администратора, бригадир имеет возможность изменять только статус заявки и только на “В процессе” или “Завершена”. Кнопки “Редактировать” и “Сохранить изменения” работают аналогично одноименным кнопкам в окне полной информации о заявке для администратора.

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 двух ролей. Описаны возможности приложения для реализации перечня задач для каждой роли.

**Заключение**

В ходе выполнения курсовой работы были изучены основы объектно-ориентированного анализа предметной области, выявлены классы и отношения между ними. Для реализации выявленных классов были освоены приемы ООП, также была рассмотрена ORM система EntityFraemwork. Спроектирована и реализована модель базы данных. Для реализации пользовательского интерфейса была изучена технология построения интерфейса WPF. Реализованы все поставленные задачи предметной области.

**Приложение А**

(обязательное)

**Ключевые классы**

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Client : Human //Клиент

{

public int ID { get; set; }

[Required]

public bool IsOldClient { get; set; }

private static ApplicationContext db = Context.Db;

public Client()

{

}

public Client(string Surname, string Name, string MiddleName, string PhoneNumber, bool IsOldClient)

: base(Surname, Name, MiddleName, PhoneNumber)

{

this.IsOldClient = IsOldClient;

}

public override string GetFullName()

{

return "Клиент: " + base.GetFullName();

}

public static void ClientIsOld(int id)

{

Client oldClient = db.Client.Where(c => c.ID == id).FirstOrDefault();

oldClient.IsOldClient = true;

}

public static Client GetClientByTelefon(string telefon)

{

return db.Client.Where(e => e.PhoneNumber == telefon).ToList()[0];

}

public static bool ClientByTelefonIsNew(string telefon)

{

return !db.Client.Where(e => e.PhoneNumber == telefon).Any();

}

public static List<ClientInfo> GetClientInfo(string Telefon) //Клиент не связан с адресом. В листе хранится клиент с разными адресами

{

return (from c in db.Client

where c.PhoneNumber == Telefon

join o in db.Order on c.ID equals o.Client.ID

join a in db.Address on o.Address.ID equals a.ID

select new ClientInfo()

{

ID = c.ID,

Surname = c.Surname,

Name = c.Name,

MiddleName = c.MiddleName,

Street = a.Street,

HouseNumber = a.HouseNumber,

Building = a.Building,

Entrance = a.Entrance,

Apartment\_Number = a.Apartment\_Number,

IsOldClient = c.IsOldClient

}).ToList();

}

public class ClientInfo

{

public int ID { get; set; }

public string Surname { get; set; }

public string Name { get; set; }

public string MiddleName { get; set; }

public string Street { get; set; }

public string HouseNumber { get; set; }

public string Building { get; set; }

public string Entrance { get; set; }

public string Apartment\_Number { get; set; }

public bool IsOldClient { get; set; }

}

public static bool proverkaClientTelefon(string Telefon)

{

return db.Client.Where(a => a.PhoneNumber == Telefon).Any();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Employee : Human //Сотрудник

{

public int ID { get; set; }

[Required]

[MaxLength(10)] public string PassportData { get; set; }

[Required]

public virtual Position Position { get; set; }

[Required]

public int PositionID { get; set; }

public virtual Brigade? Brigade { get; set; }

public int? BrigadeID { get; set; }

[Required]

public DateTime EmploymentDate { get; set; }

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

[MaxLength(64)] [MinLength(64)] public string? Password { get; set; }

private static ApplicationContext db = Context.Db;

public Employee()

{

}

public Employee(string Surname, string Name, string MiddleName, string PassportData, string PhoneNumber, int PositionID, int? BrigadeID,

DateTime EmploymentDate, string? Login, string? Password) : base(Surname, Name, MiddleName, PhoneNumber)

{

this.PassportData = PassportData;

this.PositionID = PositionID;

this.BrigadeID = BrigadeID;

this.EmploymentDate = EmploymentDate;

this.Login = Login;

this.Password = Password;

}

public override string GetFullName()

{

if(BrigadeID != null )

{

return "Исполнитель: " + base.GetFullName();

}

return "Заявку принял: " + base.GetFullName();

}

public class EmployeeInfo

{

public int Surname { get; set; }

public string Name { get; set; }

public string MiddleName { get; set; }

}

public static Employee GetBrigadirByBrigada(int id)

{

return db.Employee.Where(a => a.BrigadeID == id && a.Position.ID == 2).FirstOrDefault();

}

public static Employee GetEmployee(string Login, string Password)

{

return db.Employee.Where(a => a.Login == Login && a.Password == Password).FirstOrDefault();

}

public static Employee GetEmployeeBrigade(int Brigade)

{

return db.Employee.Where(a => a.BrigadeID == Brigade).FirstOrDefault();

}

public static List<EmployeeFullInfo> GetEmployeeFullInfo()

{

return (from e in db.Employee

join p in db.Position on e.PositionID equals p.ID

select new EmployeeFullInfo()

{

ID = e.ID,

Cleaner = e.AddFIO(),

Positions = p.NamePosition,

WorkExperience = e.EmploymentDate.ToString("d"),

Brigade = e.Brigade.ID,

Telefone = e.PhoneNumber,

}).ToList();

}

public class EmployeeFullInfo

{

public int ID { get; set; }

public string Cleaner { get; set; }

public string Positions { get; set; }

public string WorkExperience { get; set; }

public int? Brigade { get; set; }

public string Telefone { get; set; }

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Order //Заявка

{

public int ID { get; set; }

[Required]

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

[Required]

public virtual Client Client { get; set; }

[Required]

public virtual Employee Employee { get; set; }

[Required]

public virtual Address Address { get; set; }

[Required]

public virtual Brigade Brigade { get; set; }

public int BrigadeID { get; set; }

[Required]

public DateTime Date { get; set; }

public int FinalPrice { get; set; }

public int ApproximateTime { get; set; }

public string? Comment { get; set; }

public Contract? Contract { get; set; }

public int ContractID { get; set; }

public Order()

{

}

public Order(string Status, Client Client, Employee Employee, Address Address, Brigade Brigade, DateTime Date,

int FinalPrice, int ApproximateTime, string? Comment)

{

this.Status = Status;

this.Client = Client;

this.Employee = Employee;

this.Address = Address;

this.Brigade = Brigade;

this.Date = Date;

this.FinalPrice = FinalPrice;

this.ApproximateTime = ApproximateTime;

this.Comment = Comment;

this.Contract = new Contract( Employee, Client, DateTime.Now);

}

private static ApplicationContext db = Context.Db;

public static void CheckOrder()

{

List<Order> orders = db.Order.Where(o => o.Status == EnumStatus.GetDescription(EnumStatus.Status.wait) && o.Date < DateTime.Today).ToList();

foreach (var d in orders)

{

d.Status = EnumStatus.GetDescription(EnumStatus.Status.canceled);

}

db.SaveChanges();

}

public static bool IsOldClienCheck(int clientId)

{

List<Order> ordersByClientID = db.Order.Where(o => o.Client.ID == clientId && o.Status == EnumStatus.GetDescription(EnumStatus.Status.сompleted)).ToList();

if (ordersByClientID.Count >= 3)

return true;

return false;

}

public static Order GetOrderById(int id)

{

return db.Order.Where(e => e.ID == id).FirstOrDefault();

}

public static List<OrderInfo> GetOrderInfo()

{

return (from o in db.Order

join a in db.Address on o.Address.ID equals a.ID

select new OrderInfo()

{

ID = o.ID,

Time = o.Date.ToString("t"),

Date = o.Date.ToString("d"),

Brigade = o.Brigade.ID,

Status = o.Status,

Client = o.Client.AddFIO(),

Address = a.AddAddress(),

Telefone = o.Client.PhoneNumber,

FinalPrice = o.FinalPrice,

ApproximateTime = GetTimeByInt(o.ApproximateTime).ToString()

}).ToList();

}

public class OrderInfo

{

public int ID { get; set; }

public string Date { get; set; }

public string Time { get; set; }

public string Address { get; set; }

public string Status { get; set; }

public int Brigade { get; set; }

public string Telefone { get; set; }

public string Client { get; set; }

public string ApproximateTime { get; set; }

public int FinalPrice { get; set; }

}

public static string GetPriceByInt(int p)

{

return (p + " ₽");

}

public static int GetPriceByString(string p)

{

int price = Convert.ToInt32(p.Trim(new Char[] { ' ', '₽' }));

return price;

}

public static string GetTimeByInt(int t)

{

t = t / 60;

int h = t / 60;

int m = t % 60;

return (h + "ч. " + m + "мин.");

}

}

}

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Service //Услуга

{

public int ID { get; set; }

[Required]

[MaxLength(100)] public string ServiceName { get; set; }

[Required]

[MaxLength(200)] public string Description { get; set; }

[Required]

public decimal Price { get; set; }

public virtual InventoryType? InventoryType { get; set; }

public int InventoryTypeID { get; set; }

[Required]

public int Time { get; set; }

public Service()

{

}

public Service(string ServiceName, string Description, decimal Price, int InventoryTypeID, int Time)

{

this.ServiceName = ServiceName;

this.Description = Description;

this.Price = Price;

this.InventoryTypeID = InventoryTypeID;

this.Time = Time;

}

private static ApplicationContext db = Context.Db;

public static int GetIdService(string str)

{

int idService;

Service service = db.Service.Where(s => s.ServiceName == str).FirstOrDefault();

idService = service.ID;

return idService;

}

public static Service GetServiceById(int idService)

{

return db.Service.Where(s => s.ID == idService).FirstOrDefault();

}

}

}

**Приложение Б**

(обязательное)

**Вид отношений ассоциация**

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Position //Должность

{

public int ID { get; set; }

[Required]

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

[Required]

[MaxLength(150)] public string Description { get; set; }

public virtual ICollection<Employee> Employees { get; set; }

private static ApplicationContext db = Context.Db;

public Position()

{

}

public Position(string NamePosition, string Description)

{

Employees = new List<Employee>();

this.NamePosition = NamePosition;

this.Description = Description;

}

public static Position GetByID(int id)

{

return db.Position.Where(d => d.ID == id).FirstOrDefault();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Employee : Human //Сотрудник

{

public int ID { get; set; }

[Required]

[MaxLength(10)] public string PassportData { get; set; }

[Required]

public virtual Position Position { get; set; }

[Required]

public int PositionID { get; set; }

public virtual Brigade? Brigade { get; set; }

public int? BrigadeID { get; set; }

[Required]

public DateTime EmploymentDate { get; set; }

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

[MaxLength(64)] [MinLength(64)] public string? Password { get; set; }

private static ApplicationContext db = Context.Db;

public Employee()

{

}

public Employee(string Surname, string Name, string MiddleName, string PassportData, string PhoneNumber, int PositionID, int? BrigadeID,

DateTime EmploymentDate, string? Login, string? Password) : base(Surname, Name, MiddleName, PhoneNumber)

{

this.PassportData = PassportData;

this.PositionID = PositionID;

this.BrigadeID = BrigadeID;

this.EmploymentDate = EmploymentDate;

this.Login = Login;

this.Password = Password;

}

...

**Приложение В**

(обязательное)

**Вид отношений агрегация**

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class ProvidedService //Оказываемая услуга

{

public int ID { get; set; }

[Required]

public virtual Order Order { get; set; }

public int OrderID { get; set; }

[Required]

public virtual Service Service { get; set; }

public int ServiceID { get; set; }

[Required]

public int Amount { get; set; }

public ProvidedService()

{

}

public ProvidedService(Order Order, Service Service, int Amount)

{

this.Order = Order;

this.Service = Service;

this.Amount = Amount;

}

private static ApplicationContext db = Context.Db;

public static List<ProvidedService> GetPSByOrder(int id)

{

return (from ps in db.ProvidedService

where ps.OrderID == id

select ps

).ToList();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Order //Заявка

{

public int ID { get; set; }

[Required]

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

[Required]

public virtual Client Client { get; set; }

[Required]

public virtual Employee Employee { get; set; }

[Required]

public virtual Address Address { get; set; }

[Required]

public virtual Brigade Brigade { get; set; }

public int BrigadeID { get; set; }

[Required]

public DateTime Date { get; set; }

public int FinalPrice { get; set; }

public int ApproximateTime { get; set; }

public string? Comment { get; set; }

public Contract? Contract { get; set; }

public int ContractID { get; set; }

public Order()

{

}

public Order(string Status, Client Client, Employee Employee, Address Address, Brigade Brigade, DateTime Date,

int FinalPrice, int ApproximateTime, string? Comment)

{

this.Status = Status;

this.Client = Client;

this.Employee = Employee;

this.Address = Address;

this.Brigade = Brigade;

this.Date = Date;

this.FinalPrice = FinalPrice;

this.ApproximateTime = ApproximateTime;

this.Comment = Comment;

this.Contract = new Contract( Employee, Client, DateTime.Now);

}

...

**Приложение Г**

(обязательное)

**Вид отношения композиция**

using System;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Contract //Договор с клиентом

{

public int ID { get; set; }

[Required]

public virtual Employee Employee { get; set; }

[Required]

public virtual Client Client { get; set; }

[Required]

public DateTime DateOfContract { get; set; }

private static ApplicationContext db = Context.Db;

public Contract()

{

}

public Contract(Employee Employee, Client Client, DateTime DateOfContract)

{

this.Employee = Employee;

this.Client = Client;

this.DateOfContract = DateOfContract;

}

public static Contract GetContractById(int id)

{

return db.Contract.Where(c => c.ID == id).FirstOrDefault();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Order //Заявка

{

public int ID { get; set; }

[Required]

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

[Required]

public virtual Client Client { get; set; }

[Required]

public virtual Employee Employee { get; set; }

[Required]

public virtual Address Address { get; set; }

[Required]

public virtual Brigade Brigade { get; set; }

public int BrigadeID { get; set; }

[Required]

public DateTime Date { get; set; }

public int FinalPrice { get; set; }

public int ApproximateTime { get; set; }

public string? Comment { get; set; }

public Contract? Contract { get; set; }

public int ContractID { get; set; }

public Order()

{

}

public Order(string Status, Client Client, Employee Employee, Address Address, Brigade Brigade, DateTime Date,

int FinalPrice, int ApproximateTime, string? Comment)

{

this.Status = Status;

this.Client = Client;

this.Employee = Employee;

this.Address = Address;

this.Brigade = Brigade;

this.Date = Date;

this.FinalPrice = FinalPrice;

this.ApproximateTime = ApproximateTime;

this.Comment = Comment;

this.Contract = new Contract( Employee, Client, DateTime.Now);

}

...

**Приложение Д**

(обязательное)

**Реализация полиморфизма**

namespace CleaningDLL.Entity

{

public class Human

{

public string Surname { get; set; }

public string Name { get; set; }

public string? MiddleName { get; set; }

public string PhoneNumber { get; set; }

public Human()

{

}

public Human(string Surname, string Name, string PhoneNumber)

{

this.Surname = Surname;

this.Name = Name;

this.PhoneNumber = PhoneNumber;

}

public Human(string Surname, string Name, string MiddleName, string PhoneNumber)

{

this.Surname = Surname;

this.Name = Name;

this.MiddleName = MiddleName;

this.PhoneNumber = PhoneNumber;

}

public virtual string AddFIO()

{

string str = $"{Surname} {Name.Substring(0, 1)}.";

if (MiddleName != "") str += $"{MiddleName.Substring(0, 1)}.";

return str;

}

public virtual string GetFullName()

{

string str = $"{Surname} {Name}";

if (MiddleName != "") str += $" {MiddleName}";

return str;

}

}

}

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Client : Human //Клиент

{

public int ID { get; set; }

[Required]

public bool IsOldClient { get; set; }

private static ApplicationContext db = Context.Db;

public Client()

{

}

public Client(string Surname, string Name, string MiddleName, string PhoneNumber, bool IsOldClient)

: base(Surname, Name, MiddleName, PhoneNumber)

{

this.IsOldClient = IsOldClient;

}

public override string GetFullName()

{

return "Клиент: " + base.GetFullName();

}

...

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

namespace CleaningDLL.Entity

{

public class Employee : Human //Сотрудник

{

public int ID { get; set; }

[Required]

[MaxLength(10)] public string PassportData { get; set; }

[Required]

public virtual Position Position { get; set; }

[Required]

public int PositionID { get; set; }

public virtual Brigade? Brigade { get; set; }

public int? BrigadeID { get; set; }

[Required]

public DateTime EmploymentDate { get; set; }

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

[MaxLength(64)] [MinLength(64)] public string? Password { get; set; }

private static ApplicationContext db = Context.Db;

public Employee()

{

}

public Employee(string Surname, string Name, string MiddleName, string PassportData, string PhoneNumber, int PositionID, int? BrigadeID,

DateTime EmploymentDate, string? Login, string? Password) : base(Surname, Name, MiddleName, PhoneNumber)

{

this.PassportData = PassportData;

this.PositionID = PositionID;

this.BrigadeID = BrigadeID;

this.EmploymentDate = EmploymentDate;

this.Login = Login;

this.Password = Password;

}

public override string GetFullName()

{

if(BrigadeID != null )

{

return "Исполнитель: " + base.GetFullName();

}

return "Заявку принял: " + base.GetFullName();

}

...

using System.Windows;

using CleaningDLL.Entity;

using System.Collections.Generic;

using System;

namespace WPFCleaning.AdminFolder.ApplicationsFolder

{

/// <summary>

/// Логика взаимодействия для ContractWindow.xaml

/// </summary>

public partial class ContractWindow : Window

{

public ContractWindow(Order order)

{

InitializeComponent();

GetContractInfo(order);

}

private void GetContractInfo(Order order)

{

Client client = order.Client;

Employee admin = order.Employee;

Employee brigadir = Employee.GetBrigadirByBrigada(order.BrigadeID);

decimal finalPrice = order.FinalPrice;

DateTime dateTime = Contract.GetContractById(order.ContractID).DateOfContract;

int orderNumber = order.ID;

List<Human> people = new List<Human>();

people.Add(client);

people.Add(admin);

people.Add(brigadir);

string result = "";

foreach(var human in people)

{

result += ("\n\n" + human.GetFullName());

}

result += "\n\n\nНомер заявки: " + orderNumber;

result += "\n\nСтоимость оказываемых услуг: " + finalPrice;

result += "\n\nДата оформления: " + dateTime;

ContracrTextBox.Text = result;

}

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

{

this.Close();

}

}

}

**Приложение Ж**

(обязательное)

**Контекст БД**

using Microsoft.EntityFrameworkCore;

using CleaningDLL.Entity;

using System;

namespace CleaningDLL

{

public class ApplicationContext : DbContext

{

public DbSet<Address> Address { get; set; }

public DbSet<Order> Order { get; set; }

public DbSet<Brigade> Brigade { get; set; }

public DbSet<Client> Client { get; set; }

public DbSet<Consumable> Consumable { get; set; }

public DbSet<ConsumptionRate> ConsumptionRate { get; set; }

public DbSet<Contract> Contract { get; set; }

public DbSet<Delivery> Delivery { get; set; }

public DbSet<DeliveryContent> DeliveryContent { get; set; }

public DbSet<DeliveryContract> DeliveryContract { get; set; }

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

public DbSet<Inventory> Inventory { get; set; }

public DbSet<InventoryType> InventoryType { get; set; }

public DbSet<Position> Position { get; set; }

public DbSet<ProvidedService> ProvidedService { get; set; }

public DbSet<Provider> Provider { get; set; }

public DbSet<PurchaseRequisition> PurchaseRequisition { get; set; }

public DbSet<ReferenceUnitsOfMeasurement> ReferenceUnitsOfMeasurement { get; set; }

public DbSet<RequisitionContent> RequisitionContent { get; set; }

public DbSet<Service> Service { get; set; }

public DbSet<BrigadeInventory> BrigadeInventory { get; set; }

public DbSet<ConsumablesService> ConsumablesService { get; set; }

public ApplicationContext(DbContextOptions<ApplicationContext> options) : base(options)

{

Database.Migrate();

Context.AddDb(this);

}

public static DbContextOptions<ApplicationContext> GetDb()

{

var optionsBuilder = new DbContextOptionsBuilder<ApplicationContext>();

return optionsBuilder.UseNpgsql("Host=localhost;Port=5432;Database=Cleaning;Username=postgres;Password=qwertyuiop228").Options;

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Ignore<Human>();

modelBuilder.Entity<Employee>().HasIndex(s => s.PhoneNumber).IsUnique();

modelBuilder.Entity<Employee>().HasIndex(s => s.PassportData).IsUnique();

modelBuilder.Entity<Client>().HasIndex(s => s.PhoneNumber).IsUnique();

modelBuilder.Entity<Provider>().HasIndex(s => s.CompanyName).IsUnique();

modelBuilder.Entity<Provider>().HasIndex(s => s.ProviderTelefonNumber).IsUnique();

modelBuilder.Entity<ReferenceUnitsOfMeasurement>().HasIndex(s => s.Unit).IsUnique();

modelBuilder.Entity<Consumable>().HasIndex(s => s.ConsumableName).IsUnique();

modelBuilder.Entity<Employee>().HasIndex(s => s.Login).IsUnique();

modelBuilder.Entity<Employee>().HasIndex(s => s.Password).IsUnique();

modelBuilder.Entity<Position>(EntityConfigure.PositionConfigure);

modelBuilder.Entity<Brigade>(EntityConfigure.BrigadeConfigure);

modelBuilder.Entity<Employee>(EntityConfigure.EmployeeConfigure);

modelBuilder.Entity<InventoryType>(EntityConfigure.InventoryTypeConfigure);

modelBuilder.Entity<Service>(EntityConfigure.ServiceConfigure);

}

}

}

**Приложение З**

(обязательное)

**Миграции**

using Microsoft.EntityFrameworkCore.Migrations;

namespace CleaningDLL.Migrations

{

public partial class contractIDAdded : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.AddColumn<int>(

name: "ContracrID",

table: "Order",

type: "integer",

nullable: false,

defaultValue: 0);

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropColumn(

name: "ContracrID",

table: "Order");

}

}

}

using Microsoft.EntityFrameworkCore.Migrations;

namespace CleaningDLL.Migrations

{

public partial class compozition : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.AddColumn<int>(

name: "ContractID",

table: "Order",

type: "integer",

nullable: true);

migrationBuilder.CreateIndex(

name: "IX\_Order\_ContractID",

table: "Order",

column: "ContractID");

migrationBuilder.AddForeignKey(

name: "FK\_Order\_Contract\_ContractID",

table: "Order",

column: "ContractID",

principalTable: "Contract",

principalColumn: "ID",

onDelete: ReferentialAction.Restrict);

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropForeignKey(

name: "FK\_Order\_Contract\_ContractID",

table: "Order");

migrationBuilder.DropIndex(

name: "IX\_Order\_ContractID",

table: "Order");

migrationBuilder.DropColumn(

name: "ContractID",

table: "Order");

}

}

}

using Microsoft.EntityFrameworkCore.Migrations;

namespace CleaningDLL.Migrations

{

public partial class rename : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.RenameColumn(

name: "Smena\_Number",

table: "Brigade",

newName: "SmenaNumber");

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.RenameColumn(

name: "SmenaNumber",

table: "Brigade",

newName: "Smena\_Number");

}

}

}

**Приложение И**

(обязательное)

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

using System;

using System.Windows;

using System.Windows.Controls;

using CleaningDLL.Entity;

using System.Collections.Generic;

using System.Linq;

//Класс реализующий расчет стоимости и времени оказания услуг

namespace WPFCleaning.Admin

{

public static class OrderPrice

{

public static int[,] arrayService = new int[2, 7];

public static void Calculate(NewApplication newApplication, ClientPage clientPage)

{

Array.Clear(arrayService, 0, arrayService.Length);

arrayService[0, 1] = Service.GetIdService("Мойка окон");

arrayService[0, 2] = Service.GetIdService("Мойка стеклянных дверей");

arrayService[0, 3] = Service.GetIdService("Химчистка диванов");

arrayService[0, 4] = Service.GetIdService("Химчистка кресел");

arrayService[0, 5] = Service.GetIdService("Химчистка ковров");

arrayService[0, 6] = Service.GetIdService("Дезинфекция");

newApplication.PriceBox.Text = "";

newApplication.ApproximateTime.Text = "";

newApplication.finalPrice = 0;

newApplication.approximateTime = 0;

List<CheckBox> checkBoxes = GetCheckBoxIsChecked(newApplication);

if(checkBoxes.Count != 0)

{

if (checkBoxes[0].Name == newApplication.CheckExpressClean.Name ||

checkBoxes[0].Name == newApplication.CheckGeneralClean.Name ||

checkBoxes[0].Name == newApplication.CheckBuildingClean.Name ||

checkBoxes[0].Name == newApplication.CheckOfficeClean.Name)

{

if (newApplication.TextBoxSquare.Text != "")

{

arrayService[0, 0] = (Service.GetIdService(checkBoxes[0].Content.ToString()));

arrayService[1, 0] = Convert.ToInt32(newApplication.TextBoxSquare.Text);

newApplication.finalPrice += Service.GetServiceById(Service.GetIdService(checkBoxes[0].Content.ToString())).Price

\* Convert.ToInt32(newApplication.TextBoxSquare.Text);

newApplication.approximateTime += Service.GetServiceById(Service.GetIdService(checkBoxes[0].Content.ToString())).Time

\* Convert.ToInt32(newApplication.TextBoxSquare.Text);

}

else

{

checkBoxes[0].IsChecked = false;

MessageBox.Show("Введите площадь");

checkBoxes.RemoveAll(e => e.IsChecked.GetValueOrDefault());

}

checkBoxes.RemoveAt(0);

}

decimal tmpPrice = newApplication.finalPrice;

int tmpApproximateTime = newApplication.approximateTime;

foreach (var checkBox in checkBoxes)

{

newApplication.finalPrice = tmpPrice;

newApplication.approximateTime = tmpApproximateTime;

List<int> quantityService = GetTextFromTextBox(newApplication);

for (int i = 1; i <= 6; i++)

{

arrayService[1, i] = quantityService[i - 1];

newApplication.finalPrice += quantityService[i - 1] \* Service.GetServiceById(arrayService[0, i]).Price;

newApplication.approximateTime += quantityService[i - 1] \* Service.GetServiceById(arrayService[0, i]).Time;

}

}

if (clientPage.CheckOldClient.IsChecked.GetValueOrDefault())

{

newApplication.finalPrice = Convert.ToInt32((newApplication.finalPrice \* 90) / 100);

}

newApplication.PriceBox.Text = Order.GetPriceByInt(Convert.ToInt32(newApplication.finalPrice));

newApplication.ApproximateTime.Text = Order.GetTimeByInt(newApplication.approximateTime);

}

}

public static List<CheckBox> GetCheckBoxIsChecked(NewApplication newApplication)

{

List<CheckBox> list = new List<CheckBox>();

list.Add(newApplication.CheckExpressClean);

list.Add(newApplication.CheckGeneralClean);

list.Add(newApplication.CheckBuildingClean);

list.Add(newApplication.CheckOfficeClean);

list.Add(newApplication.ChemistryClean);

list.Add(newApplication.WindowClean);

list.Add(newApplication.Dezinfection);

return list.Where(c => c.IsChecked.GetValueOrDefault()).ToList();

}

public static List<int> GetTextFromTextBox(NewApplication newApplication)

{

List<int> list = new List<int>();

list.Add(Convert.ToInt32(newApplication.KolvoWindow.Text));

list.Add(Convert.ToInt32(newApplication.KolvoDoor.Text));

list.Add(Convert.ToInt32(newApplication.KolvoSofa.Text));

list.Add(Convert.ToInt32(newApplication.KolvoArmcheir.Text));

list.Add(Convert.ToInt32(newApplication.KolvoCarpet.Text));

list.Add(Convert.ToInt32(newApplication.KolvoDezinfection.Text));

return list;

}

}

}

using System;

using System.Windows;

using CleaningDLL.Entity;

using CleaningDLL;

namespace WPFCleaning.Admin

{

public static class NewOrder

{

public static void AddOrder(NewApplication newApplication, ClientPage clientPage, Employee emp)

{

Client client;

if (Client.ClientByTelefonIsNew(clientPage.Telefon.Text))

{

client = new Client(clientPage.Surname.Text, clientPage.Name.Text, clientPage.MiddleName.Text, clientPage.Telefon.Text, false);

}

else client = Client.GetClientByTelefon(clientPage.Telefon.Text);

var address = new Address(clientPage.Street.Text, clientPage.HouseNumber.Text, clientPage.Building.Text, clientPage.Entrance.Text,

clientPage.Apartment\_Number.Text);

if (newApplication.IsCorrectData())

{

string NewDate = (newApplication.DatePicker.Text + " " + newApplication.SelectTime.Text);

Order order;

if (DateTime.Parse(NewDate) > DateTime.Now)

{

if (DateTime.Parse(NewDate) > DateTime.Now)

{

order = new Order(EnumStatus.GetDescription(EnumStatus.Status.wait), client, emp, address,

Brigade.GetBrigadeByID(Convert.ToInt32(newApplication.BrigadeBox.Text)), DateTime.Parse(NewDate),

Order.GetPriceByString(newApplication.PriceBox.Text), newApplication.approximateTime,

newApplication.Comment.Text);

Context.Db.Order.Add(order);

for (int i = 0; i < 7; i++)

{

if (newApplication.arrayService[1, i] != 0)

{

var providedService = new ProvidedService(order, Service.GetServiceById(newApplication.arrayService[0, i]),

newApplication.arrayService[1, i]);

Context.Db.ProvidedService.Add(providedService);

}

}

Context.Db.Address.Add(address);

Context.Db.SaveChanges();

MessageBox.Show("Успешно!");

newApplication.ClearNewApplication();

clientPage.ClearClientInfo();

}

else MessageBox.Show("Что-то пошло не так :(");

}

else MessageBox.Show("Некорректная дата");

}

}

}

}

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using CleaningDLL.Entity;

using System.Linq;

using CleaningDLL;

using WPFCleaning.AdminFolder.ApplicationsFolder;

namespace WPFCleaning.Admin

{

/// <summary>

/// Класс реализующий отображение полной информации о заявке

/// </summary>

public partial class ApplicationsFullInfo : Window

{

private Applications \_applications;

Order order;

public int idService;

public decimal finalPrice = 0;

public int approximateTime = 0;

public ApplicationsFullInfo(int id, Applications a)

{

order = Order.GetOrderById(id); //выбранная заявка

\_applications = a; //страница вызова

InitializeComponent();

WindowState = WindowState.Maximized;

GetStatus();

StatusBox.IsEnabled = false;

SaveUpdatedOrder.IsEnabled = false;

BrigadeBox.IsEnabled = false;

DatePicker.IsEnabled = false;

SelectTime.IsEnabled = false;

BtnBrigadeInfo.IsEnabled = false;

Comment.IsEnabled = false;

UpdateOrder.IsEnabled = true;

}

public void AddSelectedOrder()

{

Telefon.Text = order.Client.PhoneNumber;

Surname.Text = order.Client.Surname;

Name.Text = order.Client.Name;

if (order.Client.MiddleName != null) MiddleName.Text = order.Client.MiddleName;

if (order.Client.IsOldClient) CheckOldClient.IsChecked = true;

Street.Text = order.Address.Street;

HouseNumber.Text = order.Address.HouseNumber;

Building.Text = order.Address.Building;

Entrance.Text = order.Address.Entrance;

Apartment\_Number.Text = order.Address.Apartment\_Number;

PriceBox.Text = Order.GetPriceByInt(order.FinalPrice);

ApproximateTime.Text = Order.GetTimeByInt(order.ApproximateTime);

StatusBox.Text = order.Status;

DatePicker.Text = order.Date.ToString("d");

SelectTime.Text = order.Date.ToString("t");

if(!int.TryParse(SelectTime.Text.Substring(0,2),out int i))

{

SelectTime.Text = "0" + SelectTime.Text;

}

BrigadeBox.Text = order.BrigadeID.ToString();

Employee brigadir = Employee.GetBrigadirByBrigada(order.BrigadeID);

BrigadirTelefon.Text = brigadir.PhoneNumber;

BrigadirSurname.Text = brigadir.Surname;

BrigadirName.Text = brigadir.Name;

BrigadirMiddleName.Text = brigadir.MiddleName;

BrigadeNumber.Text = order.BrigadeID.ToString();

Comment.Text = order.Comment;

CheckService.GetCheckedService(order.ID, this);

}

private void LockSelection(object sender, EventArgs e)

{

if (sender is CheckBox)

((CheckBox)sender).IsChecked = !((CheckBox)sender).IsChecked;

}

public void SaveUpdatedOrder\_Click(object sender, RoutedEventArgs e)

{

string NewDate = (DatePicker.Text + " " + SelectTime.Text);

if (DateTime.Parse(NewDate) > DateTime.Now)

{

order.Status = StatusBox.Text;

order.Brigade = Brigade.GetBrigadeByID(Convert.ToInt32(BrigadeBox.Text));

order.Date = DateTime.Parse(NewDate);

order.FinalPrice = Order.GetPriceByString(PriceBox.Text);

order.Comment = Comment.Text;

Context.Db.SaveChanges();

MessageBox.Show("Заявка изменена успешно!");

if(\_applications != null)

\_applications.SelectedOrderInfo();

UpdateOrder.IsEnabled = true;

this.Close();

}

else MessageBox.Show("Некорректная дата");

}

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

{

if (StatusBox.Text == "Завершена" || StatusBox.Text == "Отменена")

{

MessageBox.Show("Статус заявки изменить нельзя!");

}

else

{

StatusBox.IsEnabled = true;

SaveUpdatedOrder.IsEnabled = true;

BrigadeBox.IsEnabled = true;

DatePicker.IsEnabled = true;

SelectTime.IsEnabled = true;

BtnBrigadeInfo.IsEnabled = true;

Comment.IsEnabled = true;

UpdateOrder.IsEnabled = false;

}

}

public void GetStatus()

{

var statuses = EnumStatus.GetStatusesForAdmin(order.Status);

StatusBox.ItemsSource = statuses;

}

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

{

if (BrigadeBox.SelectedIndex != -1)

{

BrigadeInfoWindow brigadeInfoWindow = new BrigadeInfoWindow(Employee.GetEmployeeBrigade(int.Parse(BrigadeBox.Text)));

brigadeInfoWindow.Show();

}

else MessageBox.Show("Выберите бригаду");

}

private void SelectTime\_PreviewKeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Space)

{

e.Handled = true; // если пробел, отклоняем ввод

}

}

private void SelectTime\_TextChanged(object sender, TextChangedEventArgs e)

{

ChangeOrder.TextChanged(this);

}

private void SelectTime\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

ChangeOrder.PreviewTextInput(this, e);

}

}

}

Приложение К

(справочное)

**Библиографический список**

1. Entity Framework [электронный ресурс] // обновлено 02.19.2020 URL: <https://docs.microsoft.com/en-us/aspnet/entity-framework/> (дата обращения 24.11.2021).
2. Entity Framework [электронный ресурс] // Википедия — свободной энциклопедия URL: <https://en.wikipedia.org/wiki/Entity_Framework/> (дата обращения 24.11.2021).
3. Руководство в Entity Framework [электронный ресурс] // Последнее обновление 31.10.2015 URL: <https://metanit.com/sharp/efcore/> (дата обращения 24.11.2021).
4. Управление схемой БД и миграции [электронный ресурс] // Последнее обновление 24.11.2021 URL: <https://metanit.com/sharp/efcore/2.2.php> (дата обращения 02.12.2021).
5. Отношения между классами и объектами [электронный ресурс] // Последнее обновление 31.10.2015 URL: <https://metanit.com/sharp/patterns/1.2.php/> (дата обращения 24.11.2021).
6. Руководство по WPF [электронный ресурс] // Последнее обновление 13.11.2021 URL: <https://metanit.com/sharp/wpf/> (дата обращения 01.12.2021).