|  |  |
| --- | --- |
| Департамент образования Ярославской области  Государственное профессиональное образовательное  автономное учреждение Ярославской области  «ЯРОСЛАВСКИЙ ПРОМЫШЛЕННО-ЭКОНОМИЧЕСКИЙ КОЛЛЕДЖ им. Н.П. ПАСТУХОВА» | |
|  |  |
| **Отчет ПО УЧЕБНОЙ ПРАКТИКЕ**  **по профессиональному модулю ПМ.07**  **«Соадминистрирование и автоматизация БД и серверов»** | |
| ОУП ПМ.07.09.02.07.19ИП1.20 | |
|  | |
|  | |
|  | |
|  | |
|  | Студент  \_\_\_\_\_\_\_\_\_\_\_\_\_ Г.Г. Мымрин  «3» декабря 2022 г. |
|  | Преподаватель  \_\_\_\_\_\_\_\_\_\_\_\_\_ Е.О. Куксов  «3» декабря 2022 г. |
|  |  |
| 2022 | |

**Содержание:**

[**Введение** 3](#_Toc121690186)

[**Практическая работа № 1.1** 6](#_Toc121690187)

[**Практическая работа № 1.2** 11](#_Toc121690188)

[**Практическая работа № 1.3** 15](#_Toc121690189)

[**Практическая работа № 1.4** 22](#_Toc121690190)

[**Практическая работа № 1.5** 25](#_Toc121690191)

[**Практическая работа № 2** 29](#_Toc121690192)

[**Практическая работа № 3** 34](#_Toc121690193)

[**Практическая работа № 4** 40](#_Toc121690194)

[**Практическая работа № 5.1** 42](#_Toc121690195)

[**Практическая работа № 5.2** 43](#_Toc121690196)

[**Практическая работа № 6** 44](#_Toc121690197)

[**Практическая работа № 7** 47](#_Toc121690198)

[**Практическая работа № 8** 51](#_Toc121690199)

# **Введение**

Учебная практика направлена на:

­ формирование у студентов практических профессиональных умений, приобретение первоначального практического опыта;

­ закрепление теоретических знаний, полученными студентами в процессе обучения профессиональных модулей;

­ углубление первоначального профессионального опыта студента, развития общих и профессиональных компетенций, проверку его готовности к самостоятельной трудовой деятельности;

­ сбор, систематизация и обобщение практического материала (в том числе) для использования в выпускной квалификационной работе.

По окончании практики студент сдаёт отчет в соответствии с содержанием тематического плана практики и по форме, установленной в ГПОАУ ЯО «Ярославский промышленно-экономический колледж имени Н.П. Пастухова».

Итоговая аттестация проводится в форме - дифференцированного зачёта.

С целью овладения указанным видом профессиональной деятельности и соответствующими профессиональными компетенциями студент в ходе прохождения учебной практики должен:

Данные методические указания предназначены для студентов 4 курса специальности 09.02.07 «Информационные системы и программирование».

Методические указания содержат:

* требования к знаниям и умениям студентов;
* правила выполнения лабораторных работ (практических заданий);
* содержание лабораторных работ (практических заданий);
* контрольные вопросы;
* список использованной литературы.

В результате выполнения данного курса работ студент должен:

*Профессиональные компетенции*

ПК 7.1. Выявлять технические проблемы, возникающие в процессе эксплуатации баз данных и серверов

ПК 7.2. Осуществлять администрирование отдельных компонент серверов.

ПК 7.3. Формировать требования к конфигурации локальных компьютерных сетей и серверного оборудования, необходимые для работы баз данных и серверов.

ПК 7.4. Осуществлять администрирование баз данных в рамках своей компетенции.

ПК 7.5. Проводить аудит систем безопасности баз данных и серверов, с использованием регламентов по защите информации.

*Общие компетенции*

ОК 1. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 2. Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности.

ОК 3. Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях.

ОК 4. Эффективно взаимодействовать и работать в коллективе и команде

ОК 5. Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 6. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей, в том числе с учетом гармонизации межнациональных и межрелигиозных отношений, применять стандарты антикоррупционного поведения.

ОК 7. Содействовать сохранению окружающей среды, ресурсосбережению, применять знания об изменении климата, принципы бережливого производства, эффективно действовать в чрезвычайных ситуациях.

ОК 8. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.

ОК 9. Пользоваться профессиональной документацией на государственном и иностранном языках.

|  |  |  |
| --- | --- | --- |
| *Код*  *ПК, ОК* | *Умения* | *Знания* |
| ОК 1, 2, 3, 4, 5, 6, 7,8,9  ПК 7.1., 7.2., 7.3., 7.4., 7.5. | У1. проектировать и создавать базы данных;  выполнять запросы по обработке данных на языке SQL;  У2. осуществлять основные функции по администрированию баз данных;  У3. разрабатывать политику безопасности SQL сервера, базы данных и отдельных объектов базы данных;  У4. владеть технологиями проведения сертификации программного средства. | З1. модели данных, основные операции и ограничения;  З2. технологию установки и настройки сервера баз данных;  З3. требования к безопасности сервера базы данных;  З4. государственные стандарты и требования к обслуживанию баз данных |

# **Практическая работа № 1.1**

**Создание базы данных**

**Цель:** по заданной схеме данных разработать базу данных, создать и сохранить скрипт с БД.

**Задание:**

1. Изучите Пояснения к работе.
2. По диаграмме ERD, построенной на УП.05, с помощью СУБД MS SQL реализуйте базы данных.
3. Создайте скрипт базы данных.
4. Заполните базу данных первичными данными. (Если в базе данных нет таблицы с данными для авторизации, то добавьте ее).
5. Оформите отчет о проделанной работе (согласно плану отчета).

**Теоретический материал**

*База данных* - набор сведений, хранящихся некоторым упорядоченным способом.

*Система управления базами данных* — это совокупность языковых и программных средств, которая осуществляет доступ к данным, позволяет их создавать, менять и удалять, обеспечивает безопасность данных и т.д.

*SQL* - язык структурированных запросов, основной задачей которого является предоставление простого способа считывания и записи информации в базу данных.

Простейшая схема работы с базой данных выглядит примерно так:

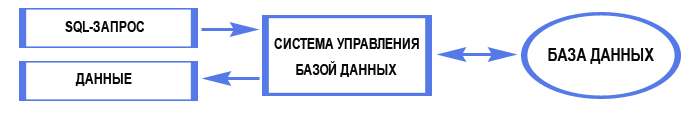


Рисунок 1. Схема работы с базой данных

В реляционной базе данных все данные можно представить в виде простых таблиц. Таблицы в реляционных базах данных обладают рядом свойств. Основными являются следующие:

* В таблице не может быть двух одинаковых строк.
* Столбцы располагаются в определенном порядке, который создается при создании таблицы. В таблице может не быть ни одной строки, но обязательно должен быть хотя бы один столбец.
* У каждого столбца есть уникальное имя (в пределах таблицы), и все значения в одном столбце имеют один тип (число, текст, дата...).
* На пересечении каждого столбца и строки может находиться только атомарное значение (одно значение, не состоящее из группы значений). Таблицы, удовлетворяющие этому условию, называют нормализованными.

*Первичный ключ* (сокращенно РК - primary key) - столбец, значения которого во всех строках различны.

В реляционных базах данных некоторая информация из одних таблиц присутствует в других, т.е. между ними имеются связи.

Всего существует 3 типа связей:

* [Один к одному](https://office-menu.ru/uroki-sql/41-tipy-svyazej-v-relyatsionnykh-bazakh-dannykh#onetoone);
* [Один ко многим](https://office-menu.ru/uroki-sql/41-tipy-svyazej-v-relyatsionnykh-bazakh-dannykh#onetomany);
* [Многие ко многим](https://office-menu.ru/uroki-sql/41-tipy-svyazej-v-relyatsionnykh-bazakh-dannykh#manytomany).

**Практическая часть**

В ходе выполнения данной практической работы при помощи программного продукта MS Visio, а также учебных материалов, была создана БД, представленная на рисунках ниже.

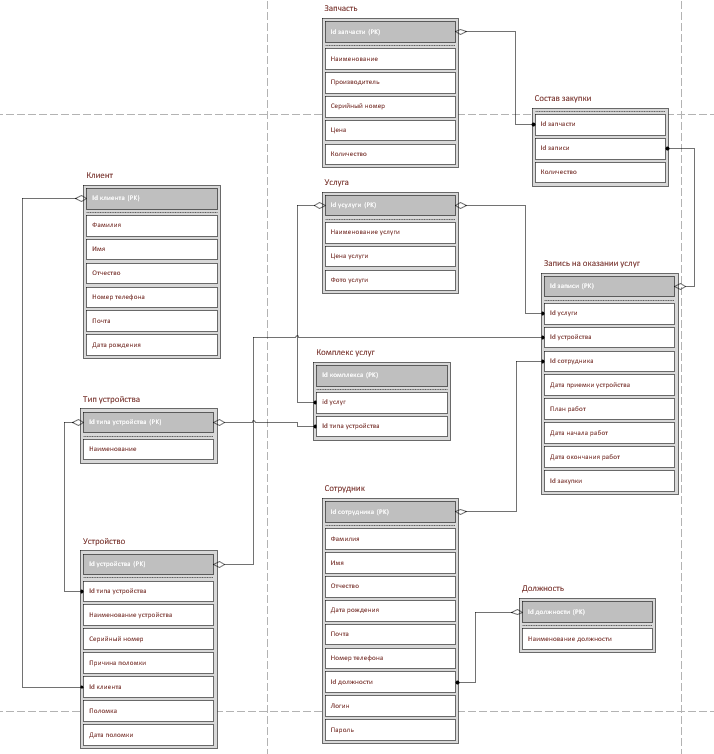


Рисунок 1 ERD

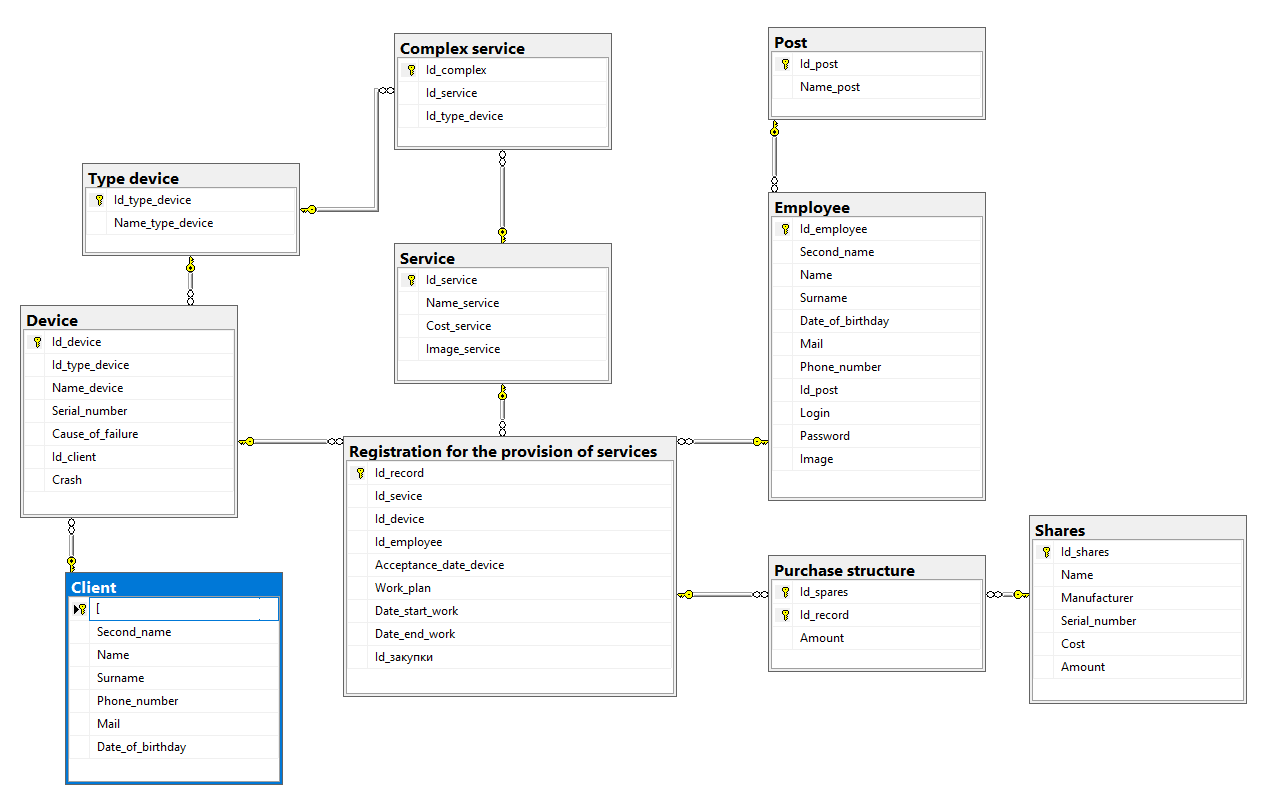


Рисунок 2 Диаграмма БД в SQL

Вывод: в ходе проделанной работы, была реализована схема БД в visio и добавлена бд в SQL

Контрольные вопросы

1. Что такое «база данных»?

База данных - набор сведений, хранящихся некоторым упорядоченным способом.

1. Что такое первичный ключ (Primary key)?

Первичный ключ (сокращенно РК - primary key) - столбец, значения которого во всех строках различны.

1. Что такое внешний ключ (Foreign key)?

Внешний ключ (англ. foreign key) — понятие теории реляционных баз данных, относящееся к ограничениям целостности базы данных.

1. Какие связи существуют в реляционных базах данных?

* Один к одному;
* Один ко многим;
* Многие ко многим.

1. Что подразумевается под целостностью данных?

Целостность базы данных (англ. database integrity) — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам.

1. Что такое нормализация и каковы ее преимущества?

Нормализация – это метод проектирования базы данных, который позволяет привести базу данных к минимальной избыточности.

Нормализация имеет целый ряд преимуществ:

* лучшая общая организация базы данных;
* сокращение числа ненужных повторений данных;
* согласованность данных внутри базы данных;
* более гибкая структура базы данных;
* эффективные возможности обеспечения безопасности и надежности базы данных.

# **Практическая работа № 1.2**

**Подключение базы данных.**

**Цель:** по заданной схеме данных разработать базу данных, создать и сохранить скрипт с БД, реализовать программное приложение с возможностью авторизации в системе.

**Задание:**

1. Изучите Пояснения к работе.
2. Создайте приложение (WPF) с помощью средств Visual Studio.
3. Оформите отчет о проделанной работе (согласно плану отчета).

**Теоретический материал**

*Windows Presentation Foundation* (WPF) — аналог WinForms, система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML.

В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление.

Преимущества WPF:

* Использование традиционных языков .NET-платформы - C# и VB.NET для создания логики приложения;
* Возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML, основанном на xml и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать XAML и C#/VB.NET;
* Независимость от разрешения экрана: поскольку в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением;
* Новые возможности, которых сложно было достичь в WinForms, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и др.;
* Хорошее взаимодействие с WinForms, благодаря чему, например, в приложениях WPF можно использовать традиционные элементы управления из WinForms;
* Богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое;
* Аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2D или 3D, графикой или текстом, все компоненты приложения транслируются в объекты, понятные Direct3D, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной;
* Создание приложений под множество ОС семейства Windows - от Windows XP до Windows 10.

Также стоит учитывать, что по сравнению с приложениями на Windows Forms объем программ на WPF и потребление ими памяти в процессе работы в среднем несколько выше. Но это с лихвой компенсируется более широкими графическими возможностями и повышенной производительностью при отрисовке графики.

**Практическая часть**

В ходе выполнения данной практической работы при помощи программного продукта Visual Studio, а также MS SQL, была подключена база данных к проекту, представленному на рисунках ниже.

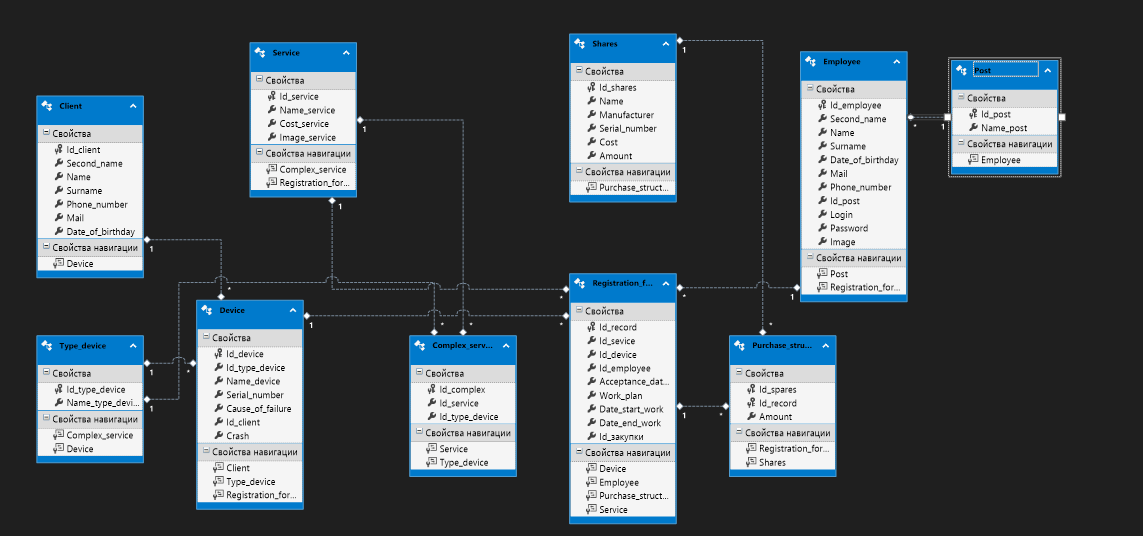


Рисунок 3 Модель БД в приложении

Реализация подключения БД в отдельно классе (Рисунок 4)

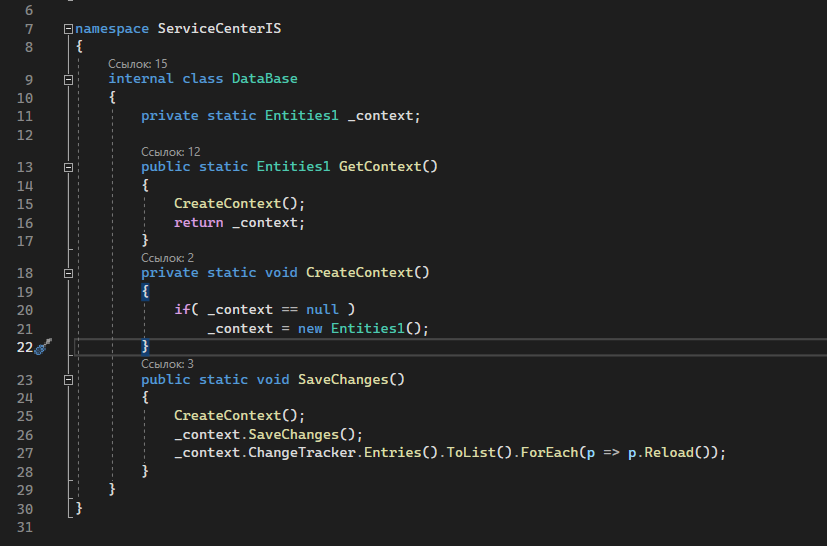


Рисунок 4 Класс DataBase

**Вывод:** в ходе проделанной работы было реализовано подключение к БД в приложении

**Контрольные вопросы**

1. Как создать подключение к базе данных?

Откроем среду разработки Visual Studio и создадим новый проект «WPF». В качестве названия проекта укажите наименование вашей предметной области + «IS». Нажмите кнопку «Ок». Следующим шагом будет подключение базы данных к проекту. Для этого на панели проекта нажмем правой кнопкой мыши по нашему проекту и выберем пункт «Добавить» -> «Создать новый элемент». Из списка элементов выберем элемент «Модель ADO.NET EDM». Зададим данному элементу название. Откроется мастер создания моделей EDM. Выбираем «Конструктор EF из базы данных» и жмем кнопку «Далее». В следующем окне нажимаем на кнопку «Создать соединение». В окне «Свойства подключения» укажите актуальные для вас данные подключения. Поменяйте «Проверка подлинности Windows» на «Проверка подлинности SQL». Выбор версии оставьте без изменений. В следующем окне укажите необходимые таблицы. Пространство имен может указать свое название, либо оставить без изменений. После завершения работы с мастером, система покажет вам схему данных вашей базы данных. Обратите внимание, как здесь отражены связи между сущностями (например, в связях «многие ко многим» отсутствует промежуточная таблица, которая связывала две сущности). Теперь нужно реализовать возможность работать с базой данных непосредственно из кода. Для этого создадим новый класс, например, с названием DataBase. В классе нам необходимо немного дописать код, чтобы с помощью данного класса мы могли иметь доступ к базе данных. Для этого: необходимо объявить статичную приватную переменную, которая будет являться объектом класса «Entities». Написать публичный статичный метод, который будет возвращать нам данный объект. В случае, если этот объект не был создан, необходимо создать его.

# **Практическая работа № 1.3**

**Создание руководства по стилю, логотипа, иконки. Размещение файлов в репозитории**

**Цель:** разместить скрипт с базой данных, руководство по стилю, логотип и иконку в удаленный репозиторий проекта.

**Задание:**

1. Изучите Пояснения к работе.
2. Создайте руководство по стилю для своего будущего проекта.
3. Подберите или нарисуйте логотип и иконку для своей темы.
4. Разместите скрипт с базой данных, руководство по стилю и иконку в удаленный репозиторий на ветку «PR1\_3».
5. Оформите отчет о проделанной работе.

**Теоретическая часть**

*Система контроля версий* — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Git — одна из распределенных систем контроля версий.

GitHub — один из сервисов для использования системы контроля версий Git.

Git — одна из систем контроля версий.

Git предназначена, в основном, для работы распределенной команды разработчиков.

То есть разработчики могут находиться в разных концах света и работать над одним проектом.

Система Git очень экономична и не требует рассылки большого количества файлов. Отслеживаются и пересылаются изменения в файлах и ссылки на эти изменения. То есть основная рассылка — это рассылка разницы в ваших редактированиях.

Отсылаются только различия в папках и файлах. В любой момент времени вы можете возвратиться к тому или иному состоянию системы. Многие компании уделяют внимание хорошей и быстрой коммуникации между сотрудниками. В этом отношении, система контроля версий предоставляет большие возможности. Всю мощь и гибкость системы управления версиями вы сможете ощутить после изучения некоторого теоретического материала и применения на практике.

Основные обозначения:

* repository — некоторое хранилище файлов, ссылок на изменения в файлах
* commit — отслеживание изменений, сохраняет разницу в изменениях
* working directory — рабочий каталог на вашем компьютере
* staging area — область подготовленных файлов или рабочая область
* branch — ветка, состоит из набора коммитов, обычно ссылается на последний коммит
* merge — слияние, слияние веток в одну
* pull — втянуть, взять проект с сервера, получить изменения из удаленного репозитория

push — вытолкнуть, отправить изменения на сервер

**Практическая часть**

В ходе выполнения данной практической работы при помощи программного продукта Visual Studio, MS SQL, а также удаленный репозиторий GitHub представленная на рисунках ниже.

Создадим руководство по стилю (рисунок 5)

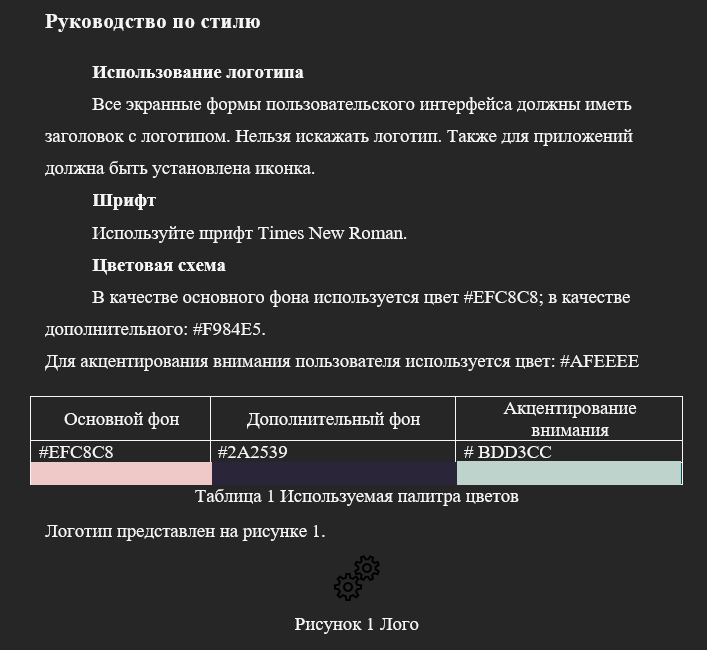


Рисунок 5. Руководство по стилю

Далее зальем все на удаленный репозиторий (рисунок 6)

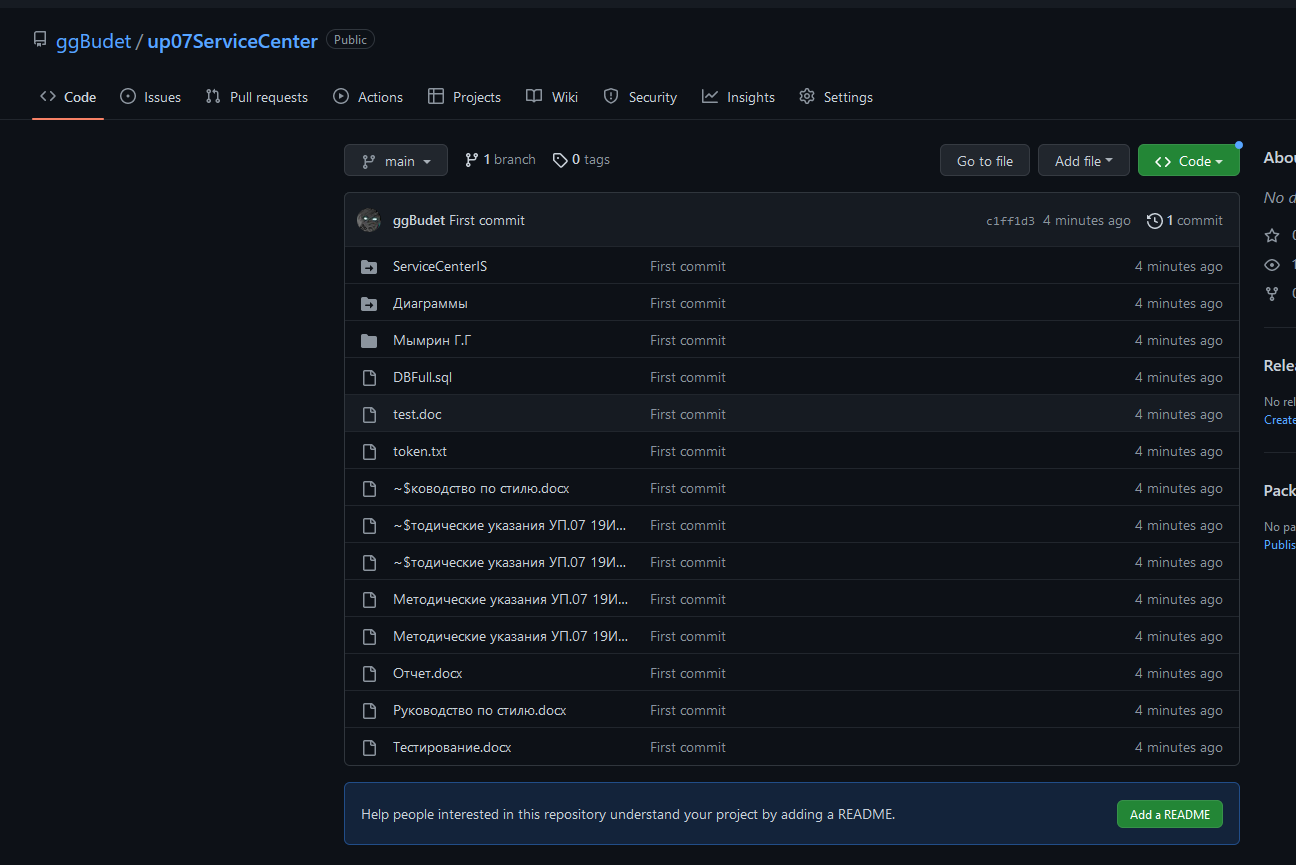


Рисунок 6. Репозиторий

**Контрольные вопросы**

1. В чем заключается экономия времени при использовании системы контроля версий?

Если умрет любая копия репозитория, то это не приведет к потере кодовой базы, поскольку она может быть восстановлена с компьютера любого разработчика. Каждая копия является полным бэкапом данных.

1. В чем преимущества использования системы контроля версий?

Полная история изменений каждого файла за длительный период. Это касается всех изменений, внесенных огромным количеством людей за долгие годы. Изменением считается создание и удаление файлов, а также редактирование их содержимого. Различные инструменты VCS отличаются тем, насколько хорошо они обрабатывают операции переименования и перемещения файлов. В историю также должны входить сведения об авторе, дата и комментарий с описанием цели каждого изменения. Наличие полной истории позволяет возвращаться к предыдущим версиям, чтобы проводить анализ основных причин возникновения ошибок и устранять проблемы в старых версиях программного обеспечения. Если над программным обеспечением ведется активная работа, то «старой версией» можно считать почти весь код этого ПО.

Ветвление и слияние. Эта возможность полезна не только при одновременной работе участников команды: отдельные люди также могут извлечь из нее пользу и работать над несколькими независимыми направлениями. Создание «веток» в инструментах VCS позволяет иметь несколько независимых друг от друга направлений разработки, а также выполнять их слияние, чтобы разработчики могли проверить, что изменения, внесенные в каждую из веток, не конфликтуют друг с другом. Многие команды разработчиков программного обеспечения создают отдельные ветки для каждой функциональной возможности, для каждого релиза либо и для того, и для другого. Наличие множества различных рабочих процессов позволяет командам выбирать подходящий для них способ использования ветвления и слияния в VCS.

Отслеживаемость. Возможность отслеживать каждое изменение, внесенное в программное обеспечение, и связывать его с ПО для управления проектами и отслеживания ошибок, например Jira, а также оставлять к каждому изменению комментарий с описанием цели и назначения изменения может помочь не только при анализе основных причин возникновения ошибок, но и при проведении другого анализа. История с комментариями во время чтения кода помогает понять, что этот код делает и почему действие реализовано именно таким образом. Благодаря этому разработчики могут вносить корректные и совместимые изменения в соответствии с долгосрочным планом разработки системы. Это особенно важно для эффективной работы с унаследованным кодом, поскольку дает разработчикам возможность точнее оценить объем дальнейшей работы.

1. Что такое Git?

Git (произносится «гит») — распределённая система управления версиями.

1. Как начать использовать Git?

* Установи GIT;
* Настрой Git, так Git связывает Ваш аккаунт и Ваши изменения;
* Настрой Git для Windows;
* Создай локальный GIT репозиторий для своего продукта;
* Подключи стандартные каталоги;
* Создай файл в локальном репозитории;
* Свяжи новый файл с удаленным репозиторием;
* Отправь код на удаленный репозиторий.

1. Как начать использовать GitHub?

* Создание аккаунта в Github
* Нажать на кнопку «Start a project».
* Ввести название и описание репозитория.
* Поставить галочку на «Initialize this repository with a README».
* Выбрать нужный тип лицензии и нажать на кнопку «Create project».

1. Основные (наиболее часто используемые) команды Git.

git status - можно посмотреть текущую ветку, список измененных файлов, а также файлы которые нужно добавить под версионный контроль

git add имя\_файла - добавить файл под версионный контроль

git add -A - добавить все файлы

git commit -m "Text message" - зафиксировать изменения

git push - вытолкнуть зафиксированные изменения

git pull - притянуть все изменения из репозитория

1. Какие сервисы существуют для Git?

GitHub; Bitbucket; Directual; GitLab; Beanstalk.

1. Как работать с локальным репозиторием?

Откройте свой Git Bash и перейдите к хранилищу, которое необходимо связать. First\_Project\_Repo

Проверьте, чист ли репозиторий, используя команду git status.

Проверьте, чист ли репозиторий, используя команду git status.

Выполните команду git remote

Поскольку связанного репозитория не существует, из Git не было получено никаких выходных данных.

Теперь с помощью приведенного выше URL-адреса мы свяжем репозиторий. Чтобы связать репозиторий, выполните следующую команду и нажмите клавишу enter:

git remote add origin <https://github.com/harishrajora805/myFirstRepo.git>. git\_remote\_add\_origin

Как только это будет сделано, локальный репозиторий будет связан с репозиторием GitHub. Примечание: пожалуйста, используйте свой собственный URL-адрес репозитория для ссылки.

Как проверить, подключен ли локальный репозиторий к удаленному репозиторию?

Чтобы проверить, связали ли мы наш репозиторий или нет, снова выполните команду git remote: git\_remote\_origin

Как видно, исходный репозиторий доступен. Продолжайте и используйте команду git remote-v для просмотра того же результата вместе с URL-адресом, как показано на рисунке.

1. Как работать с распределенным репозиторием?

В распределенной системе управления версий, кроме версии репозитория на сервере, у каждого разработчика есть свой репозиторий. Это позволяет быстро восстановить актуальное состояние нашего программного обеспечения.

**Вывод:** благодаря данной практической работе был размещён скрипт с базой данных, руководство по стилю, логотип и иконка в удаленный репозиторий проекта.

# **Практическая работа № 1.4**

**Создание словаря стилей. Создание Git репозитория с помощью Visual Studio**

**Цель:** создание словаря стилей. Создание Git репозитория с помощью Visual Studio.

**Задание:**

1. Изучите Пояснения к работе.
2. Используя информацию в пункте «Пояснение к работе», а также руководство по стилю, создайте и настройте проект Visual Studio, реализуйте словарь ресурсов.
3. Создайте репозиторий согласно материалу, изложенному в «Пояснении к работе».

**Теоретический материал**

Чтобы разделить ресурсы между множеством проектов можно создать словарь ресурсов.

**Словарь ресурсов** представляет собой просто XAML-документ, который всего лишь хранит необходимые ресурсы.

Чтобы использовать словарь ресурсов, где-нибудь в приложении его необходимо объединить с коллекцией ресурсов. Это можно делать в каком-то конкретном окне, однако чаще объединение осуществляется на уровне коллекции ресурсов приложения, как показано ниже:

<Application x:Class="WpfApplication1.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

StartupUri="MainWindow.xaml">

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary Source="Dictionary1.xaml"></ResourceDictionary>

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

</Application>

В приведенном коде разметки объект ResourceDictionary создается явно. Коллекция ресурсов всегда представляет собой объект ResourceDictionary, но данный случай является одним из тех, в которых эта деталь должна быть обязательно указана явно, чтобы иметь возможность также устанавливать свойство **ResourceDictionary.MergedDictionaries**. Если не предпринять этот шаг, значением свойства **MergedDictionaries** будет null.

**MergedDictionaries** — это коллекция объектов **ResourceDictionary**, которые будут использоваться для пополнения коллекции ресурсов.

Чтобы добавить собственные ресурсы и включить их в словари ресурсов, необходимо просто разместить их перед или после раздела **MergedProperties**.

Как упоминалось ранее, хранить ресурсы с одинаковыми именами в разных, но перекрывающихся коллекциях ресурсов вполне допустимо, однако объединять словари ресурсов, которые содержат ресурсы с одинаковыми именами — нет. При обнаружении дубликата во время компиляции приложения генерируется исключение **XamlParseException**.

Одна из причин применения словарей ресурсов — определение одной или нескольких многократно используемых «обложек» приложения, которые можно применять к элементам управления. Еще одна причина связана с необходимостью сохранения содержимого, которое должно быть локализовано (такого как строки сообщений об ошибках).

**Стили** позволяют определить набор некоторых свойств и их значений, которые потом могут применяться к элементам в xaml. Стили хранятся в ресурсах и отделяют значения свойств элементов от пользовательского интерфейса. Также стили могут задавать некоторые аспекты поведения элементов с помощью триггеров. Аналогом стилей могут служить каскадные таблицы стилей (CSS), которые применяются в коде html на веб-страницах.

Стиль создается как ресурс с помощью объекта Style, который представляет класс **System.Windows.Style**. И как любой другой ресурс, он обязательно должен иметь ключ. С помощью коллекции **Setters** определяется группа свойств, входящих в стиль. В нее входят объекты **Setter**, которые имеют следующие свойства:

* **Property**: указывает на свойство, к которому будет применяться данный сеттер. Имеет следующий синтаксис: Property="Свойство\_элемента".
* **Value:** устанавливает значение.

**Относительный путь к файлу от документа** — это путь к файлу относительно текущего документа. Такой адрес зависит от расположения файла, в котором он записан.

**Практическая часть**

Создадим словарь ресурсов и добавим туда стили (рисунок 7)

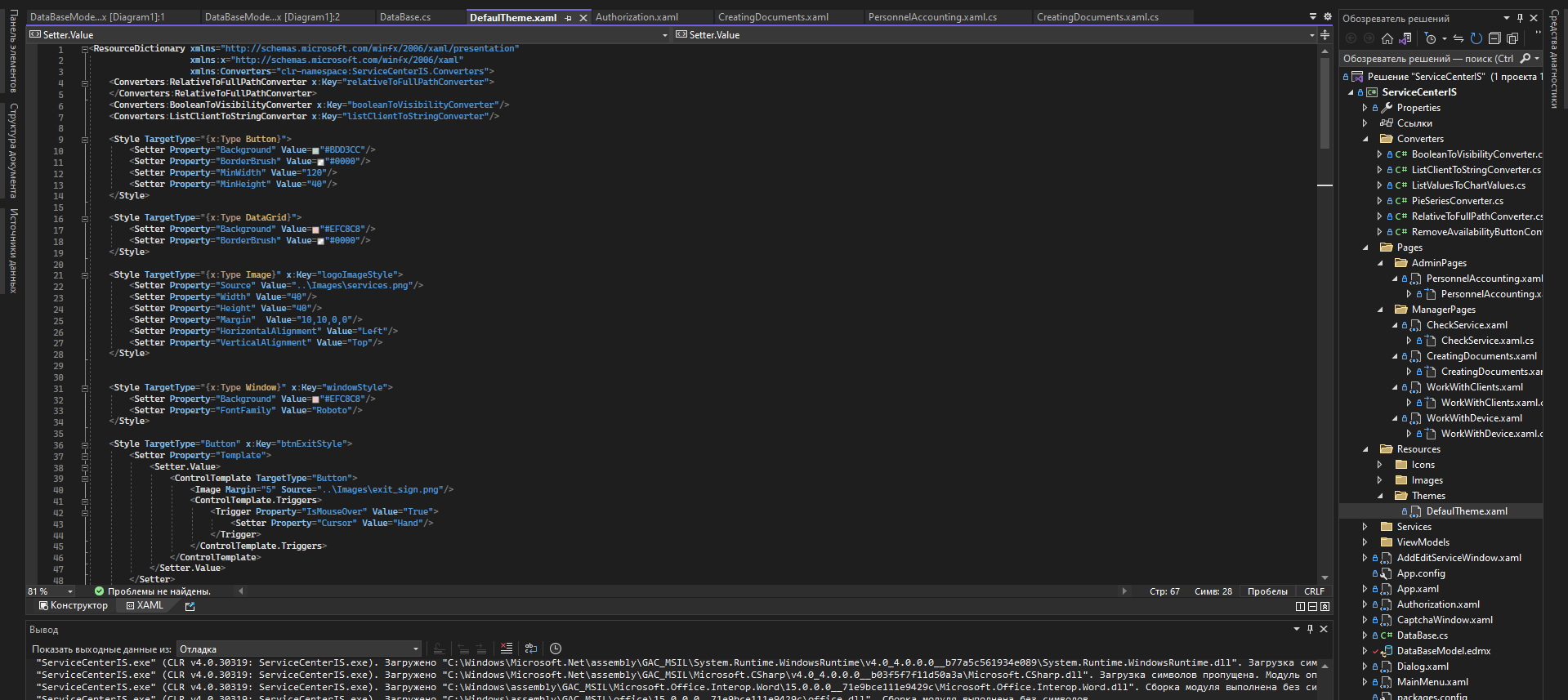


Рисунок 7. Словарь стилей

Подключим словарь ресурсов (рисунок 8)

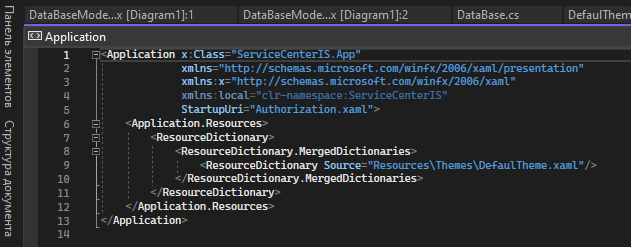


Рисунок 8. Подключение словаря стилей

**Вывод:** в ходе проделанной работы был реализован словарь ресурсов и подключен в приложении.

# **Практическая работа № 1.5**

**Реализация авторизации пользователя.**

**Цель** реализовать авторизацию в системе.

**Задание:**

1. Изучите Пояснения к работе.
2. Реализуйте форму авторизации. Данные для авторизации берутся из базы данных
3. Оформите отчет о проделанной работе (согласно плану отчета).

**Теоретический материал**

*Авторизация* — процесс предоставления пользователю или группе пользователей определенных разрешений, прав доступа и привилегий в компьютерной системе

**Практическая часть**

Создадим интерфейс авторизации (рисунок 9)

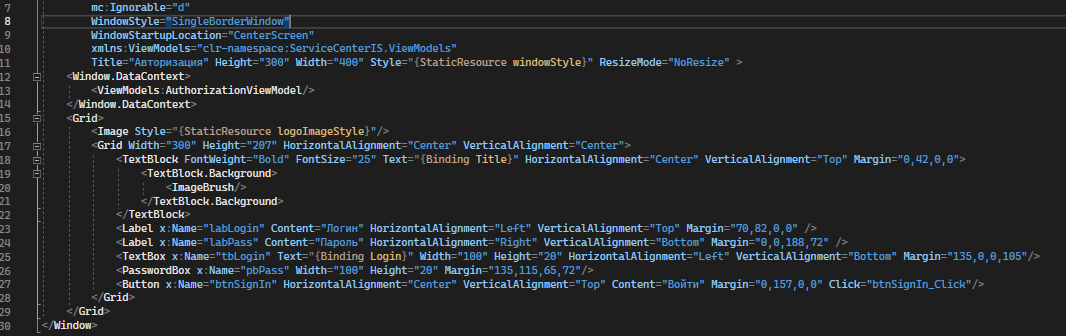


Рисунок 9. Xaml код

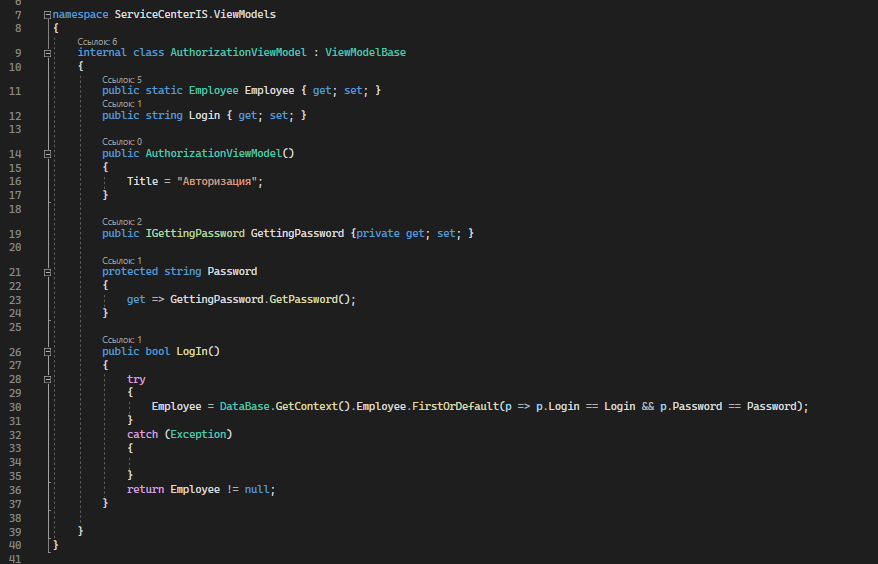


Рисунок 10. Код модели

Результат окна авторизации (рисунок 11)

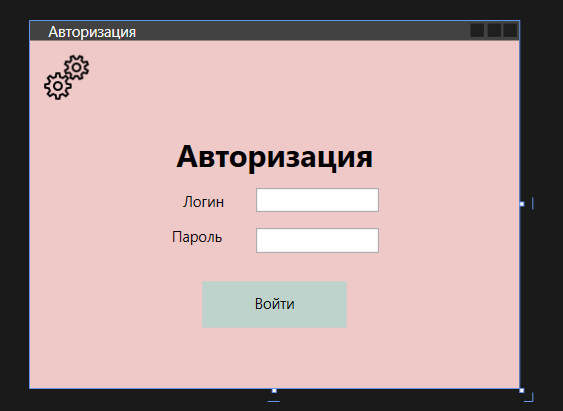


Рисунок 11. Авторизация

Далее реализуем капчу (рисунки ниже)



Рисунок 12. Xaml капчи

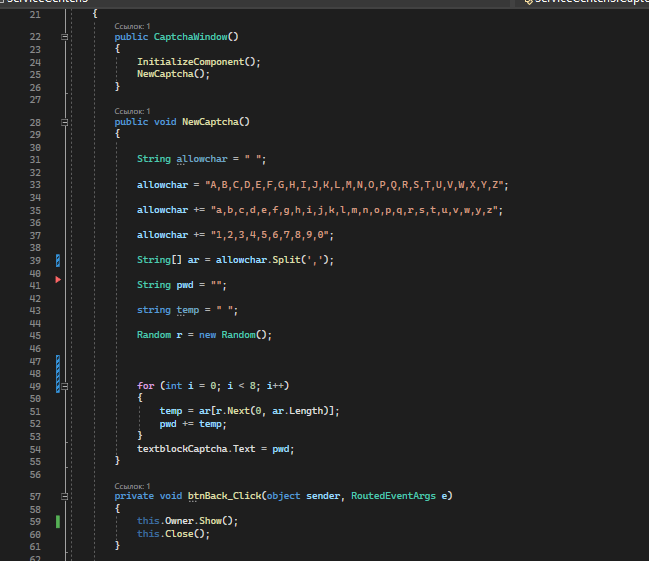


Рисунок 13. Cs код капчи

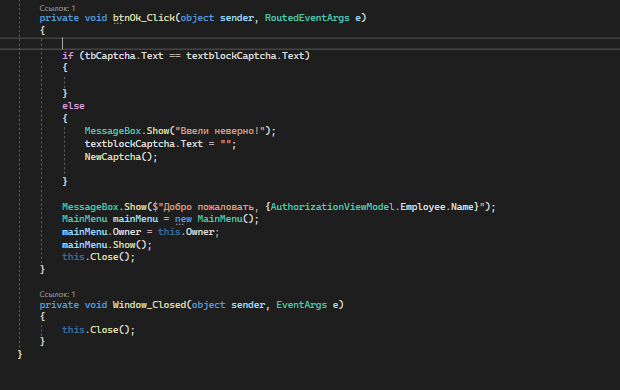


Рисунок 14. Cs код капчи

**Контрольные вопросы**

1. Что такое «авторизация», для чего она нужна?

Авторизация — процесс предоставления пользователю или группе пользователей определенных разрешений, прав доступа и привилегий в компьютерной системе. Авторизация нужна для обеспечения безопасности при совершении действий, для разграничения прав пользователей, для защиты от злоумышленников.

1. Как реализовать авторизацию в wpf-приложении.

Для того чтобы разработать систему «Авторизации и регистрации» в приложении WPF, для начала нужно, создать страницы, на которых будет происходить регистрация и авторизация. Создаётся начальное окно, через которое происходит общение между всеми страницами.

На данном окне нет ничего, кроме компонента «Frame», который был добавлен из «Панели элементов» и имеет следующие свойства: имя компонента – frame, ширина и высота компонента растянута на всю ширину и высоту окна. «Frame» позволяет открывать страницы, в области, которую он занимает. Теперь перейдём к коду данного окна.

Здесь используются две сторонние функции, разработка которых указана в пунктах «Подключение к базе данных» и «Открытие страниц».

Страница Авторизации имеет:

* два компонента «Label» (текст). Данные поля нужны для отображения надписей на рабочей форме, в данном случае «Введите логин» и «Введите пароль»;
* компонент «TextBox» (поле ввода текста) с наименованием «textBox\_login». Позволяет вносить текст в специальное поле ввода, необходим для указания логина пользователя;
* компонент «PasswordBox» (поле ввода текста скрывающие введённый текст) с наименованием «password». Данное поле, позволяет вносить текст в специальное поле для ввода, скрывая введённый текст специальным символом;
* два компонента «Button» (кнопки) с прикреплёнными функциями. Компонент типа «Button» предназначены для создания кнопок на форме, которые в свою очередь способны обрабатывать большое количество событий.

**Вывод:** в ходе проделанной работы была реализована авторизация с последующей проверкой в виде капчи.

# **Практическая работа № 2**

**Разработка тестового набора и тестового сценария для программного обеспечения.**

**Цель:** реализовать тестовый набор для авторизации пользователя в системе. Разработать тестовый сценарий и провести тестирование.

**Задание:**

1. Изучите Пояснения к работе.
2. Реализуйте тестовый набор для авторизации в системе.
3. Разработайте тестовый сценарий.
4. Проведите тестирование.
5. Создайте подключение к базе данных и реализуйте отображение данных на форме.
6. Оформите отчет о проделанной работе.

**Теоретический материал**

**Правила написания тест-кейсов**

1. Заголовок:
   * должен быть чётким, кратким, понятным и однозначно характеризующим суть тест-кейса;
   * не может содержать выполняемые шаги и ожидаемый результат.
2. Предусловие:
   * может содержать полную информацию о состоянии системы или объекта, необходимом для начала выполнения шагов тест-кейса;
   * может содержать ссылки на информационные источники, которые необходимо изучить перед прохождением тест-кейса (инструкции, описание систем…);
   * не может содержать ссылки на тестируемый ресурс, если у информационной системы более одной среды (прод, тест, препрод…), данная информация должна быть вынесена в инструкцию, и ссылка приложена в предусловии;
   * не может содержать данные для авторизации, данная информация должна быть вынесена в инструкцию, и ссылка приложена в предусловии;
   * не может содержать выполняемые шаги и ожидаемый результат, если нам нужно, чтобы до выполнения шагов проверки у нас была открыта главная страница, то мы в предусловии указываем «открыта главная страница сайта»;
   * не может содержать ожидаемый результат.
3. Шаги проверки:
   * должны быть чёткими, понятными и последовательными;
   * следует избегать излишней детализации шагов.

Правильно: «ввести в поле число 12».  
Неправильно: «нажать на клавиатуре на цифру ‘1’, следующим шагом нажать на клавиатуре на цифру ‘2’»;

* + должны использоваться безличные глаголы.  
    Правильно: нажать, ввести, перейти.  
    Неправильно: нажмите, введите, идите;
  + не должно быть комментариев и пояснений, если есть необходимость привести мини-инструкцию, то оформляем инструкции в базе-знаний и в предусловии ссылаемся на неё;
  + не должно быть жёстко прописанных статических данных (логины, пароли, имена файлов) и примеров, для исключения эффекта пестицида.

1. Ожидаемый результат:
   * должен быть у каждого шага проверки;
   * должно быть кратко и понятно описано состояние системы или объекта, наступающее после выполнения соответствующего шага;
   * не должно быть избыточного описания.
2. Общие требования к тест-кейсам:
   * язык описания тест-кейсов должен быть понятен широкому кругу пользователей, а не узкой группе лиц;
   * тест-кейс должен быть максимально независим от других тест-кейсов и не ссылаться на другие тест-кейсы (лучшая практика, когда зависимостей нет вообще);
   * тест-кейсы группируются в функциональные блоки по их назначению;
   * в тест-кейсах, проверяющих работу функционала скриншотов быть не должно, иначе вы будете посвящать сотни часов на изменение всех скриншотов в тысячах тест-кейсах при изменении интерфейса тестируемой программы. Скриншоты могут быть добавлены только в тест-кейсы, проверяющие отображение страниц и форм.

На самом деле правила простые, однако их не так-то просто соблюдать. Если же придерживаться данных правил, то тест-кейсы будут легко поддерживаемыми, легко читаемыми, не будут вызывать отторжения и могут быть использованы всеми участниками команды в процессе разработки программного обеспечения.

**Практическая часть**

В ходе данной практической работы на основе шаблонов, были разработаны собственные тест-кейсы, представленные в таблицах ниже.

|  |  |
| --- | --- |
| **Поле** | **Описание** |
| **Название проекта** | ServiceCenterIS |
| **Рабочая версия** | 1.0 |
| **Имя тестирующего** | Мымрин Г.Г. |
| **Дата(ы) теста** | 22.11.2022 |
| **Тестовый пример #** | TC\_UI\_1 |
| **Приоритет тестирования**  *(Низкий/Средний/Высокий)* | Средний |
| **Заголовок/название теста** | Подтвердите страницу авторизации с действительным именем пользователя и паролем. |
| **Краткое изложение теста** | Успешная авторизация пользователя с действительным именем пользователя и паролем. |
| **Этапы теста** | 1. Нажать Tab. 2. Ввести в поле Логин. 3. Нажать Tab. 4. Ввести в поле Пароль. 5. Нажать кнопку Вход. 6. Проверить окно на успешный вход. |
| **Тестовые данные** | Логин – «Ivan» Пароль – «qwe123» |
| **Ожидаемый результат** | После нажатия кнопки Вход, появляется окно с капчей |
| **Фактический результат** | После нажатия кнопки Вход, появляется окно с капчей |
| **Предварительное условие** | Узнать свой логин и пароль. |
| **Постусловие** | Состояние системы в норме. |
| **Статус** *(Зачет/Незачет)* | Зачет |
| **Примечания/комментарии** | Нет. |

**Вывод:** в ходе проделанной работы был разработан тест-кейс.

**Контрольные вопросы**

1. Что такое тест-кейс?

Тест-кейс — это четкое описание входных данных, условий выполнения, процедуры тестирования и ожидаемых результатов.

1. Перечислите общие требования к тест-кейсам.

* ID — уникальное сочетание букв и цифр.
* Заголовок — основная идея тест-кейса, краткое описание его сути. Например, заголовок тест-кейса для ручного тестирования страницы входа может выглядеть следующим образом: «Проверить вход пользователя с корректными данными».
* Предусловия — список действий, которые необходимо выполнить перед выполнением тест-кейса. При необходимости здесь могут указываться учетные данные.
* Шаги — описание действий, необходимых для проверки.
* Постусловия — список действий, возвращающих систему в исходное состояние (указывается при необходимости).
* Ожидаемый результат — то, что мы ожидаем получить после успешного выполнения тест-кейса.
* Фактический результат — то, что мы получаем после выполнения тест-кейса (указывается при необходимости).
* Статус — Success (успех), Failed (провал), Blocked (блокировка) (указывается при необходимости).

# **Практическая работа № 3**

**Реализация главного окна навигации с использованием страниц.**

**Цель:** создать главное окно с возможностью навигации по страницам.

**Задание:**

1. Изучите Пояснения к работе.
2. Используя информацию в пункте «Пояснение к работе», выполните реализацию окна навигации
3. Оформите отчет

**Теоретический материал**

Чтобы создать страничное приложение в WPF, нужно использовать класс System.Windows.Controls.Page.

Модель для создания страниц в WPF во многом похожа на модель для создания окон. Хотя создавать объекты страниц можно и с помощью одного лишь кода, обычно для каждой страницы создается файл XAML и файл отделенного кода. При компиляции этого приложения компилятор создает производный класс страницы, который объединяет написанный разработчиком код с генерируемыми автоматически связующими элементами (такими как поля, которые ссылаются на каждый именованный элемент на странице).

Хотя страницы и являются самым высокоуровневым компонентом пользовательского интерфейса при проектировании приложения, во время его выполнения контейнером наивысшего уровня они уже не будут. Вместо этого они обслуживаются в другом контейнере. Именно в этом и состоит секрет гибкости, обеспечиваемой WPF в случае страничных приложений, ведь в качестве такого контейнера WPF позволяет использовать любой из нескольких следующих объектов:

* объект NavigationWindow, который представляет собой немного видоизмененную версию класса Window;
* объект Frame, находящийся внутри другого окна;
* объект Frame, находящийся внутри другой страницы;
* объект Frame, обслуживаемый непосредственно в Internet Explorer или Firefox.

**Класс Page**

Класс Page допускает наличие только единственного вложенного элемента. Однако класс Page не является элементом управления содержимым: он на самом деле унаследован непосредственно от класса FrameworkElement. Вдобавок класс Page является более простым и отлаженным, чем класс Window. Он имеет небольшой набор дополнительных свойств, которые позволяют настраивать его внешний вид, взаимодействовать с контейнером только определенным, ограниченным образом и применять навигацию. Все эти свойства перечислены ниже:

Background - принимает кисть, которая позволяет устанавливать заливку для фона

Content - принимает один элемент, который отображается на странице. Обычно в роли такого элемента выступает контейнер компоновки, такой как Grid или StackPanel

Foreground, FontFamily и FontSize - определяют внешний вид по умолчанию для текста внутри страницы. Значения этих свойств наследуются элементами внутри страницы. Например, если устанавливается заливка переднего плана и размер шрифта, по умолчанию содержимое внутри страницы получает эти же настройки

WindowWidth, WindowHeight и WindowTitle - определяют внешний вид окна, в которое упаковывается страница. Эти свойства позволяют управлять хостом путем установки его ширины, высоты и заголовка. Однако они действуют только в том случае, если страница обслуживается в окне (а не во фрейме)

NavigationService - возвращает ссылку на объект NavigationService, которую можно использовать для отправки пользователя на другую страницу программным путем

KeepAlive - определяет, должен ли объект страницы оставаться действующим после перехода пользователя на другую страницу.

ShowsNavigationUI - определяет, должен ли хост для данной страницы отображать навигационные элементы управления (кнопки "назад" и "вперед"). По умолчанию имеет значение true.

Title - устанавливает имя, которое должно применяться для страницы в хронологии навигации. Хост не использует свойство Title для установки заголовка в строке заголовка: для этой цели у него есть свойство WindowTitle.

Также важно обратить внимание на отсутствующие компоненты — в классе Page нет эквивалентов для методов Hide() и Show(), доступных в классе Window. Если потребуется показать другую страницу, придется воспользоваться навигацией.

**MVVM (Model-View-ViewModel)** — способ организации кода. Он помогает отделить пользовательский интерфейс от логики.

**Паттерн MVVM**

**View** содержит только код пользовательского интерфейса. Он отображает экран (текстовые поля, ярлыки, кнопки) и пользовательский ввод. Например, в [Windows Forms](https://ru.wikipedia.org/wiki/Windows_Forms) это обычно Form или Control. В [Windows Presentation Foundation](https://ru.wikipedia.org/wiki/Windows_Presentation_Foundation) (WPF) — это класс Window: файлы XAML и xaml.cs.

**Model** — это классы. Это «вещи» в вашей программе. Например, в программе ввода заказов вашими классами моделей могут быть: Customer, Order, InventoryItem. Эти классы будут содержать логику, необходимую для выполнения функций.

**ViewModel** используются для связи между View и Model. Они обычно не содержат много логики. Они содержат Model, который должен быть использован во View. Когда человек взаимодействует с пользовательским интерфейсом, ViewModel узнает, какое действие необходимо выполнить, и просит Model сделать это.

**Как работает паттерн MVVM?**

Как работает паттерн MVVM можно разобрать на примере из реального мира. Представьте, что у нас есть владелец интернет-магазина, веб-мастер и контент-менеджер:

* владелец интернет-магазина (Model) — занимается своей работой, ни на что не отвлекаясь. Если начинается важная акция, он передает веб-мастеру, что нужно добавить новость о ней в блог;
* веб-мастер (ViewModel) – получает информацию от владельца интернет-магазина и передает ее контент-менеджеру;
* контент-менеджер (View) — пишет новость в блог, основываясь на данных, которые предоставил веб-мастер, и публикует ее.

**Практическая часть**

Данная практическая работа была выполнена с помощью средств Visual Studio. Главное окно, с возможностью навигации представлено на рисунках ниже.

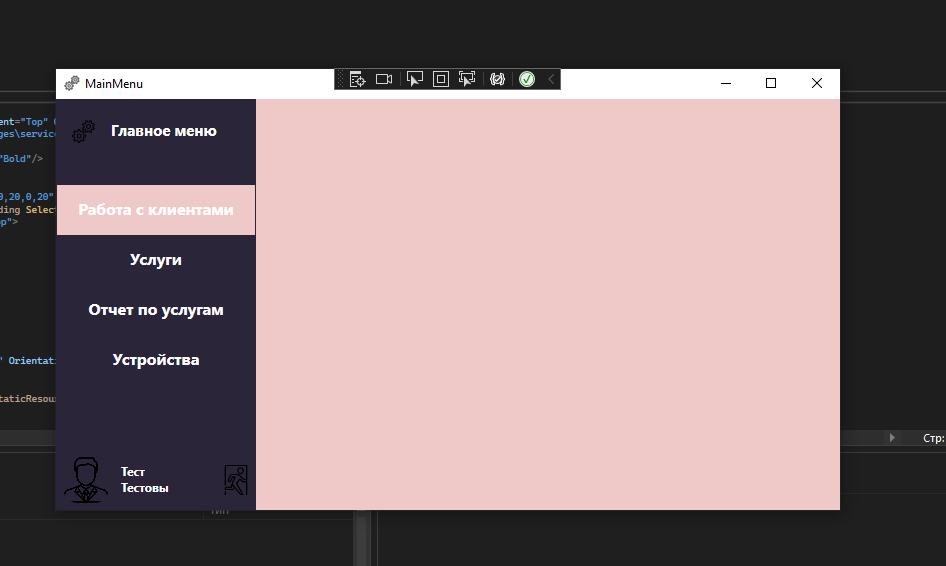


Рисунок 15. Главное меню

**Вывод:** в ходе проделанной работы было разработано главное окно программы.

# **Практическая работа № 4**

**Отображение данных на форме.**

**Цель:** реализовать отображение данных базы данных на форме.

**Задание:**

1. Изучите Пояснения к работе.
2. Создайте элемент TabControl и измените его стиль.
3. Реализуйте отображение данных с помощью DataGrid на второй вкладке TabControl.
4. Разработайте шаблон отображения данных на форме.
5. Реализуйте отображение данных по шаблону на первой вкладе TabControl.
6. Оформите отчет по практической работе.

**Теоретический материал**

ListBox представляет собой обычный список. Содержит коллекцию элементов ListBoxItem, которые являются типичными элементами управления содержимым. Также ListBox может содержать любые другие элементы.

Мы можем использовать элементы как внутри элемента ListBoxItem, так и непосредственно вставляя их в список.

**Практическая часть**

Данная практическая работа была выполнена с помощью средств Visual Studio. Как было реализовано отображение данных с базы данных, представлено на рисунках ниже.

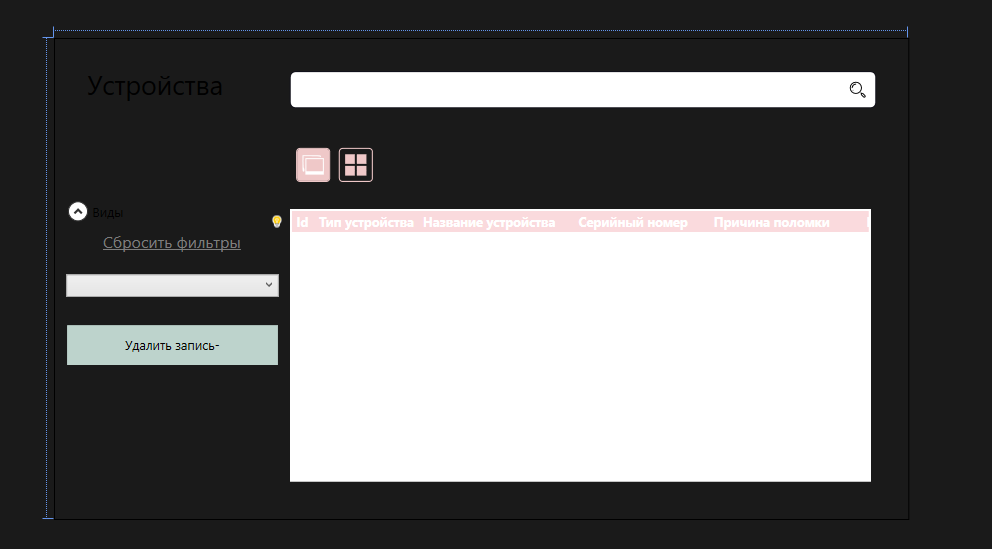


Рисунок 16. Окно вывода данных



Рисунок 17. Cs код капчи

**Вывод:** в ходе проделанной работы было реализовано отображение данных базы данных на форме.

**Контрольные вопросы:**

1. Что из себя представляет ListBox?

ListBox представляет собой обычный список.

1. Что такое TabControl и какими свойствами он обладает?

TabControl позволяет разбить ваш интерфейс на несколько разных зон, которые можно выбрать, нажав на заголовок, который обычно находится сверху элемента управления.

Чтобы применить одни и те же параметры свойства к нескольким TabControl элементам управления, используйте Style свойство. Можно изменять свойства ControlTemplate, заданные по умолчанию, чтобы придать элементу управления уникальный внешний вид.

Свойства зависимостей для этого элемента управления могут быть заданы стилем элемента управления по умолчанию. Если свойство задано стилем по умолчанию, при отображении элемента управления в приложении свойство может измениться по умолчанию. Стиль по умолчанию определяется темой рабочего стола, используемой при запуске приложения.

1. Какими свойствами обладает элемент GeometryDrawing?

Элемент GeometryDrawing обладает свойствами: Geometry, Brush, Pen.

# **Практическая работа № 5.1**

**Реализация поиска и фильтрации.**

**Цель:** реализовать функцию поиска и фильтрации данных из базы данных с помощью программного приложения.

**Задание:**

1. Изучите Пояснения к работе.
2. С помощью средств Visual Studio реализуйте окно редактирования данных в базе данных.
3. Оформите отчет о проделанной работе.

**Практическая часть**

Данная практическая работа была выполнена с помощью средств Visual Studio. Как было реализовано отображение данных с базы данных, представлено на рисунках ниже.

Поиск по названию устройства (рисунок 18)

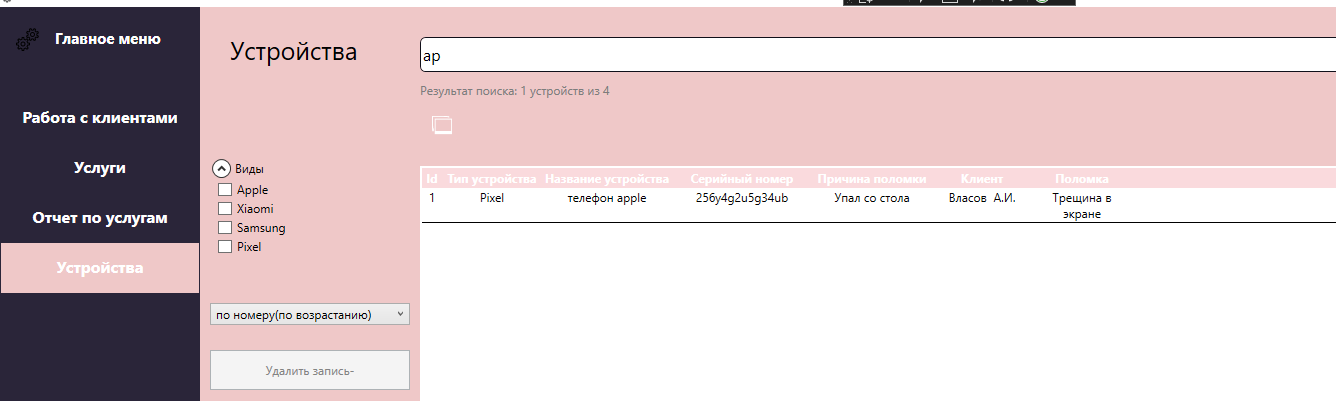


Рисунок 18. Поиск

Фильтрация (рисунок 19)

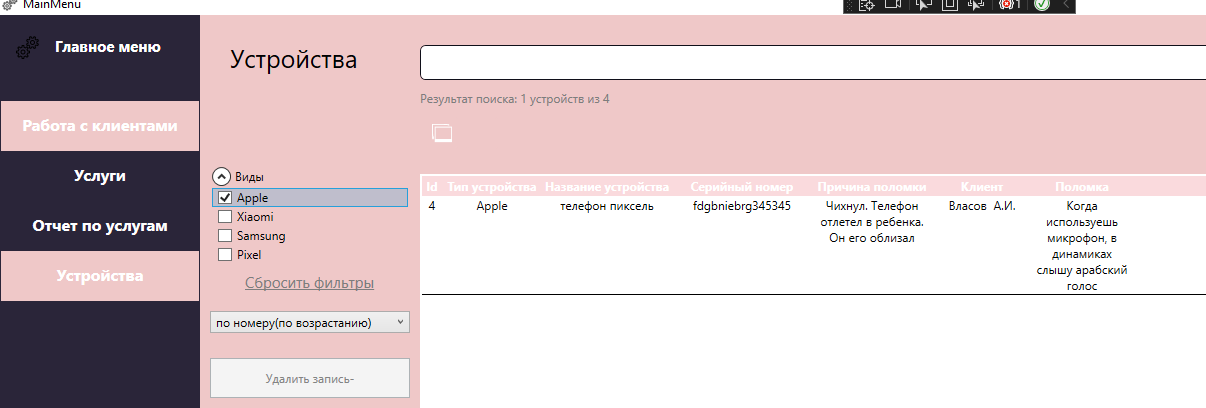


Рисунок 19. Фильтрация

**Вывод:** в ходе проделанной работы была реализована функция поиска и фильтрации данных из базы данных.

# **Практическая работа № 5.2**

**Реализация сортировки**

**Цель:** реализовать сортировку данных из базы данных с помощью программного приложения.

**Задание:**

1. Изучите Пояснения к работе.
2. С помощью средств Visual Studio реализуйте окно редактирования данных в базе данных.
3. Оформите отчет о проделанной работе.

**Практическая часть**

Реализуем сортировку, на рисунке ниже показаны параметры сортировки.

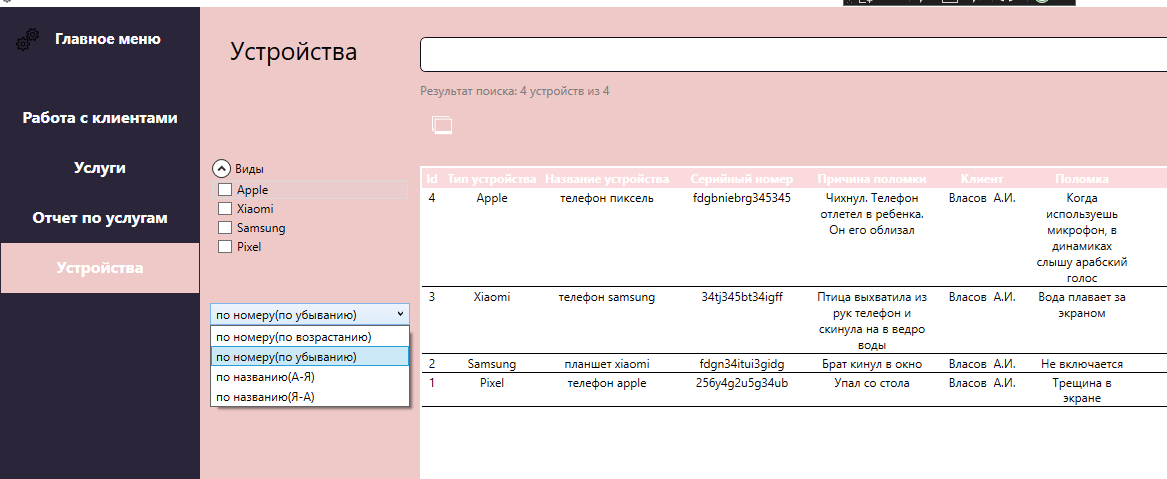


Рисунок 20. Параметры сортировки

**Вывод:** в ходе проделанной работы была реализована функция сортировки данных из базы данных.

# **Практическая работа № 6**

**Реализация удаления записей**

**Цель:** реализовать удаление данных из базы данных с помощью программного приложения.

**Задание:**

1. Изучите Пояснения к работе.
2. С помощью средств Visual Studio реализуйте окно редактирования данных в базе данных.
3. Оформите отчет о проделанной работе.

**Теоретический материал**

В WPF кроме обработки событий приложение может взаимодействовать с пользователем с помощью команд. Команды представляют механизм выполнения какой-нибудь задачи, например, копирования текста - когда мы нажимаем Ctrl+C, то мы копируем текст в буфер. В процессе копирования выполняется ряд действий, и все вместе эти действия объединяются в одну команду. Использование команд помогает нам сократить объем кода и использовать одну и ту же команду для нескольких элементов управления в различных местах программы. Таким образом, команды позволяют абстрагировать набор действий от конкретных событий конкретных элементов.

В некотором роде команды в WPF являются реализацией общераспространенного паттерна [Команда](http://metanit.com/sharp/patterns/3.3.php).

Модель команд в WPF состоит из четырех аспектов:

* Сама команда, которая представляем выполняемую задачу
* Привязка команд, которая связывает команду с определенной логикой приложения
* Источник команды - элемент пользовательского интерфейса, который запускает команду (например, кнопка, по нажатию который выполняется команда)
* Цель команды - элемент интерфейса, на котором выполняется команда

**Практическая часть**

Реализовано удаление по нажатие на кнопку, показано на рисунках ниже.

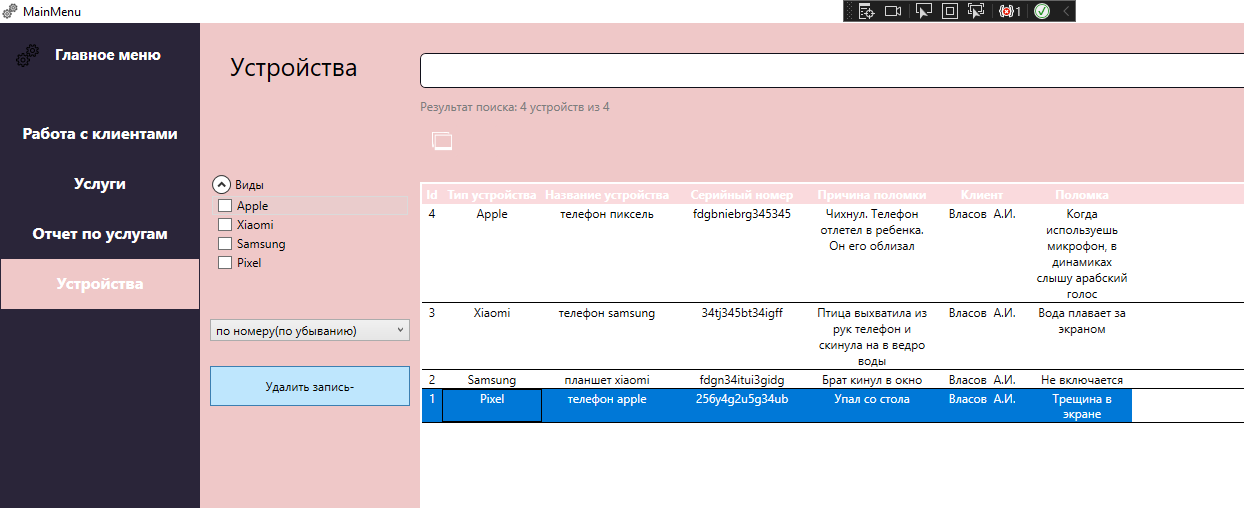


Рисунок 21. Удаление последней записи

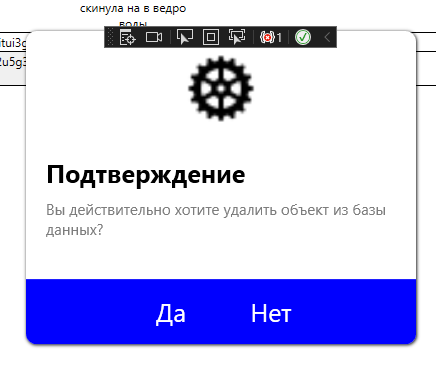


Рисунок 22. Подтверждение удаления

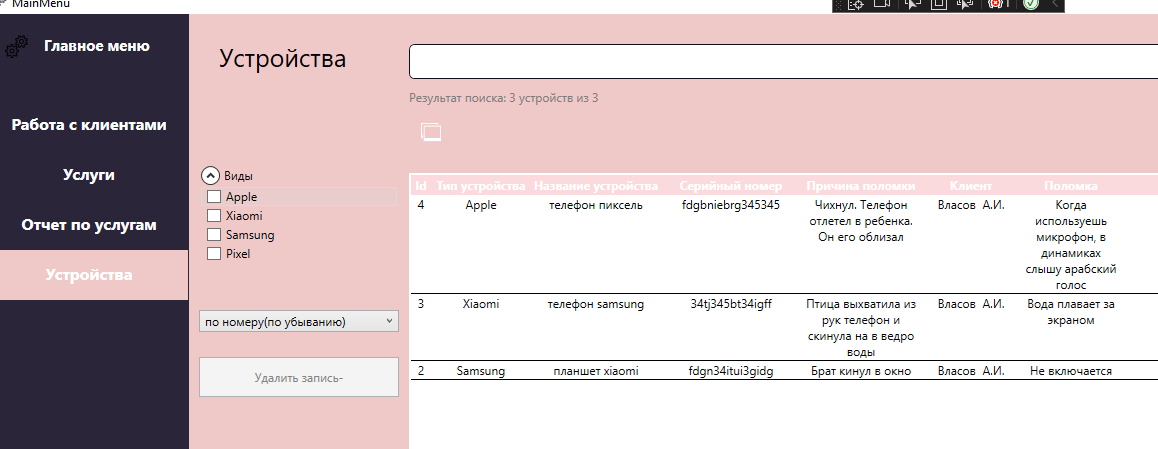


Рисунок 23. Результат удаления

**Вывод:** в ходе проделанной работы была реализована функция удаление данных из базы данных.

**Контрольные вопросы**

1. Что такое «команда» в WPF?

В некотором роде команды в WPF являются реализацией общераспространенного паттерна [Команда](http://metanit.com/sharp/patterns/3.3.php).

1. Из каких компонентов состоит команда в WPF?

Модель команд в WPF состоит из четырех аспектов:

* Сама команда, которая представляем выполняемую задачу
* Привязка команд, которая связывает команду с определенной логикой приложения
* Источник команды - элемент пользовательского интерфейса, который запускает команду (например, кнопка, по нажатию который выполняется команда)
* Цель команды - элемент интерфейса, на котором выполняется команда.

# **Практическая работа № 7**

**Добавление и редактирование записей**

**Цель:** реализовать добавление и редактирование данных из базы данных с помощью программного приложения.

**Задание:**

1. Изучите Пояснения к работе.
2. С помощью средств Visual Studio реализуйте окно редактирования данных в базе данных.
3. Оформите отчет о проделанной работе.

**Теоретический материал**

User Control предоставляет простой способ создания собственных элементов управления и является объединением нескольких существующих элементов управления в один элемент, допускающий простое повторное использование.

**Практическая часть**

Данная практическая работа была выполнена с помощью средств Visual Studio. Было реализовано добавление и редактирование данных из базы данных.

Добавление записи показано на рисунке ниже.

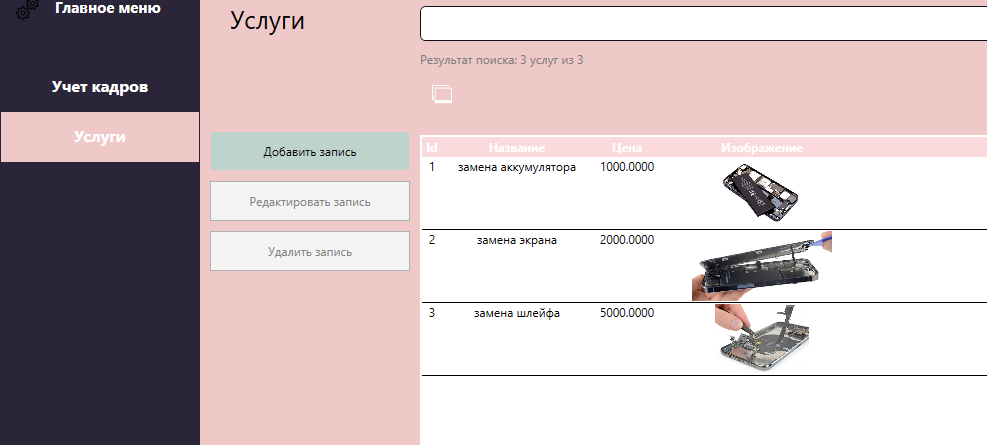


Рисунок 24. Добавление записи

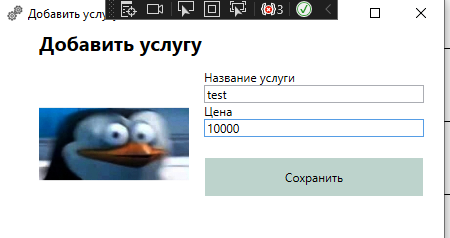


Рисунок 25. Добавление услуги

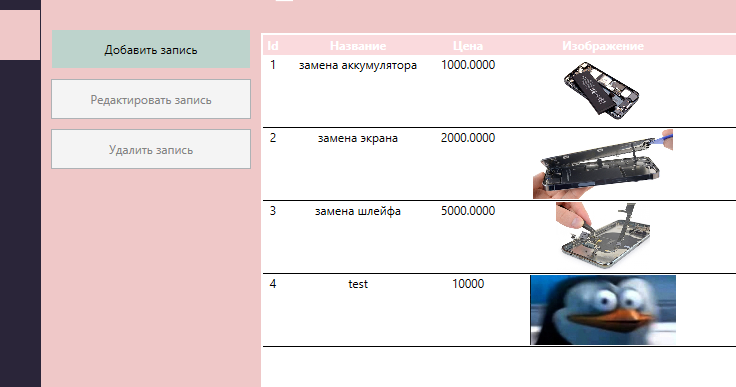


Рисунок 26. Результат добавления услуги

**Вывод:** в ходе проделанной работы была реализована функция добавления и редактирования данных из базы данных.

**Контрольные вопросы**

1. Что такое обобщение в C#?

Обобщения — конструкции языка программирования C#, позволяющие писать код, который будет одинаково работать с различными типами данных с сохранением строгой типизации языка.

1. Какие ограничения обобщений вы узнали при работе над данной практической?

Обобщение, которое имеет ограничение в виде семейства классов ViewModelBase. Второе обобщение – это классы семейства Window, которые при этом имеют конструктор без параметров (ограничение по конструктору устанавливается через new()).

1. Для чего используется элемент UserControl?

Элемент UserControl используется для выбор изображений в проекте.

1. Каким образом заводится Dependency Property?

Модификаторы данного объекта – поле должно быть публично, оно статическое и доступно только для чтения (модификатор readonly). Создание экземпляра DependencyProperty происходит через статический метод данного класса Register.

# **Практическая работа № 8**

**Формирование отчета. Графики. Программное формирование документа Word.**

**Цель:** реализовать построение графиков с помощью библиотеки LiveCharts. Реализовать программное формирование отчета в виде Word документа.

**Задание:**

1. Изучите Пояснения к работе.
2. С помощью средств Visual Studio реализуйте построение графиков с помощью библиотеки LiveCharts. Реализовать программное формирование отчета в виде Word документа.
3. Оформите отчет о проделанной работе.

**Теоретический материал**

Ключевой инструмент для любой современной платформы разработки — это механизм, с помощью которого разработчики могут создавать, передавать друг другу и использовать полезный код. Часто такой код распределен по "пакетам", включающим скомпилированный код (в виде библиотек DLL) и другое содержимое, необходимое использующим эти пакеты проектам.

Для .NET (в том числе .NET Core) механизмом совместного использования кода, поддерживаемым Майкрософт, является NuGet, который определяет, как создаются, размещаются и используются пакеты для .NET, а также [предоставляет средства](https://learn.microsoft.com/ru-ru/nuget/install-nuget-client-tools) для каждой из этих ролей.

Проще говоря, пакет NuGet представляет собой отдельный ZIP-файл с расширением .nupkg, который содержит скомпилированный код (DLL), другие файлы, связанные с этим кодом, и описательный манифест, включающий такие сведения, как номер версии пакета. Разработчики, у которых есть код, к которому нужно предоставить общий доступ, создают пакеты и публикуют их на закрытых или открытых узлах. Потребители получают эти пакеты из соответствующих узлов, добавляют их в свои проекты, а затем вызывают функции пакета в коде своего проекта. При этом NuGet сам обрабатывает все промежуточные данные.

Так как NuGet поддерживает закрытые узлы наряду с открытым узлом nuget.org, с помощью пакетов NuGet вы можете делиться кодом, используемым в рамках организации или рабочей группы. Пакеты NuGet также являются удобным способом факторизовать свой код для использования только в собственных проектах. Иными словами, пакет NuGet является совместно используемой единицей кода, однако не требует и не подразумевает какого-либо определенного способа предоставления общего доступа.

Нередко программа выполняет такие операции, которые могут занять продолжительное время, например, обращение к сетевым ресурсам, чтение-запись файлов, обращение к базе данных и т.д. Такие операции могут серьезно нагрузить приложение. Особенно это актуально в графических (десктопных или мобильных) приложениях, где продолжительные операции могут блокировать интерфейс пользователя и негативно повлиять на желание пользователя работать с программой, или в веб-приложениях, которые должны быть готовы обслуживать тысячи запросов в секунду. В синхронном приложении при выполнении продолжительных операций в основном потоке этот поток просто бы блокировался на время выполнения операции. И чтобы продолжительные операции не блокировали общую работу приложения, в C# можно задействовать асинхронность.

Асинхронность позволяет вынести отдельные задачи из основного потока в специальные асинхронные методы и при этом более экономно использовать потоки. Асинхронные методы выполняются в отдельных потоках. Однако при выполнении продолжительной операции поток асинхронного метода возвратится в пул потоков и будет использоваться для других задач. А когда продолжительная операция завершит свое выполнение, для асинхронного метода опять выделяется поток из пула потоков, и асинхронный метод продолжает свою работу.

Ключевыми для работы с асинхронными вызовами в C# являются два оператора: async и await, цель которых - упростить написание асинхронного кода. Они используются вместе для создания асинхронного метода.

Асинхронный метод обладает следующими признаками:

В заголовке метода используется модификатор async

Метод содержит одно или несколько выражений await

В качестве возвращаемого типа используется один из следующих:

* void
* Task
* Task<T>
* ValueTask<T>

Асинхронный метод, как и обычный, может использовать любое количество параметров или не использовать их вообще. Однако асинхронный метод не может определять параметры с модификаторами out, ref и in.

Также стоит отметить, что слово async, которое указывается в определении метода, НЕ делает автоматически метод асинхронным. Оно лишь указывает, что данный метод может содержать одно или несколько выражений await.

**Практическая часть**

Данная практическая работа была выполнена с помощью средств Visual Studio и библиотеки LiveCharts. Было реализовано построение графиков, а также программное формирование отчета в виде Word документа.

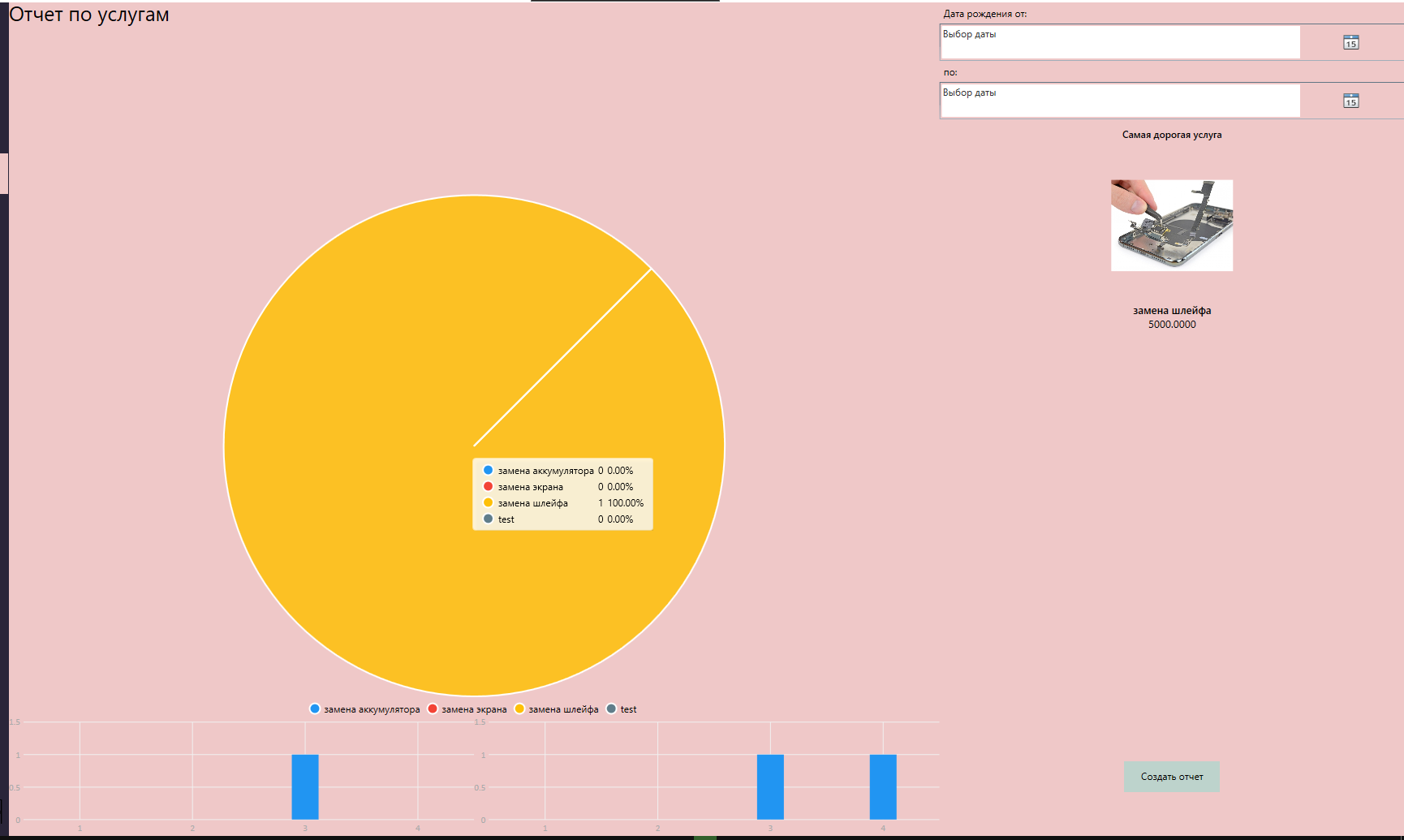


Рисунок 27. Окно диаграмм и формирования отчета

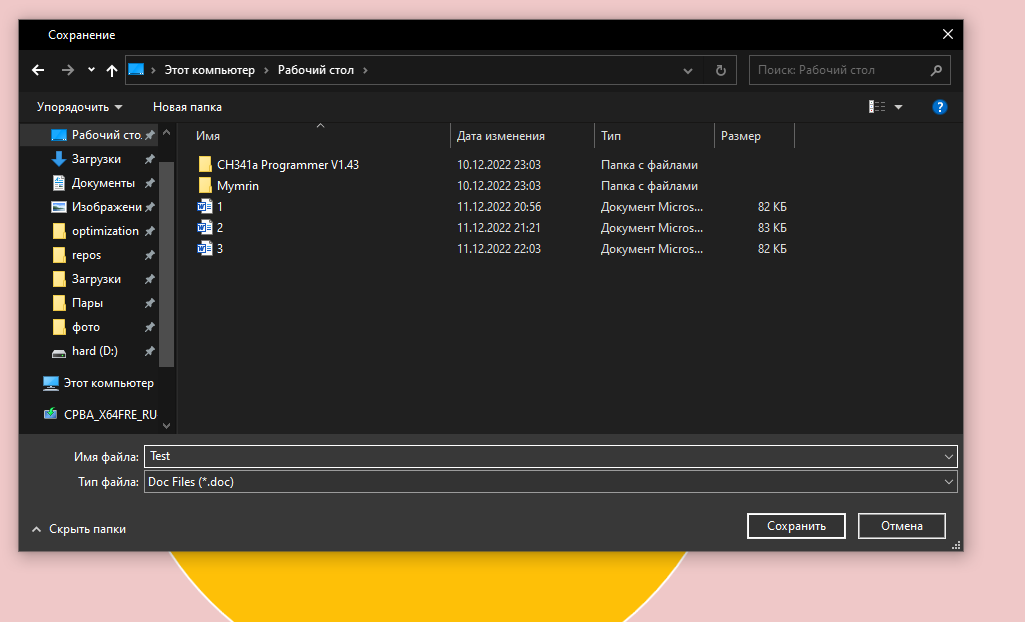


Рисунок 28. Выбор директории сохранения документа

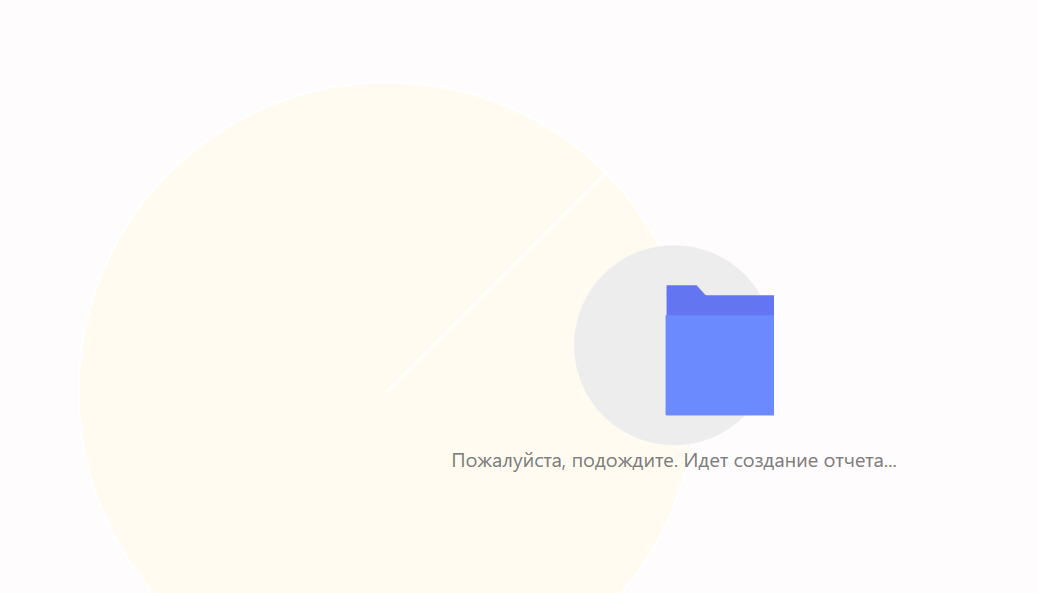


Рисунок 29. Анимация при сохранении документа

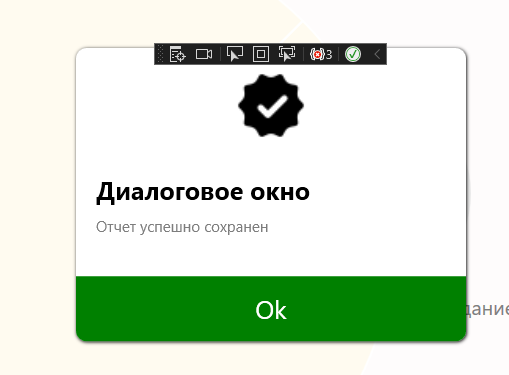


Рисунок 30. Сообщение об успешном сохранении

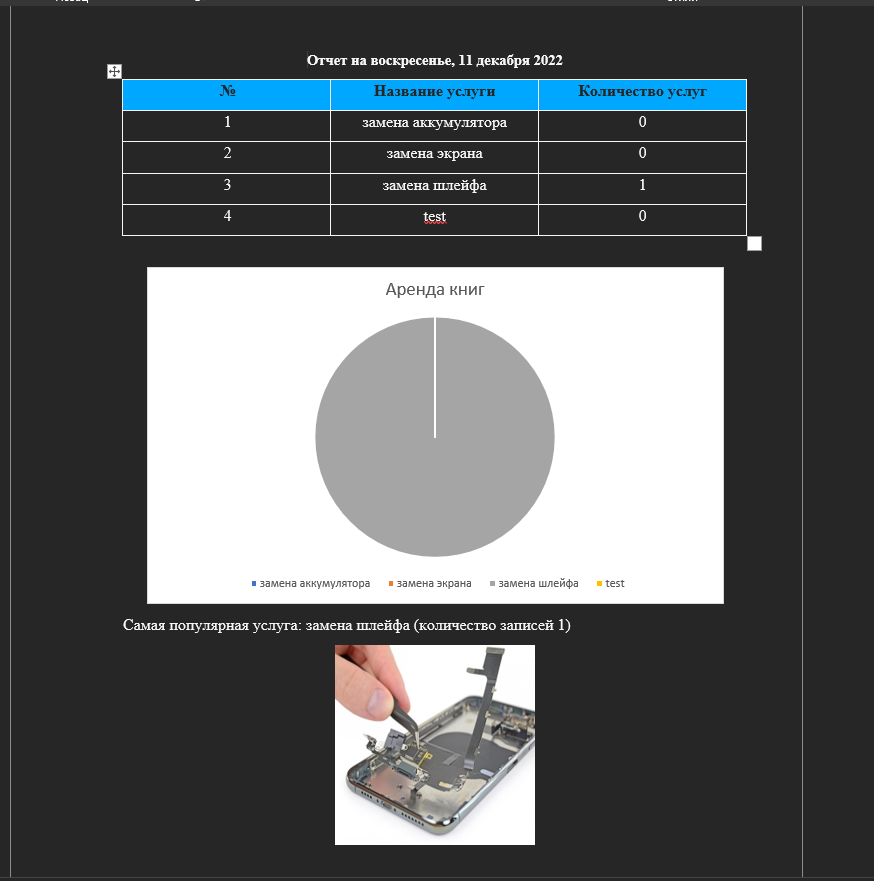


Рисунок 31. Документ с результатом работы

**Вывод:** в ходе проделанной работы было реализовано построение графиков с помощью библиотеки LiveCharts, а также программное формирование отчета в виде Word документа.

**Контрольные вопросы**

1. Что такое NuGet? Для чего он предназначен?

NuGet определяет, как создаются, размещаются и используются пакеты для .NET, а также предоставляет средства для каждой из этих ролей. Пакет NuGet представляет собой отдельный ZIP-файл с расширением .nupkg, который содержит скомпилированный код (DLL), другие файлы, связанные с этим кодом, и описательный манифест, включающий такие сведения, как номер версии пакета.

1. Что обозначают ключевые слова async и await?

Async и await, цель которых - упростить написание асинхронного кода. Async указывается в определении метода, не делает автоматически метод асинхронным. Оно лишь указывает, что данный метод может содержать одно или несколько выражений await.

1. Для чего нужен класс Task в C#?

Класс Task в C# — это операция, которая выполняется асинхронно и не возвращает значения.

1. Как в данной практической происходило обращение к основному потоку из потока для формирования отчета?

В данной практической обращение к основному потоку из потока для формирования отчета происходило с помощью асинхронности. Асинхронность позволяет вынести отдельные задачи из основного потока в специальные асинхронные методы и при этом более экономно использовать потоки. Асинхронные методы выполняются в отдельных потоках. Однако при выполнении продолжительной операции поток асинхронного метода возвратится в пул потоков и будет использоваться для других задач. А когда продолжительная операция завершит свое выполнение, для асинхронного метода опять выделяется поток из пула потоков, и асинхронный метод продолжает свою работу.

1. С каким объектами Word была произведена работа при формировании отчет?

Document, Paragraph, Table, InlineShape.

1. Для чего используется ключевое слов dynamic в с#?

Ключевое слово dynamic позволяет использовать переменные, которые не зависят от типа, а действуют как тип данных, которые они содержат.