**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КОМИ**

**Государственное профессиональное образовательное учреждение**

**«Воркутинский политехнический техникум»**

**КУРСОВОЙ ПРОЕКТ**

По дисциплине МДК.05.02 Разработка кода

информационных систем

Разработка информационной системы «Планирование технического обслуживания станков в цеху»

Выполнил студент гр. ИСП-20 /\_\_\_\_\_\_\_\_\_\_\_\_\_/Оскретков Данил Васильевич/

(подпись) (Ф.И.О.)

**ОЦЕНКА:\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

**ПРОВЕРИЛ**

Научный руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_/ Егоров Данил Павлович/

(подпись) (Ф.И.О.)

Воркута

2023

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc133346751)

[ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ 4](#_Toc133346752)

[1.1 Платформа .NET 4](#_Toc133346753)

[1.2 Язык программирования С# 5](#_Toc133346754)

[1.3 Windows Presentation Foundation (WPF) 6](#_Toc133346755)

[1.4 СУБД SQL server 7](#_Toc133346756)

[1.5 Microsoft SQL Server Management Studio 9](#_Toc133346757)

[1.6 Entity Framework 10](#_Toc133346758)

[ГЛАВА 2. ПРОЕТИРОВАНИЕ БАЗЫ ДАННЫХ 12](#_Toc133346759)

[2.1 Разработка диаграммы ERD 12](#_Toc133346760)

[2.2 Разработка базы данных 13](#_Toc133346761)

[ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ 18](#_Toc133346762)

[3.1 Разработка интерфейса информационной системы 18](#_Toc133346763)

[3.2 Программирование информационной системы 27](#_Toc133346764)

[ЗАКЛЮЧЕНИЕ 32](#_Toc133346765)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 33](#_Toc133346766)

# ВВЕДЕНИЕ

Планирования технического обслуживания станков в цеху деятельности неразделимо с внедрением передовых технологий и процессов с целью обеспечения наиболее качественного обслуживания станков. Внедрение автоматизированных информационных систем – это не только тенденция времени, но и хорошая база для увеличения производительности и качества труда.

С помощью запросов реализованы функции поиска информации по заданной тематике, группировки информации для различных отчетов, редактирования и удаления записей в таблицах. Поиск станков, результатов тестирования, дате тестирования или соответствию некоторому разделу осуществляется с помощью запросов с параметром. Их удобно использовать в качестве источников записей для форм, отчетов и страниц доступа к данным.

**Объект** - информационная система «Планирования технического обслуживания станков в цеху».

**Предмет** - автоматизация бизнес-процесса «Планирования технического обслуживания станков в цеху»

**Цель:** создание информационной системы «Планирования технического обслуживания станков в цеху»

**Задачи:**

* выбор инструментария;
* спроектировать базу данных;
* разработать информационную систему.

# ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ

## Платформа .NET

Платформа .NET Framework — это технология, которая поддерживает создание и выполнение веб-служб и приложений Windows. При разработке платформы .NET Framework учитывались следующие цели.

* Обеспечение согласованной объектно-ориентированной среды программирования для локального сохранения и выполнения объектного кода, для локального выполнения кода, распределенного в Интернете, либо для удаленного выполнения.
* Предоставление среды выполнения кода, в которой:
  + сведена к минимуму вероятность конфликтов в процессе развертывания программного обеспечения и управления его версиями;
  + гарантируется безопасное выполнение кода, включая код, созданный неизвестным или не полностью доверенным сторонним изготовителем;
  + исключаются проблемы с производительностью сред выполнения скриптов или интерпретируемого кода;
* обеспечиваются единые принципы разработки для разных типов приложений, таких как приложения Windows и веб-приложения;
* обеспечивается взаимодействие на основе промышленных стандартов, которое гарантирует интеграцию кода платформы .NET Framework с любым другим кодом.

Платформа .NET Framework состоит из общеязыковой среды выполнения (среды CLR) и библиотеки классов .NET Framework. Основой платформы .NET Framework является среда CLR. Среду выполнения можно считать агентом, который управляет кодом во время выполнения и предоставляет основные службы, такие как управление памятью, управление потоками и удаленное взаимодействие. При этом средой накладываются условия строгой типизации и другие виды проверки точности кода, обеспечивающие безопасность и надежность. Фактически основной задачей среды выполнения является управление кодом. Код, который обращается к среде выполнения, называют управляемым кодом, а код, который не обращается к среде выполнения, называют неуправляемым кодом. Библиотека классов является комплексной объектно-ориентированной коллекцией повторно используемых типов, которые применяются для разработки приложений — начиная с обычных приложений, запускаемых из командной строки, и приложений с графическим интерфейсом (GUI) и заканчивая приложениями, использующими последние технологические возможности ASP.NET, такие как веб-формы и веб-службы XML.

Платформа .NET Framework может размещаться неуправляемыми компонентами, которые загружают среду CLR в собственные процессы и запускают выполнение управляемого кода, создавая таким образом программную среду, позволяющую использовать средства как управляемого, так и неуправляемого выполнения. Платформа .NET Framework не только предоставляет несколько базовых сред выполнения, но также поддерживает разработку базовых сред выполнения независимыми производителями.

Например, ASP.NET размещает среду выполнения и обеспечивает масштабируемую среду для управляемого кода на стороне сервера. ASP.NET работает непосредственно со средой выполнения, чтобы обеспечить выполнение приложений ASP.NET и веб-служб XML, обсуждаемых ниже в этой статье.

Обозреватель Internet Explorer может служить примером неуправляемого приложения, размещающего среду выполнения (в виде расширений типов MIME). Размещение среды выполнения в обозревателе Internet Explorer позволяет внедрять управляемые компоненты или элементы управления Windows Forms в HTML-документы. Такое размещение среды позволяет выполнять управляемый мобильный код и пользоваться его существенными преимуществами, в частности выполнением в условиях неполного доверия и изолированным хранением файлов.

## 1.2 Язык программирования С#

C# — объектно-ориентированный, ориентированный на компоненты язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО. В основном C# — объектно-ориентированный язык. Вы определяете типы и их поведение.

Программы C# выполняются в .NET, виртуальной системе выполнения, вызывающей общеязыковую среду выполнения (CLR) и набор библиотек классов. Среда CLR — это реализация общеязыковой инфраструктуры языка (CLI), являющейся международным стандартом, от корпорации Майкрософт. CLI является основой для создания сред выполнения и разработки, в которых языки и библиотеки прозрачно работают друг с другом.

Исходный код, написанный на языке C# компилируется в промежуточный язык (IL), который соответствует спецификациям CLI. Код на языке IL и ресурсы, в том числе растровые изображения и строки, сохраняются в сборке, обычно с расширением .dll. Сборка содержит манифест с информацией о типах, версии, языке и региональных параметрах для этой сборки.

При выполнении программы C# сборка загружается в среду CLR. Среда CLR выполняет JIT-компиляцию из кода на языке IL в инструкции машинного языка. Среда CLR также выполняет другие операции, например, автоматическую сборку мусора, обработку исключений и управление ресурсами. Код, выполняемый средой CLR, иногда называют "управляемым кодом". "Неуправляемый код" компилируется на машинный язык, предназначенный для конкретной платформы.

Обеспечение взаимодействия между языками является ключевой особенностью .NET. Код IL, созданный компилятором C#, соответствует спецификации общих типов (CTS). Код IL, созданный из кода на C#, может взаимодействовать с кодом, созданным из версий .NET для языков F#, Visual Basic, C++. Существует более 20 других языков, совместимых с CTS. Одна сборка может содержать несколько модулей, написанных на разных языках .NET, и все типы могут ссылаться друг на друга, как если бы они были написаны на одном языке.

В дополнение к службам времени выполнения .NET также включает расширенные библиотеки. Эти библиотеки поддерживают множество различных рабочих нагрузок. Они упорядочены по пространствам имен, которые предоставляют разные полезные возможности: от операций файлового ввода и вывода до управления строками и синтаксического анализа XML, от платформ веб-приложений до элементов управления Windows Forms. Обычно приложение C# активно используют библиотеку классов .NET для решения типовых задач.

## 1.3 Windows Presentation Foundation (WPF)

WPF существует в виде подмножества типов .NET, которые по большей части находятся в пространстве имен System.Windows. Если ранее вы создавали приложения в .NET с помощью таких платформ, как ASP.NET и Windows Forms, то должны быть знакомы с фундаментальными принципами программирования WPF. Вы выполняете следующие действия.

* Создание экземпляров классов
* Настройка свойств
* Вызов методов
* Обработка событий

WPF включает в себя дополнительные конструкции программирования, которые расширяют возможности свойств и событий: свойства зависимостей и перенаправленные события.

WPF позволяет разрабатывать приложения, используя как разметку, так и код программной части, что привычно для разработчиков на ASP.NET. Разметка XAML обычно используется для определения внешнего вида приложения, а управляемые языки программирования (код программной части) — для реализации его поведения. Такое разделение внешнего вида и поведения имеет ряд преимуществ.

* Затраты на разработку и обслуживание снижаются, так как разметка, определяющая внешний вид, не связана тесно с кодом, обуславливающим поведение.
* Повышается эффективность разработки, так как дизайнеры, занимающиеся внешним видом приложения, могут работать параллельно с разработчиками, реализующими поведение приложения.
* Глобализация и локализация приложений WPF упрощена.

XAML — это язык разметки на основе XML, который служит для определения внешнего вида приложения в декларативной форме. Обычно он используется для определения окон, страниц и пользовательских элементов управления, а также их заполнения элементами управления, фигурами и графическими элементами.

## 1.4 СУБД SQL server

СУБД SQL server используются для создания, размещения, хранения и управления реляционными (табличными) базами данных на специальных серверах или в облаке. Они работают через настольные приложения и web-сайты. К основным преимуществам их функционирования относятся:

* высокоскоростной доступ к данным, обеспечиваемый надежной клиент-серверной архитектурой СУБД;
* простота работы и администрирования, обусловленные понятной структурой языка программирования SQL;
* безопасность хранения информации в БД - благодаря возможности шифрования данных и резервного копирования.

Специфика работы сервера базы данных SQL server заключается в транзакционной обработке данных. Это означает, что по каждому запросу от СУБД обрабатывается и сохраняется небольшое количество информации.

Применение SQL server позволяет автоматизировать решение различных бизнес-задач, поддерживать проведение аналитики данных в режиме онлайн, отслеживать направление ресурсов СУБД, управлять транзакциями (операциями по обработке данных).

Система управления базами данных (СУБД) - это общий набор различных программных компонентов баз данных и собственно баз данных, содержащий следующие составляющие:

* прикладные программы баз данных;
* клиентские компоненты;
* серверы баз данных;
* собственно базы данных.

Прикладная программа баз данных представляет собой программное обеспечение специального назначения, разработанное и реализованное пользователями или сторонними компаниями-разработчиками ПО. В противоположность, клиентские компоненты - это программное обеспечение баз данных общего назначения, разработанное и реализованное компанией-разработчиком базы данных. С помощью клиентских компонентов пользователи могут получить доступ к данным, хранящимся на локальном или удаленном компьютере.

Сервер баз данных выполняет задачу управления данными, хранящимися в базе данных. Клиенты взаимодействуют с сервером баз данных, отправляя ему запросы. Сервер обрабатывает каждый полученный запрос и отправляет результаты соответствующему клиенту.

В общих чертах, базу данных можно рассматривать с двух точек зрения - пользователя и системы базы данных. Пользователи видят базу данных как набор логически связанных данных, а для системы баз данных это просто последовательность байтов, которые обычно хранятся на диске. Хотя это два полностью разных взгляда, между ними есть что-то общее: система баз данных должна предоставлять не только интерфейс, позволяющий пользователям создавать базы данных и извлекать или модифицировать данные, но также системные компоненты для управления хранимыми данными. Поэтому система баз данных должна предоставлять следующие возможности:

* разнообразные пользовательские интерфейсы;
* физическую независимость данных;
* логическую независимость данных;
* оптимизацию запросов;
* целостность данных;
* управление параллелизмом;
* резервное копирование и восстановление;
* безопасность баз данных

## 1.5 Microsoft SQL Server Management Studio

Среда SQL Server Management Studio предназначена для доступа к службам Службы Analysis Services, Integration Services и Reporting Services, а также для их настройки, администрирования и управления ими. Хотя все три технологии бизнес-аналитики полагаются на среду SQL Server Management Studio, административные задачи, связанные с каждой из этих технологий, несколько отличаются.

Управление решениями Analysis Services с помощью SQL Server Management Studio

Среда SQL Server Management Studio позволяет управлять объектами Службы Analysis Services, например выполнять их резервное копирование и обработку.

Среда Management Studio позволяет создавать проекты скриптов служб Службы Analysis Services, в которых выполняются разработка и сохранение скриптов с использованием многомерных выражений (MDX), расширений интеллектуального анализа данных (DMX) и XML для аналитики (XMLA). Проекты скриптов служб Службы Analysis Services используются для выполнения задач управления или повторного создания баз данных, кубов и других объектов в экземплярах служб Службы Analysis Services. Например, можно разработать скрипт XMLA в проекте скрипта служб Службы Analysis Services, который создает объекты непосредственно в существующем экземпляре служб Службы Analysis Services . Проекты скриптов служб Службы Analysis Services могут быть сохранены в составе решения и интегрироваться с контролем исходного кода.

Управление решениями Integration Services с помощью SQL Server Management Studio

Среда SQL Server Management Studio позволяет использовать службу Integration Services для управления пакетами и наблюдения за выполняющимися пакетами. В среде Среда Management Studio можно организовать пакеты в папки, выполнять, импортировать и экспортировать пакеты, переносить пакеты служб DTS и обновлять пакеты служб Integration Services.

Управление проектами Reporting Services с помощью SQL Server Management Studio

Среда SQL Server Management Studio позволяет включать компоненты служб Reporting Services, администрировать серверы и базы данных, управлять ролями и заданиями.

Она реализует функции управления общими расписаниями (в папке «Общие расписания») и базами данных сервера отчетов (ReportServer, ReportServerTempdb). Можно также создать роль RSExecRole в системной базе данных Master, когда база данных сервера отчетов перемещается в новое или другое ядро СУБД SQL Server.

## 1.6 Entity Framework

Entity Framework — это набор технологий в ADO.NET, которые поддерживают разработку программных приложений, ориентированных на данные. Архитекторам и разработчикам приложений, ориентированных на обработку данных, приходится учитывать необходимость достижения двух совершенно различных целей. Они должны моделировать сущности, связи и логику решаемых бизнес-задач, а также работать с ядрами СУБД, используемыми для сохранения и получения данных. Данные могут распределяться по нескольким системам хранения данных, в каждой из которых применяются свои протоколы, но даже в приложениях, работающих с одной системой хранения данных, необходимо поддерживать баланс между требованиями системы хранения данных и требованиями написания эффективного и удобного для обслуживания кода приложения.

Платформа Entity Framework позволяет работать с данными в форме специфических для домена объектов и свойств (например, с клиентами и их адресами) без необходимости учитывать формат базовых таблиц и столбцов базы данных, где хранятся эти данные. Entity Framework дает разработчикам возможность работать с данными на более высоком уровне абстракции, создавать и сопровождать приложения, ориентированные на работу с данными, одновременно с этим сокращая объем кода по сравнению с традиционными приложениями.

Многолетним и общим подходом к разработке является подход, при котором построение приложения или службы представляет собой его разделение на три части: модель домена, логическую модель и физическую модель. Модель домена определяет сущности и связи в моделируемой системе. Логическая модель для реляционной базы данных обеспечивает нормализацию сущностей и связей в целях создания таблиц с ограничениями внешнего ключа. В физической модели учитываются возможности конкретной системы обработки данных путем определения зависящих от ядра базы данных подробных сведений о хранении данных, которые касаются секционирования и индексирование.

Физическая модель совершенствуется администраторами базы данных в целях повышения производительности, но программисты, которые разрабатывают код приложения, в основном вынуждены ограничиваться работой с логической моделью, подготавливая SQL-запросы и вызывая хранимые процедуры. Модели домена в основном используются как инструмент для представления и обмена мнениями о требованиях к приложению, поэтому чаще всего служат в качестве практически не изменяющихся схем, которые рассматриваются и обсуждаются на ранних стадиях проекта, после чего выходят из сферы внимания. Во многих коллективах разработчиков принято пропускать этап создания концептуальной модели и начинать с определения таблиц, столбцов и ключей в реляционной базе данных.

Entity Framework дает жизнь модели, позволяя разработчикам запрашивать сущности и связи в модели предметной области (называемой концептуальной моделью в Entity Framework), в то же время полагаться на Entity Framework преобразования этих операций в команды, относящиеся к источнику данных. Это позволяет отказаться от применения в приложениях жестко заданных зависимостей от конкретного источника данных.

При работе в режиме Code First концептуальная модель сопоставлена с режимом хранения в коде. Entity Framework может вычислять концептуальную модель на основе типов объектов и дополнительных конфигураций, которые вы определяете. Метаданные сопоставления формируются во время выполнения на основе сочетания определений типов домена и дополнительной информации о конфигурации, которая указана в коде. Entity Framework создает базу данных по мере необходимости на основе метаданных.

# ГЛАВА 2. ПРОЕТИРОВАНИЕ БАЗЫ ДАННЫХ

## 2.1 Разработка диаграммы ERD

Схема «сущность-связь» (также ERD или ER-диаграмма) — это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы. ER-диаграммы чаще всего применяются для проектирования и отладки реляционных баз данных в сфере образования, исследования и разработки программного обеспечения и информационных систем для бизнеса.

Базовыми понятиями ER-модели данных (ER — Entity— Relati-onship) являются сущность, атрибут и связь.

Каждая сущность должна иметь:

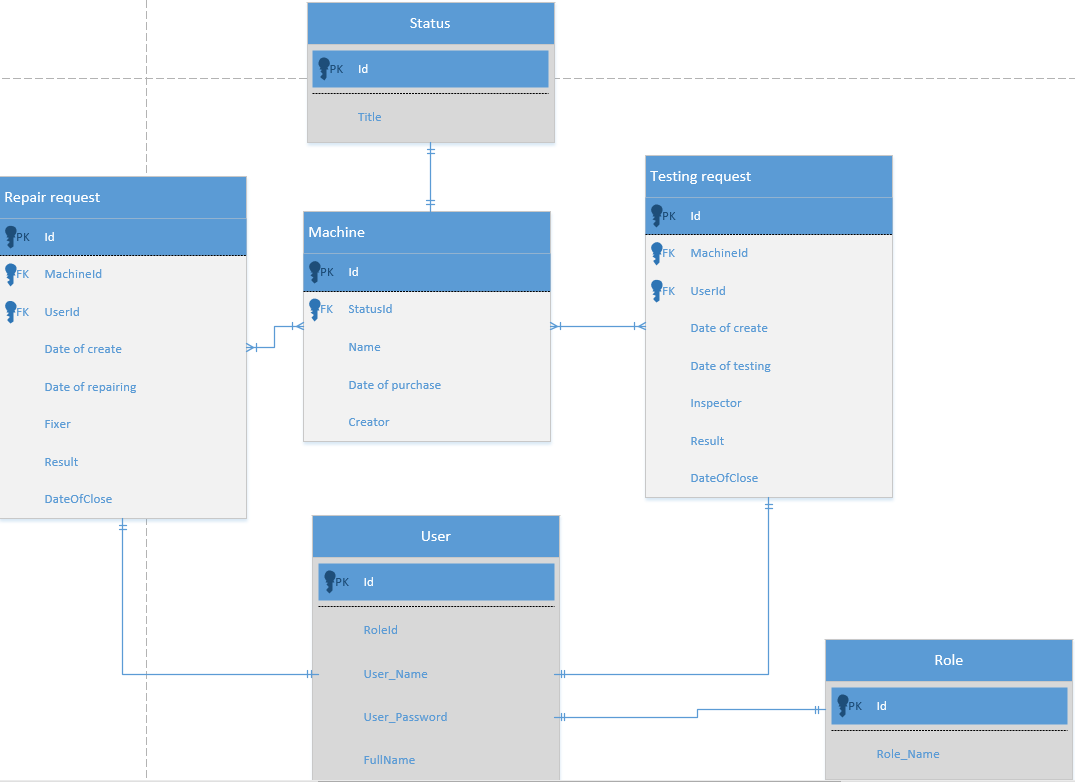
■ уникальное имя;

■ один или несколько атрибутов, которые либо принадлежат сущности, либо наследуются

через связь;

■ один или несколько атрибутов, которые однозначно идентифицируют каждый экземпляр сущности.

Основными сущностями являются: станок, заявка на тестирование, заявка на починку. Каждая сущность обделена атрибутами. У сущности «Станок» имеется: дата покупки, наименование, создатель, статус работы. У сущностей «Заявка на тестирование» и «Заявка на ремонт» атрибуты: номер станка, дата тестирования, дата создания заявки, создатель заявки, результат, дата закрытия заявки, инспектор у сущности “Заявка на тестирование” и ремонтник у сущности “Заявки на ремонт”. У сущности “Статус” атрибут: наименование. Так же имеется 2 сущности для хранения информации о пользователях информационной системы и хранения ролей для пользователей. Сущность “Пользователь” имеет такие атрибуты: наименование роли, ФИО, логин и пароль. Сущность “Роль” атрибут: наименование роли.

Рисунок 2.1 Диаграмма ERD

## 2.2 Разработка базы данных

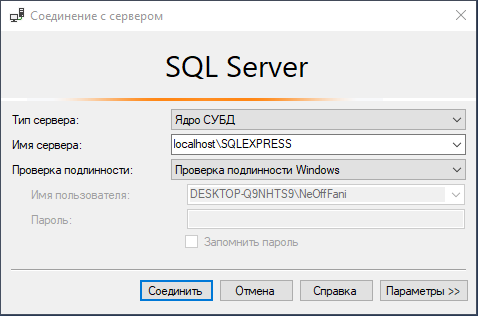
Для того что бы начать создания базы данных. Нужно запустить приложение “Microsoft SQL Server Management Studio” и подключить к серверу с именем сервера “localhost\SQLEXPRESS”.

Рисунок 2.2 Подключение к MSSMS

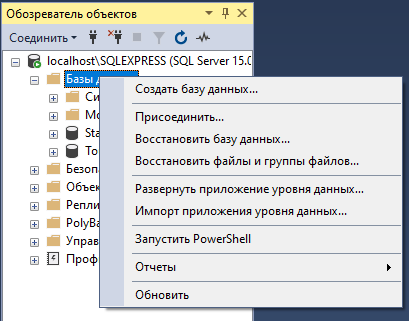
Далее в левой части экрана имеется панель “Обозреватель объектов”, нам нужно нажать правой кнопкой мыши по папке с названием “Базы данных” и выбрать пункт “Создать базу данных”.

Рисунок 2.3 Создание базы данных.

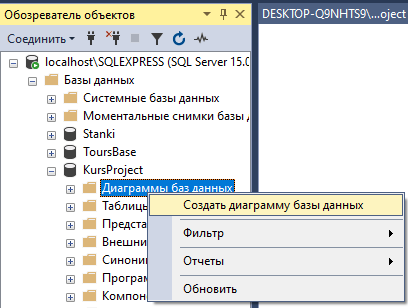
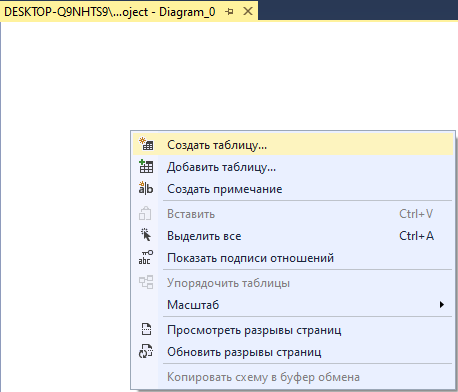
В обозреватели объектов появляется созданная нами база данных раскрываем её нажатием “+” слева от неё, нажимаем правой кнопкой мыши по папке “Диаграммы баз данных” и выбираем “Создать диаграмму базы данных”.

Рисунок 2.4 Создание диаграммы базы данных

Для того что бы создавать таблицы нашей базы данных, нужно нажать правой кнопкой мыши по пустому месту нашей диаграммы и выбрать пункт “Создать таблицу” и вводим название нашей таблицы.

Рисунок 2.5 Создание таблиц.

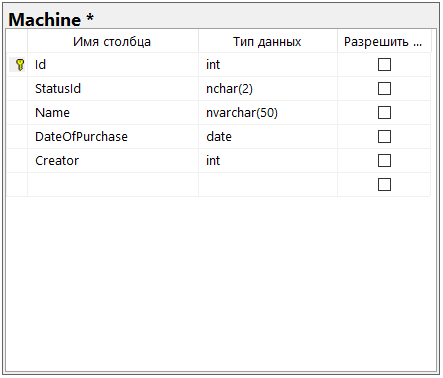
Создаётся первая таблица, в которой можно уже писать атрибуты и настраивать ключи. Вводим первые атрибуты и выбираем типы данных.

Рисунок 2.6 Атрибуты и типы данных.

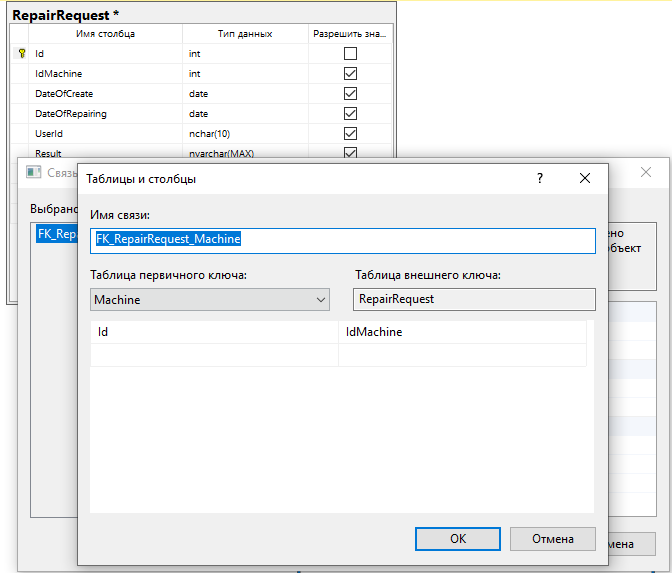
После создания таблиц, настраиваем связи. Зажимая ключ одной таблицы и ведём к полю другой таблицы с нужным атрибутом.

Рисунок 2.7 Создание свзяей

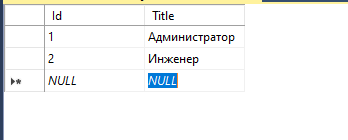
Как создали все таблицы, настроили все связи. Сохраняем диаграмму, выбираем таблицу и можно заполнять данные.

Рисунок 2.8 Заполнение таблиц.

# ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ

## 3.1 Разработка интерфейса информационной системы

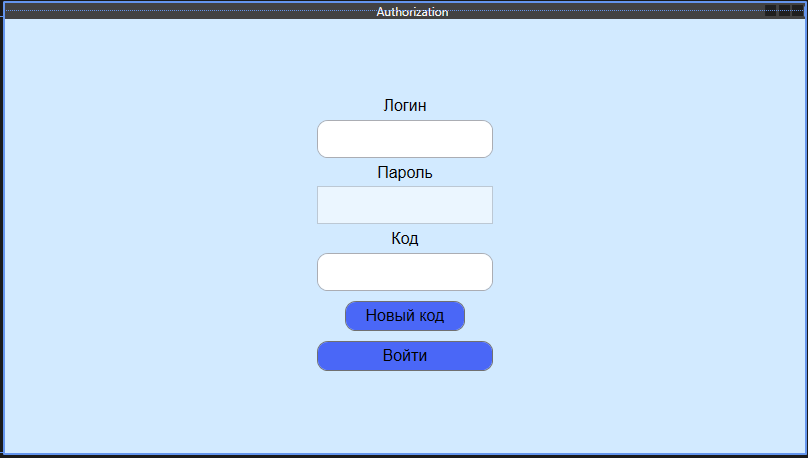
Окно авторизации имеет 3 поля для ввода данных. Поле логин изначально доступно для ввода информации. Поле пароль недоступно, пока пользователь не введёт логин и не пройдёт проверка на существование данного логина. Поле для ввода кода и кнопка, которая выдаёт новый код, не видимы для пользователя, пока не пройдёт проверка на правильность логина и пароля.

Рисунок 3.1 Окно авторизации.

<Grid Background="#FFD2EAFF">

<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">

<TextBlock TextWrapping="Wrap" Text="Логин"/>

<TextBox x:Name="TxbLogin" TextWrapping="Wrap" Text="" Width="176" KeyUp="Login\_KeyUp">

<TextBox.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</TextBox.Resources>

</TextBox>

<TextBlock TextWrapping="Wrap" Text="Пароль"/>

<PasswordBox x:Name="TxbPassword" Width="176" KeyUp="Password\_KeyUp" IsEnabled="False">

<PasswordBox.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</PasswordBox.Resources>

</PasswordBox>

<TextBlock Text="Код" Visibility="Collapsed" x:Name="CodeBlock"></TextBlock>

<TextBox x:Name="CodeBox" TextWrapping="Wrap" Text="" Width="176" Visibility="Collapsed">

<TextBox.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</TextBox.Resources>

</TextBox>

<Button Content="Новый код" x:Name="RefreshBtn" Click="Refresh" Width="120" Visibility="Collapsed">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

<Button Content="Войти" x:Name="Autorization" Click="Autorization\_Click" Width="176">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

</StackPanel>

</Grid>

Кнопка “Войти” делает переход на основное окно, которое поделено на 2 части. Верхняя часть окна имеет логотип и 3 кнопки переходов между страницами. Нижняя часть окна это сама область в которой отображаются страницы.

Рисунок 3.2 Основное окно.

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="70"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<StackPanel

HorizontalAlignment="Left"

Orientation="Horizontal"

VerticalAlignment="Center" Width="700" Height="40" Margin="100,0,0,0">

<Button x:Name="Machine" Content="Станки" Width="120" Click="Machine\_Click">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

<Button x:Name="Test" Content="Тестирование" HorizontalAlignment="Left" Width="120" Click="Test\_Click">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

<Button x:Name="Repair" Content="Ремонт" HorizontalAlignment="Left" Width="120" Click="Repair\_Click">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

</StackPanel>

<Frame x:Name="MyFrame" Grid.Row="1" Grid.ColumnSpan="2" NavigationUIVisibility="Hidden"/>

<Image Margin="20,0,0,0" Source="/pngwing.com.png" Stretch="Fill" Height="50" VerticalAlignment="Center" HorizontalAlignment="Left" Width="50"/>

</Grid>

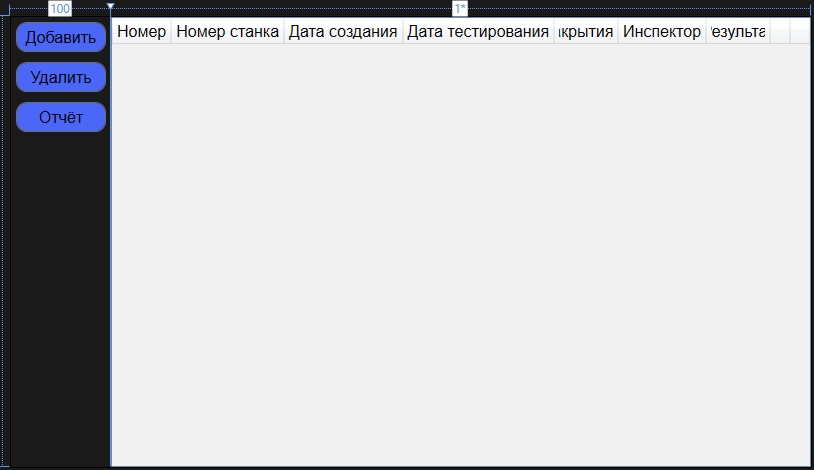
При нажатии любой из 3 кнопок в верхней части экрана, произойдёт отображение страницы. На рисунке (3.3) страница с таблицой “Тестирование” и 3 кнопки в левой части экрана для взаимодействия с этой таблицей, ещё есть 2 кнопки в самой таблице: “Редактировать” и “Закрыть заявку”. Каждая колонка таблицы привязана к атрибуту в базе данных. Кнопки слева от таблицы, скрыты для обычного пользователя, у инжинера нету кнопок: “Удалить” и “Отчёт”, а администратор видит все 3 кнопки.

Рисунок 3.3 Страница тестирование.

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="100"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<DataGrid

AutoGenerateColumns="False"

x:Name="TestGrid"

IsReadOnly="True" Grid.Column="2"

>

<DataGrid.Columns >

<DataGridTextColumn Header="Номер" Width="auto" Binding="{Binding Id}"/>

<DataGridTextColumn Header="Номер станка" Width="auto" Binding="{Binding IdMachine}"/>

<DataGridTextColumn Header="Дата создания" Width="auto" Binding="{Binding DateOfCreate, StringFormat='{}{0:dd.MM.yyyy}'}"/>

<DataGridTextColumn Header="Дата тестирования" Width="auto" Binding="{Binding DateOfTesting, StringFormat='{}{0:dd.MM.yyyy}'}"/>

<DataGridTextColumn Header="Дата закрытия заявки" Width="\*" Binding="{Binding DateOfClose, StringFormat='{}{0:dd.MM.yyyy}'}" />

<DataGridTextColumn Header="Инспектор" Width="auto" Binding="{Binding User.FullName}" />

<DataGridTextColumn Header="Результат" Width="\*" Binding="{Binding Result}">

<DataGridTextColumn.ElementStyle>

<Style TargetType="{x:Type TextBlock}">

<Setter Property="TextWrapping" Value="Wrap"/>

</Style>

</DataGridTextColumn.ElementStyle>

</DataGridTextColumn>

<DataGridTemplateColumn Width="auto">

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<Button Content="Редактировать" Name="BtnEdit" HorizontalAlignment="Left" Click="BtnEdit\_Click">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

<DataGridTemplateColumn Width="auto">

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<Button x:Name="Close" Content="Закрыть заявку" HorizontalAlignment="Left" Width="120" Click="Close\_Click">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

</DataGrid>

<StackPanel Grid.Row="0">

<Button x:Name="Add" Content="Добавить" Width="90" Click="Add\_Click">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

<Button x:Name="Delete" Content="Удалить" Width="90" Click="Delete\_Click" VerticalAlignment="Top">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

<Button Width="90" Content="Отчёт" Name="Export" Click="Export\_Click">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

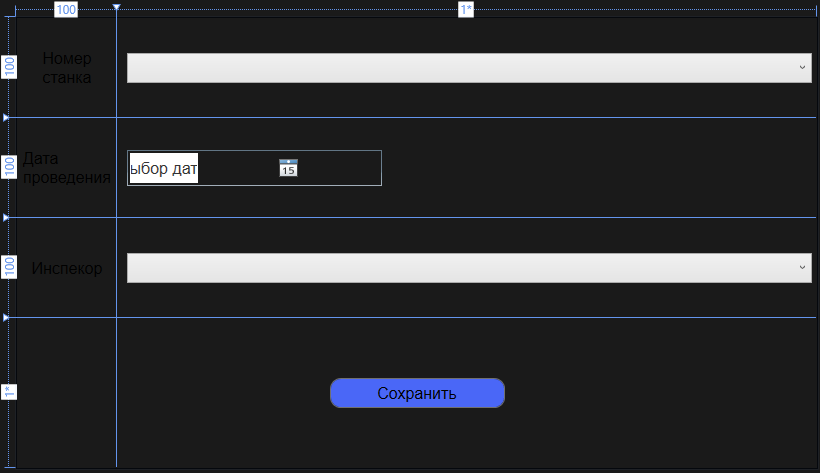
</Button.Resources>

</Button>

</StackPanel>

</Grid>

При нажатии кнопки “Добавить” открывается страница, вместо предыдущей. На ней имеются 3 элемента: 2 “ComboBox” и “DatePicker”. Комбо боксы нужны для выбора информации из других таблиц, для заполнения новой строки в таблице. Каждый элемент привязан к атрибуту в базе данных. При нажатие на кнопку “Редактировать”, появляется эта же страница, только с заполненными данными той строки, на которой была нажата кнопка. После нажатия кнопки сохранить, происходит проверка на заполненность полей, если какое то поле не заполненно, на экране появляется “MessageBox” с указанием на строку в которой не заполненное поле. Если все поля заполненны корректно, происходит добавление в базу данных.

Рисунок 3.4 Страница добавление и редактирования.

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="100"/>

<RowDefinition Height="100"/>

<RowDefinition Height="100"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="100"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<TextBlock TextWrapping="Wrap" HorizontalAlignment="Center" VerticalAlignment="Center" Text="Номер станка"/>

<TextBlock Grid.Row="1" TextWrapping="Wrap" HorizontalAlignment="Center" VerticalAlignment="Center"><Run Text="Дата"/><Run Language="ru-ru" Text=" проведения"/></TextBlock>

<TextBlock Grid.Row ="2" TextWrapping="Wrap" HorizontalAlignment="Center" VerticalAlignment="Center" Text="Инспекор"/>

<ComboBox SelectedItem="{Binding Machine}" Grid.Column="1" x:Name="CmbMachine" DisplayMemberPath="Id" Margin="10,0,5,0"/>

<DatePicker SelectedDate ="{Binding DateOfTesting}" Grid.Column="1" Margin="10,0,435,0" Grid.Row="1" DisplayDateStart="01.01.2023" VerticalAlignment="Center" Name="DateOfTesting"/>

<ComboBox SelectedItem="{Binding User}" Grid.Row="2" Grid.Column="1" x:Name="CmbUsers" DisplayMemberPath="FullName" Margin="10,0,5,0"/>

<Button Grid.ColumnSpan="2" Grid.Row="3" Content="Сохранить" HorizontalAlignment="Center" Name="AddEdit" Click="AddEdit\_Click">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

</Grid>

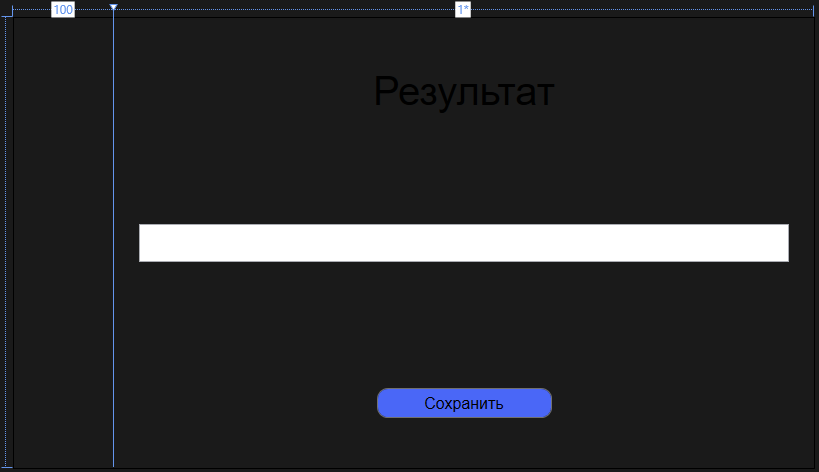
При нажатие кнопки “Закрыть заявку”, открывается страница. Которая имеет одно поле. Так же с привязкой к атрибуту в базе данных. Это поле так же имеет проверку на заполненность поля и при нажатии кнопки “Сохранить” сохраняет изменения в базе данных.

Рисунок 3.5 Страница закрытия заявки.

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="100"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<StackPanel VerticalAlignment="Center" HorizontalAlignment="Center" Orientation="Vertical" Width="700" Grid.Column="1">

<TextBox Text="{Binding Result}" TextWrapping="Wrap" x:Name="NameTxb" Width="650"/>

</StackPanel>

<TextBlock Grid.Column="1" HorizontalAlignment="Center" VerticalAlignment="Top" Margin="0 50 0 0" FontSize="40" TextWrapping="Wrap" Text="Результат" />

<Button Grid.Column="1" Content="Сохранить" HorizontalAlignment="Center" VerticalAlignment="Bottom" Margin="0 0 0 50" x:Name="Save" Click="Save\_Click">

<Button.Resources>

<Style TargetType="{x:Type Border}">

<Setter Property="CornerRadius" Value="10"/>

</Style>

</Button.Resources>

</Button>

</Grid>

На основе этих окон, были сделаны другие страницы для добавления/редактирования , страница закрытия заявок и страницы с таблицами данных.

## 3.2 Программирование информационной системы

Окно авторизации (рисунок 3.1) имеет такую логику.

Этот класс предназначен для хранения информации о авторизованном пользователи.

public static class Globals

{

public static int UserRole;

public static User userinfo { get; set; }

}

При нажатии на кнопку “Войти” срабатывает этот код, который делает проверку на соответствие логина, пароля и кода. Переводит на основное окно (рисунок 3.2) и передаёт информацию о авторизированном пользователе в класс Globals.

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

{

using (var db = new StankiEntities())

{

var auth = db.User.AsNoTracking().FirstOrDefault(m => m.Login == TxbLogin.Text && m.Password == TxbPassword.Password);

if (auth != null & code == CodeBox.Text)

{

timer.Stop();

Globals.UserRole = auth.RoleId;

Globals.userinfo = auth;

MainWindow main = new MainWindow();

main.Show();

Close();

}

else

{

MessageBox.Show("Неверный код, повторите попытку!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

timer.Stop();

}

}

}

Страница с заявками на тестирование (рисунок 3.3) имеет следующую логику.

Разграничение прав доступа, выглядит таким образом, что если авторизованный пользователь имеет роль с значением 1, ему видны все кнопки, роль с значением 2, не видит и не может взаимодействовать с кнопками: удаления и формирования отчёта и роль с значением 3, не видит и не может взаимодействовать со всеми кнопками левой части экрана.

public TestPage()

{

InitializeComponent();

if (Authorization.Globals.UserRole == 1)

{

Delete.Visibility = Visibility.Visible;

}

else if(Authorization.Globals.UserRole == 3)

{

Delete.Visibility = Visibility.Collapsed;

Add.Visibility = Visibility.Collapsed;

Export.Visibility = Visibility.Collapsed;

}

else

{

Export.Visibility = Visibility.Collapsed;

Delete.Visibility = Visibility.Collapsed;

}

}

При нажатие кнопки редактирования, происходит данный участок кода, который передаёт данные из таблицы в форму редактирования и заполняет все поля который связаны с помощью {Binding}.

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

{

Manager.MyFrame.Navigate(new AddEditTest((sender as Button).DataContext as TestRequests));

}

На данном участке кода, происходит удаление выбранных данных таблицы.

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

{

var testForRemoving = TestGrid.SelectedItems.Cast<TestRequests>().ToList();

if (MessageBox.Show($"Вы точно хотите удалить следующие {testForRemoving.Count()} элементов?", "Внимание",

MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

try

{

StankiEntities.GetContext().TestRequests.RemoveRange(testForRemoving);

StankiEntities.GetContext().SaveChanges();

MessageBox.Show("Данные удалены!");

TestGrid.ItemsSource = StankiEntities.GetContext().TestRequests.ToList();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString());

}

}

}

При нажатии на кнопку формирования отчёта. Выполняется данный участок кода, который создаёт новый Word документ, формирует в нём таблицу и заполняет её.

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

{

var allRequest = StankiEntities.GetContext().TestRequests.ToList();

var application = new Word.Application();

Word.Document document = application.Documents.Add();

Word.Paragraph userParagraph = document.Paragraphs.Add();

Word.Range userRange = userParagraph.Range;

userRange.Text = "Заявки на тестирование";

userRange.InsertParagraphAfter();

Word.Paragraph tableParagraph = document.Paragraphs.Add();

Word.Range tableRange = tableParagraph.Range;

Word.Table paymentsTable = document.Tables.Add(tableRange, allRequest.Count()+1, 6);

paymentsTable.Borders.InsideLineStyle = paymentsTable.Borders.OutsideLineStyle

= Word.WdLineStyle.wdLineStyleSingle;

paymentsTable.Range.Cells.VerticalAlignment = Word.WdCellVerticalAlignment.wdCellAlignVerticalCenter;

Word.Range cellRange;

cellRange = paymentsTable.Cell(1, 1).Range;

cellRange.Text = "Номер заявки";

cellRange = paymentsTable.Cell(1, 2).Range;

cellRange.Text = "Номер станка";

cellRange = paymentsTable.Cell(1, 3).Range;

cellRange.Text = "Дата Создания";

cellRange = paymentsTable.Cell(1, 4).Range;

cellRange.Text = "Создатель заявки";

cellRange = paymentsTable.Cell(1, 5).Range;

cellRange.Text = "Инженер";

cellRange = paymentsTable.Cell(1, 6).Range;

cellRange.Text = "Результат";

paymentsTable.Rows[1].Range.Bold = 1;

paymentsTable.Rows[1].Range.ParagraphFormat.Alignment = Word.WdParagraphAlignment.wdAlignParagraphCenter;

for (int i = 0; i < allRequest.Count(); i++)

{

var currentCategory = allRequest[i];

if (currentCategory.Result != null)

{

cellRange = paymentsTable.Cell(i + 2, 1).Range;

cellRange.Text = Convert.ToString(currentCategory.Id);

cellRange.ParagraphFormat.Alignment = Word.WdParagraphAlignment.wdAlignParagraphCenter;

cellRange = paymentsTable.Cell(i + 2, 2).Range;

cellRange.Text = Convert.ToString(currentCategory.IdMachine);

cellRange = paymentsTable.Cell(i + 2, 3).Range;

cellRange.Text = currentCategory.DateOfCreate.ToString("dd.MM.yyyy");

cellRange = paymentsTable.Cell(i + 2, 4).Range;

cellRange.Text = Convert.ToString(currentCategory.Creator);

cellRange = paymentsTable.Cell(i + 2, 5).Range;

cellRange.Text = Convert.ToString(currentCategory.User.FullName);

cellRange = paymentsTable.Cell(i + 2, 6).Range;

cellRange.Text = Convert.ToString(currentCategory.Result);

}

}

application.Visible = true;

}

Страница с добавлением/редактированием (рисунок 3.4) имеет следующую логику.

В данном участке кода происходит создания новой строки в базе данных.

private TestRequests \_currentTest = new TestRequests();

Если передаётся какое то значение “selectedTest” со страницы с заявками, идёт присвоение текущего теста к выбранному тесту.

public AddEditTest(TestRequests selectedTest)

{

InitializeComponent();

if (selectedTest != null)

{

\_currentTest = selectedTest;

}

else

{

\_currentTest.DateOfTesting = DateTime.Now;

}

DataContext = \_currentTest;

CmbMachine.ItemsSource = StankiEntities.GetContext().Machine.ToList();

CmbUsers.ItemsSource = StankiEntities.GetContext().User.ToList();

}

В данной части кода, происходит предотвращение ошибок, допущенных пользователем. Сохранение новой заявки или редактирование старой. Так же идёт автоматическое заполнение значения “Creator” и “DateOfCreate”. Поле создателя заявки заполняется на основе авторизированного пользователя, а дата создания заполняется на основе сегодняшней даты.

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

{

StringBuilder errors = new StringBuilder();

var CurrentMachine = CmbMachine.SelectedItem as Machine;

var CurrentUser = CmbUsers.SelectedItem as User;

\_currentTest.DateOfCreate = DateTime.Now;

var Creator = Authorization.Globals.userinfo.FullName;

\_currentTest.Creator = Creator;

if (CurrentMachine == null)

errors.AppendLine("Выберите станок");

if (DateOfTesting.SelectedDate == null)

errors.AppendLine("Выберите дату");

if (CurrentUser == null)

errors.AppendLine("Выберите инспектора");

if (errors.Length > 0)

{

MessageBox.Show(errors.ToString());

return;

}

if (\_currentTest.Id >= 0)

StankiEntities.GetContext().TestRequests.AddOrUpdate(\_currentTest);

try

{

StankiEntities.GetContext().SaveChanges();

MessageBox.Show("Информация сохранена!");

Manager.MyFrame.GoBack();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString());

}

Страница с закрытием заявки (рисунок 3.5) имеет такую же логику, как и редактирование заявки.

Остальные страницы сделаны по аналогии описанных.

### ЗАКЛЮЧЕНИЕ

Информационные системы очень важны в наше время. Они позволяют исключить ошибки при работе и сократить затраты времени. А также предоставляет своевременную и надёжную информацию, позволяет улучшить процесс принятия решения.

В ходе выполнения курсового проекта были выполнены следующие задачи:

* выбрал инструментария;
* спроектировал базу данных;
* разработал информационную систему.

Экономический эффект от внедрения автоматизированной информационной системы «Планирование обслуживания технического обслуживания станков в цеху» ожидается за счет сокращения затрат времени на выполняемые администратором и главным инженером, исключения ошибок при формировании отчетов, увеличения времени на анализ. При ручном создании отчетов могут быть допущены ошибки, правильно составленный алгоритм разрабатываемой программы экономит время и исключает ошибки.

Исходя из общего положения и опираясь на совокупность всех ранее вышеперечисленных и упомянутых фактов можно сделать вывод, что поставленные цель и задачи при разработки информационной системы были выполнены в полной мере.

Ссылка на репозиторий GitHub: https://github.com/NeOffFani/Kurs-Project

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Грин Дженнифер. Head First: изучаем С#/ Грин Дженнифер, Стиллмен Эндрю – Санкт-Питербург: 2022.
2. Мэтью Мак-Дональд WPF Windows Presentation Foundation в .NET 4.5 с примерами на C#: для профессионалов/ Мэтью Мак-Дональд: Санкт-Питербург, 2016.

Интернет-ресурсы

1. Документация по Windows Presentation Foundadion // Microsoft Learn. Точка доступа: <https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/?view=netdesktop-6.0>
2. Документация по C#. Руководства, справочные материал // Microsoft Learn. Точка доступа: <https://learn.microsoft.com/ru-ru/dotnet/csharp/>
3. WPF и C# | Полное руководство // Metanit. Точка доступа: <https://metanit.com/sharp/wpf/>
4. Документация по SQL Server // Microsoft Learn. Точка доступа: <https://learn.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver16>
5. Руководство по MS SQL Server // Metanit. Точка доступа: <https://metanit.com/sql/sqlserver/>