**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КОМИ**

**Государственное профессиональное образовательное учреждение**

**«Воркутинский арктический горно-политехнический колледж»**

**КУРСОВОЙ ПРОЕКТ**

По дисциплине МДК.05.02 Разработка кода информационных систем

**Разработка информационной системы «Архив»**

Выполнил студент гр. ИСП-22 /\_\_\_\_\_\_\_\_\_\_\_\_\_/Серегин Николай Витальевич/

(подпись) (Ф.И.О.)

**ОЦЕНКА:\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

**ПРОВЕРИЛ**

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

(подпись) (Ф.И.О.)

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ 3**](#_Toc194049325)

[**ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ 4**](#_Toc194049326)

[**1.1 Платформа .NET 4**](#_Toc194049327)

[**1.2 Язык программирования С# 5**](#_Toc194049328)

[**1.3 Windows Presentation Foundation (WPF) 6**](#_Toc194049329)

[**1.4 СУБД SQL server 8**](#_Toc194049330)

[**1.5 Microsoft SQL Server Management Studio 10**](#_Toc194049331)

[**1.6 Entity Framework 12**](#_Toc194049332)

[**ГЛАВА 2. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ 15**](#_Toc194049333)

[**2.1 Разработка диаграммы ERD 15**](#_Toc194049335)

[**2.2 Разработка базы данных 18**](#_Toc194049336)

[**ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ 24**](#_Toc194049337)

[**3.1 Разработка прототипа информационной системы 24**](#_Toc194049339)

[**3.2 Программирование информационной системы 26**](#_Toc194049340)

[**3.2.1 Разработка модуля «Авторизация» 26**](#_Toc194049341)

[**3.2.2 Разработка модуля ….. 26**](#_Toc194049342)

[**3.2.3 Разработка модуля ….. 26**](#_Toc194049343)

[**3.2.4 Разработка модуля ….. 26**](#_Toc194049344)

[**3.2.5 Разработка модуля «Отчет» 26**](#_Toc194049345)

[**ЗАКЛЮЧЕНИЕ 27**](#_Toc194049346)

[**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 28**](#_Toc194049347)

# ВВЕДЕНИЕ

Современные технологии играют все более значимую роль в различных сферах жизни, в том числе в области архивного дел. Создание информационных систем для архива становится все более актуальной задачей в современно мире.

Данная система может значительно упрости работу архивных учреждений. Для более точного и эффективного хранения и поиска документов в данном курсовом проекте будут рассмотрены основные принципы создания информационной системы для архива, ее основные возможности и преимущества, а также практические аспекты ее внедрения и использования.

Эффективное управление архивными данными позволяют ускорить поиск информации, снизить риски утери важных документов и улучшить организацию работы архивных служб.

Современные информационные технологии дают возможность автоматизировать процессы обработки и хранения данных, обеспечивая удобный доступ к ним.

**Объект:** информационная система «Архив».

**Предмет:** анализ бизнес-процессов «Архив».

**Цель работы:** разработать информационную систему «Архив».

**Задачи:**

* выбрать инструментарий;
* спроектировать базу данных;
* разработать информационную систему.

# ГЛАВА 1. ВЫБОР ИНСТРУМЕНТАРИЯ

* 1. Платформа .NET

Как-то Билл Гейтс сказал, что платформа **.NET** — это лучшее, что создала компания Microsoft. Возможно, он был прав. Фреймворк **.NET** представляет мощную платформу для создания приложений. Можно выделить следующие ее основные черты:

**Поддержка нескольких языков**. Основой платформы является общеязыковая среда исполнения **Common Language Runtime** **(CLR)**, благодаря чему **.NET** поддерживает несколько языков: наряду с C# это также VB.NET, C++, F#, а также различные диалекты других языков, привязанные к **.NET**, например, **Delphi.NET**. При компиляции код на любом из этих языков компилируется в сборку на общем языке **CIL** **(Common Intermediate Language)** - своего рода ассемблер платформы **.NET**. Поэтому при определенных условиях мы можем сделать отдельные модули одного приложения на отдельных языках.

**Кроссплатформенность**. **.NET** является переносимой платформой (с некоторыми ограничениями). Например, последняя версия платформы на данный момент - **.NET** **9** поддерживается на большинстве современных ОС Windows, MacOS, Linux. Используя различные технологии на платформе .NET, можно разрабатывать приложения на языке C# для самых разных платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

**Мощная библиотека классов**. **.NET** представляет единую для всех поддерживаемых языков библиотеку классов. И какое бы приложение мы не собирались писать на C# - текстовый редактор, чат или сложный веб-сайт - так или иначе мы задействуем библиотеку классов **.NET**.

**Разнообразие технологий**. Общеязыковая среда исполнения **CLR** и базовая библиотека классов являются основой для целого стека технологий, которые разработчики могут задействовать при построении тех или иных приложений. Например, для работы с базами данных в этом стеке технологий предназначена технология **ADO.NET** и **Entity Framework Core**. Для построения графических приложений с богатым насыщенным интерфейсом - технология WPF и WinUI, для создания более простых графических приложений - Windows Forms. Для разработки кроссплатформенных мобильных и десктопных приложений - Xamarin/MAUI. Для создания веб-сайтов и веб-приложений - ASP.NET и т.д.

К этому стоит добавить активной развивающийся и набирающий популяность Blazor - фреймворк, который работает поверх **.NET** и который позволяет создавать веб-приложения как на стороне сервера, так и на стороне клиента. А в будущем будет поддерживать создание мобильных приложений и, возможно, десктоп-приложений.

**Производительность**. Согласно ряду тестов веб-приложения на **.NET** в ряде категорий сильно опережают веб-приложения, построенные с помощью других технологий. Приложения на **.NET** в принципе отличаются высокой производительностью.

* 1. Язык программирования С#

**C#** — это кроссплатформенный язык общего назначения, который делает разработчиков продуктивным при написании высокопроизводительного кода. С миллионами разработчиков **C#** является самым популярным языком .NET. **C#** имеет широкую поддержку в экосистеме и всех рабочих нагрузок .NET. На основе объектно-ориентированных принципов он включает множество функций из других парадигм, а не наименее функционального программирования. Низкоуровневые функции поддерживают сценарии высокой эффективности без написания небезопасного кода. Большая часть среды выполнения и библиотек .NET написана на **C#**, и прогресс в **C#** часто используется для всех разработчиков .NET.

**C#** поддерживает кроме примитивных типов данных (int, double, char, bool и т.д) ссылочные типы данных (классы, массивы, делегаты и т.д).

**Инкапсуляция** – скрытие внутренней реализации объекта, доступ только через публичные интерфейсы.

**Наследование** – создание новых классов на основе существующих с наследованием их свойств и методов.

**Полиморфизм –** возможность обработки объектов разных типов через общий интерфейс.

**Абстракция –** скрытие деталей реализации и фокус на важной функциональности.

**Делегат –** типы, представляющие ссылки на методы, позволяя передавать методы как параметры.

**События –** механизм, позволяющий объектам уведомлять другие объекты о произошедших изменениях или действиях.

**LINQ (Language Integrated Query) –** интегрированный механизм запросов для работы с данными в различных источниках с использованием синтаксиса, похожего на SQL.

**Асинхронное программирование –** позволяет выполнять длительные операции без блокировки основного потока программы.

**Сборщик мусора (Garbage Collector) –** система управления памятью, которая автоматически освобождает неиспользуемую память, предотвращая утечки.

Пример кода на **C#**:

using System;

class Hello

{

static void Main()

{

// This line prints "Hello, World"

Console.WriteLine("Hello, World");

}

}

* 1. Windows Presentation Foundation (WPF)

Технология **WPF** (**Windows Presentation Foundation**) является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Одной из важных особенностей является использование языка декларативной разметки интерфейса **XAML**, основанного на **XML**: вы можете создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C#, VB.NET и F#, либо совмещать и то, и другое.

Первая версия - **WPF 3.0** вышла вместе с **.NET Framework 3.0** и операционной системой Windows Vista в 2006 году. И с тех пор платформа **WPF** является частью экосистемы **.**NET и развивается вместе с фреймворком **.**NET. Например, на сегодняшний день последней версией фреймворка **.**NET является **.NET 8**, и **WPF** полностью поддерживается этой версией фреймворка.

**Преимущества WPF**

Использование традиционных языков **.NET-платформы** - C#, F# 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.

**PresentationFramework.dll:** содержит все основные реализации компонентов и элементов управления, которые можно использовать при построении графического интерфейса

**PresentationCore.dll:** содержит все базовые типы для большинства классов из PresentationFramework.dll

**WindowsBase.dll:** содержит ряд вспомогательных классов, которые применяются в WPF, но могут также использоваться и вне данной платформы

Unmanaged API используется для интеграции вышележащего уровня с DirectX:

**milcore.dll:** собственно, обеспечивает интеграцию компонентов **WPF** с DirectX. Данный компонент написан на неуправляемом коде (С/С++) для взаимодействия с **DirectX**.

**WindowsCodecs.dll:** библиотека, которая предоставляет низкоуровневую поддержку для изображений в **WPF**

Еще ниже, собственно, находятся компоненты операционной системы и DirectX, которые производят визуализацию компонентов приложения, либо выполняют прочую низкоуровневую обработку. В частности, с помощью низкоуровневого интерфейса **Direct3D**, который входит в состав **DirectX**, происходит трансляция

Здесь также на одном уровне находится библиотека **user32.dll**. И хотя выше говорилось, что **WPF** не использует эту библиотеку для рендеринга и визуализации, однако для ряда вычислительных задач (не включающих визуализацию) данная библиотека продолжает использоваться.

* 1. СУБД SQL server

**SQL Server** является одной из наиболее популярных систем управления базами данных (СУБД) в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов.

**SQL Server** характеризуется такими особенностями как:

Производительность. **SQL Server** работает очень быстро.

Надежность и безопасность. **SQL Server** предоставляет шифрование данных.

Простота. С данной СУБД относительно легко работать и вести администрирование.

Центральным аспектом в **MS SQL Server**, как и в любой СУБД, является база данных. **База данных** представляет хранилище данных, организованных определенным способом. Нередко физически база данных представляет файл на жестком диске, хотя такое соответствие необязательно. Для хранения и администрирования баз данных применяются системы управления базами данных (**database management system**) или СУБД (DBMS). И как раз **MS SQL Server** является одной из такой СУБД.

Для организации баз данных **MS SQL Server** использует реляционную модель. Реляционная модель предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта.

Для идентификации каждой строки в рамках таблицы применяется первичный ключ (**primary key**). В качестве первичного ключа может выступать один или несколько столбцов. Используя первичный ключ, мы можем ссылаться на определенную строку в таблице. Соответственно две строки не могут иметь один и тот же первичный ключ.

Через ключи одна таблица может быть связана с другой, то есть между двумя таблицами могут быть организованы связи. А сама таблица может быть представлена в виде отношения ("**relation**").

Для взаимодействия с базой данных применяется язык **SQL (Structured Query Language)**. Клиент (например, внешняя программа) отправляет запрос на языке **SQL** посредством специального API. СУБД должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Выделяются две разновидности языка **SQL**: **PL-SQL** и **T-SQL**. **PL-SQL** используется в таких СУБД как **Oracle** и **MySQL**. **T-SQL** (**Transact-SQL**) применяется в **SQL Server**.

В зависимости от задачи, которую выполняет команда **T-SQL**, он может принадлежать к одному из следующих типов:

**DDL** (Data Definition Language / Язык определения данных). К этому типу относятся различные команды, которые создают базу данных, таблицы, индексы, хранимые процедуры и т.д. В общем определяют данные.

В частности, к этому типу мы можем отнести следующие команды:

**CREATE**: создает объекты базы данных (саму базу данных, таблицы, индексы и т.д.)

**ALTER**: изменяет объекты базы данных

**DROP**: удаляет объекты базы данных

**TRUNCATE**: удаляет все данные из таблиц

**DML** (Data Manipulation Language / Язык манипуляции данными). К этому типу относят команды на выборку данных, их обновление, добавление, удаление - в общем все те команды, с помощью которыми мы можем управлять данными.

К этому типу относятся следующие команды:

**SELECT**: извлекает данные из БД

**UPDATE**: обновляет данные

**INSERT**: добавляет новые данные

**DELETE**: удаляет данные

**DCL** (Data Control Language / Язык управления доступа к данным). К этому типу относят команды, которые управляют правами по доступу к данным. В частности, это следующие команды:

**GRANT**: предоставляет права для доступа к данным

**REVOKE**: отзывает права на доступ к данным

* 1. Microsoft SQL Server Management Studio

**SQL Server Management Studio (SSMS)** — это интегрированная среда для управления любой инфраструктурой **SQL**. Используйте **SSMS** для доступа, настройки, управления, администрирования и разработки всех компонентов **SQL** Server, Базы данных **SQL Azure**, Управляемого экземпляра **SQL Azure**, **SQL Server** на виртуальной машине **Azure** и **Azure Synapse Analytics**. **SSMS** предоставляет единую комплексную служебную программу, которая сочетает в себе широкую группу графических инструментов с множеством многофункциональных редакторов сценариев для предоставления доступа к **SQL** **Server** разработчикам и администраторам баз данных любого уровня квалификации.

Среда **SQL Server Management Studio (SSMS)** позволяет управлять объектами служб **Analysis Services**, например выполнять резервное копирование и обработку объектов.

**SSMS** предоставляет проект скриптов служб **Analysis Services**, в котором можно разрабатывать и сохранять скрипты, написанные в **многомерных выражениях (MDX),** **выражениях анализа данных (DAX),** **расширениях интеллектуального анализа данных (DMX)** и **XML для анализа (XMLA).**

Эти сценарии используются для выполнения задач управления или воссоздания объектов, таких как базы данных и экземпляры, в кубах служб **Analysis Services**. Например, можно разработать сценарий **XMLA** в проекте сценария служб **Analysis Services** для создания новых объектов непосредственно в существующем экземпляре. Эти проекты могут быть сохранены как часть решения и интегрированы с системой управления исходным кодом.

**SQL Server Management Studio (SSMS)** можно использовать для управления и мониторинга выполняемых пакетов служб **SSIS**. Вы можете упорядочивать пакеты по папкам, запускать, импортировать, экспортировать и обновлять пакеты служб **Integration Services**. Однако, начиная с **SSIS 2012**, хранение пакетов изменилось. Они больше не хранятся в базе данных сервера экземпляра по умолчанию, а теперь управляются через базу данных каталога служб **SSIS** (). Это означает, что вы больше не можете управлять пакетами так же, как это было в предыдущих версиях служб **SSIS**. Вы по-прежнему можете использовать **SSMS** для управления базой данных каталога служб **SSIS**, но необходимо использовать узел каталогов служб **Integration Services** в обозревателе объектов.

Последняя версия **SSMS** предоставляет интегрированную среду для управления любой инфраструктурой **SQL**. Он также позволяет пользователям запускать пакеты служб **SSIS**, хранящиеся в каталоге служб **SSIS**, из обозревателя объектов в системе **SSMS**.

Мастер импорта и экспорта в **SSMS** можно использовать для создания пакетов служб **SSIS**, что является хорошей отправной точкой для изучения служб **SSIS**. Однако для создания пакетов для более сложных пакетов и управления ими необходимо использовать **SQL Server Data Tools (SSDT).**

Среда **SQL** **Server Management Studio** (**SSMS**) включает функции служб Reporting Services, администрирует сервер и базы данных, а также управляет ролями и заданиями. Вы можете управлять общими расписаниями с помощью папки **Shared Schedules** и управлять базами данных сервера отчетов (, ). При перемещении базы данных сервера отчетов в новый экземпляр **SQL Server** необходимо создать роль **RSExecRole** в системной базе данных.

* 1. Entity Framework

**Entity Framework** представляет **ORM**-технологию (**object-relational mapping** - отображения данных на реальные объекты) от компании Microsoft для доступа к данным. **Entity Framework Core** позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными как с объектами классом независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает **Entity Framework**, мы уже работаем с объектами.

Поскольку **Entity Framework Core** работает на основе платформы .NET, то он развивается вместе с данной платформой. Текущая версия **EF Core - 9.0** была выпущена в ноябре 2024 года вместе с .NET 8., и технология продолжает развиваться.

**Entity Framework Core** поддерживает множество различных систем баз данных. Таким образом, мы можем через **EF Core** работать с любой СУБД, если для нее имеется нужный провайдер. По умолчанию на данный момент Microsoft предоставляет ряд встроенных провайдеров: для работы с **MS SQL Server**, для **SQLite**, для **PostgreSQL**. Также имеются провайдеры от сторонних поставщиков, например, для **MySQL**.

Центральной концепцией **Entity Framework** является понятие сущности или **entity**. Сущность определяет набор данных, которые связаны с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их коллекциями.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст. Свойства необязательно представляют простые данные типа int или string, но могут также представлять и более комплексные типы данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой **Entity Framework Core**, как технологии **ORM**, является использование запросов **LINQ** для выборки данных из БД. С помощью **LINQ** мы можем создавать различные запросы на выборку объектов, в том числе связанных различными ассоциативными связями. А **Entity Framework** при выполнении запроса транслирует выражения **LINQ** в выражения, понятные для конкретной СУБД (как правило, в выражения SQL).

Основная функциональность **Entity Framework Core** сосредоточена в

следующих пакетах:

Microsoft.EntityFrameworkCore: основной пакет EF Core

Microsoft.EntityFrameworkCore.SqlServer: представляет функциональность

провайдера для Microsoft SQL Server и SQL Azure

Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite: предоставляет

поддержку географических типов (spatial types) для SQL Server

Microsoft.EntityFrameworkCore.Sqlite: представляет функциональность

провайдера для SQLite и включает нативные бинарные файлы для движка базы данных

Microsoft.EntityFrameworkCore.Sqlite.Core: представляет

функциональность провайдера для SQLite, но в отличие от предыдущего пакета

не содержит нативные бинарные файлы для движка базы данных

Microsoft.EntityFrameworkCore.Sqlite.NetTopologySuite: предоставляет

поддержку географических типов (spatial types) для SQLite

Microsoft.EntityFrameworkCore.Cosmos: представляет функциональность

провайдера для Azure Cosmos DB

Microsoft.EntityFrameworkCore.InMemory: представляет

функциональность провайдера базы данных в памяти

Microsoft.EntityFrameworkCore.Tools: содержит команды EF Core

PowerShell для Visual Studio Package Manager Console; применяется в Visual

Studio для миграций и генерации классов по готовой бд

Microsoft.EntityFrameworkCore.Design: содержит вспомогательные

компоненты EF Core, применяемые в процессе разработки

Microsoft.EntityFrameworkCore.Proxies: хранит функциональность для так

называемой "ленивой загрузки" (lazy-loading) и прокси остлеживания изменений

Microsoft.EntityFrameworkCore.Abstractions: содержит набор абстракций

EF Core, которые не зависят от конкретной СУБД

Microsoft.EntityFrameworkCore.Relational: хранит компоненты EF Core для

провайдеров реляционных СУБД

Microsoft.EntityFrameworkCore.Analyzers: содержит функционал

анализаторов C# для EF Core

# ГЛАВА 2. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

1. 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. **ER-диаграммы** (или **ER-модели**) полагаются на стандартный набор символов, включая прямоугольники, ромбы, овалы и соединительные линии, для отображения сущностей, их атрибутов и связей. Эти диаграммы устроены по тому же принципу, что и грамматические структуры: сущности выполняют роль существительных, а связи — глаголов.

В ER-моделях и моделях данных обычно выделяют до трех уровней детализации:

**Концептуальная модель данных**

Схема наивысшего уровня с минимальным количеством подробностей. Достоинство этого подхода заключается в возможности отобразить общую структуру модели и всю архитектуру системы. Менее масштабные системы могут обойтись и без этой модели. В этом случае можно сразу переходить к логической модели.

**Логическая модель данных**

Содержит более подробную информацию, нежели концептуальная модель. На этом уровне определяются более подробные операционные и транзакционные сущности. Логическая модель не зависит от технологии, в которой она будет применяться.

**Физическая модель данных**

На основе каждой логической модели данных можно составить одну или две физических модели. В последних должно присутствовать достаточно технических подробностей для составления и внедрения самой базы данных.

Диаграммы «сущность-связь» (или **ERD**) — неотъемлемая составляющая процесса моделирования любых систем, включая простые и сложные базы данных, однако применяемые в них фигуры и способы нотации могут запросто ввести в заблуждение любого.

Для того чтобы построить ER-диаграмму, можно использовать разные нотации. Три самые известные из них:

1. **нотация IDEF1X.** Её относят к фундаментальным, но на практике

давно не используют, потому что есть более удобные варианты.

1. **нотация Чена.** Классическая нотация, которая состоит из простых

символов — прямоугольников, овалов и линий. Из-за этого нотацию часто используют для концептуальных моделей, которые презентуют заказчику. Человеку, который далёк от аналитики данных, проще разобраться в понятных диаграммах со знакомыми символами.

1. **нотация Мартина.** Её ещё называют «воронья лапка» (от англ.

**Crow's Foot**). Она компактнее нотации Чена, поэтому её используют для построения ER-моделей логического уровня, когда нужно описать в модели все атрибуты сущностей.

В нотациях Чена и Мартина есть одинаковые элементы: сущности, атрибуты и связи. Но эти элементы диаграмм обозначают разными символами.

Элементы ER-диаграммы в нотации Чена соединяют линиями. Если линия соединяет две сущности, сверху обозначают тип связи:

* 1:1 — «один-к-одному»;
* 1: N — «один-ко-многим»;
* M: N — «многие-ко-многим».

В нотации Мартина сущность также вписывают в прямоугольник, а атрибуты и связи обозначают по-другому:

* атрибуты перечисляют прямо под сущностью;
* связи рисуют разными соединительными линиями.

**Концептуальные модели** данных дают общее представление о том, что должно входить в состав модели. Концептуальные **ER-диаграммы** можно брать за основу логических моделей данных. Их также можно использовать для создания отношений общности между разными **ER-моделями**, положив их в основу интеграции.

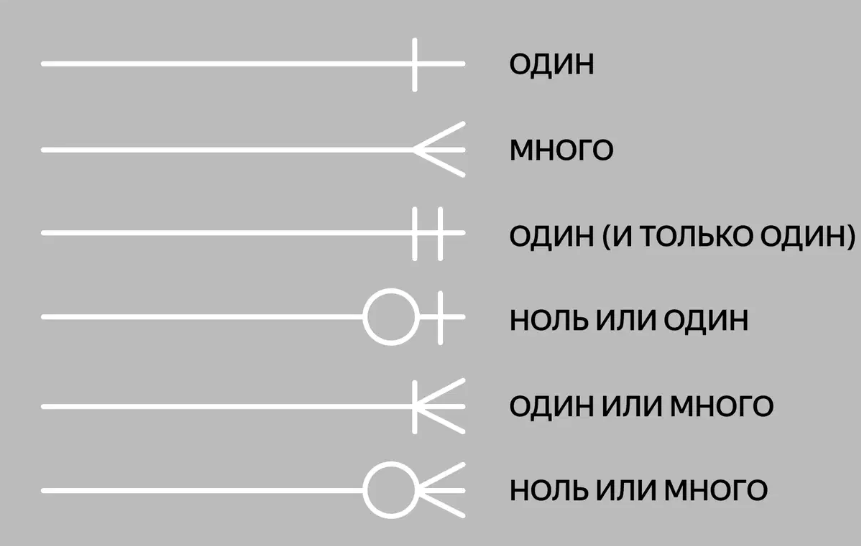


Рисунок 2.1 Виды связей в ER диаграмме



Рисунок 2.2 ER диаграмма

* 1. Разработка базы данных

Разработка базы данных будет происходить в программе **SSMS (SQL Server Management Studio).**

**SQL Server Management Studio (SSMS)** — это интегрированная среда для управления любой инфраструктурой **SQL**. Используйте **SSMS** для доступа, настройки, управления, администрирования и разработки всех компонентов **SQL** **Server**, Базы данных **SQL Azure**, Управляемого экземпляра **SQL Azure**, **SQL Server** на виртуальной машине **Azure** и **Azure Synapse Analytics**. **SSMS** предоставляет единую комплексную служебную программу, которая сочетает в себе широкую группу графических инструментов с множеством многофункциональных редакторов сценариев для предоставления доступа к **SQL** **Server** разработчикам и администраторам баз данных любого уровня квалификации.

Среда **SQL Server Management Studio (SSMS)** позволяет управлять объектами служб **Analysis Services**, например выполнять резервное копирование и обработку объектов.

Для того чтобы создать в программе SSMS базу данных необходимо нажать правой кнопкой мышки по пункту в меню справа Databases (Базы данных) и нажать на пункт New Database (Создать Базу данных) как показано на рисунке 2.3.

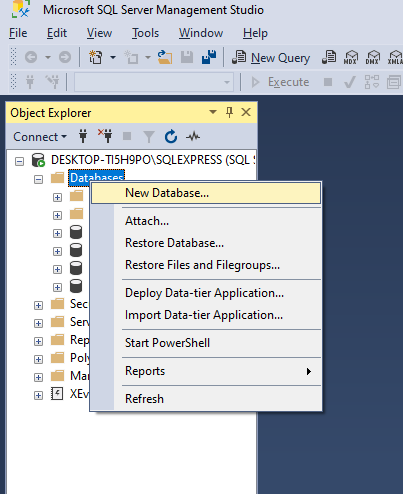


Рисунок 2.3 Меню создания базы данных

После откроется диалоговое окно, в котором нужно дать название Базе данных как показано на рисунке 2.4.

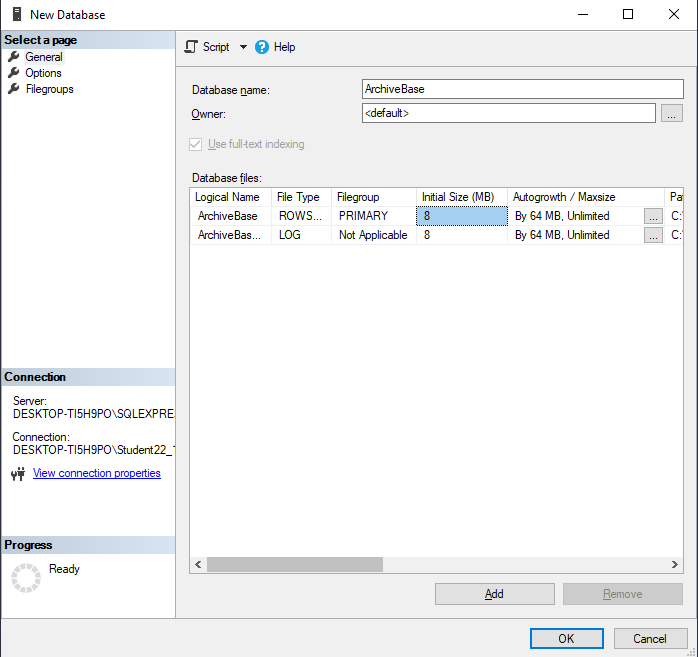


Рисунок 2.4 Диалоговое окно создания базы данных

Для того чтобы создать диаграмму базы данных необходимо раскрыть в меню базу данных и нажать правой кнопкой мыши по Database Diagrams (Диаграммы базы данных) и выбрать пункт New Database Diagram (Создать Диаграмму Базы данных) как показано на рисунке 2.5.

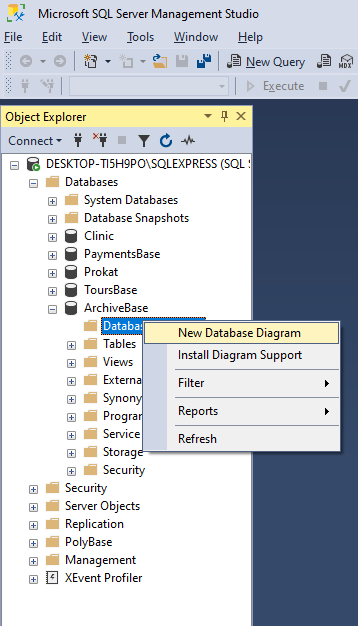


Рисунок 2.5 Меню создания диаграммы базы данных

После чего откроется пустая база данных и диалоговое окно с выбором таблиц. Теперь необходимо создать таблицы, для того чтобы создать таблицы следует нажать на пустое место на диаграмме правой кнопкой мыши и выбрать пункт New Table (Создать таблицу) как показано на рисунке 2.6, после чего появится диалоговое окно (рисунок 2.7), в котором нужно дать название таблице.

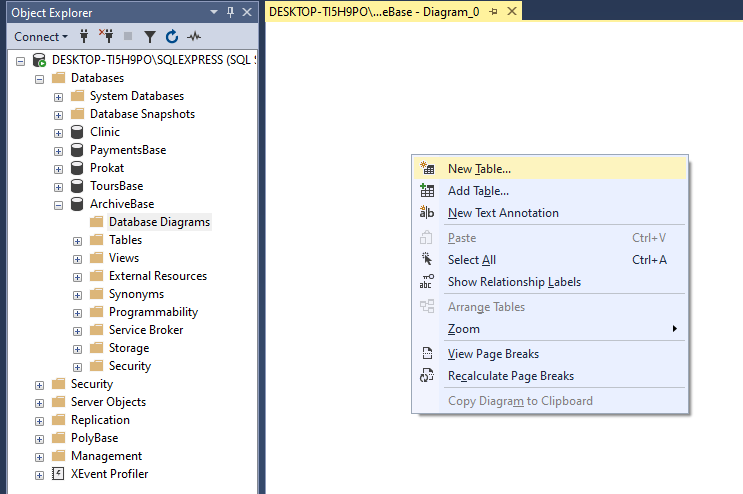


Рисунок 2.6 Меню создания таблицы в базе данных

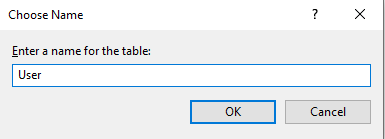


Рисунок 2.7 Диалоговое окно создания таблицы

Заполним поля в таблице основываясь на ER диаграмме (Рисунок 2.8), после чего задаем первичный ключ (Рисунок 2.9)

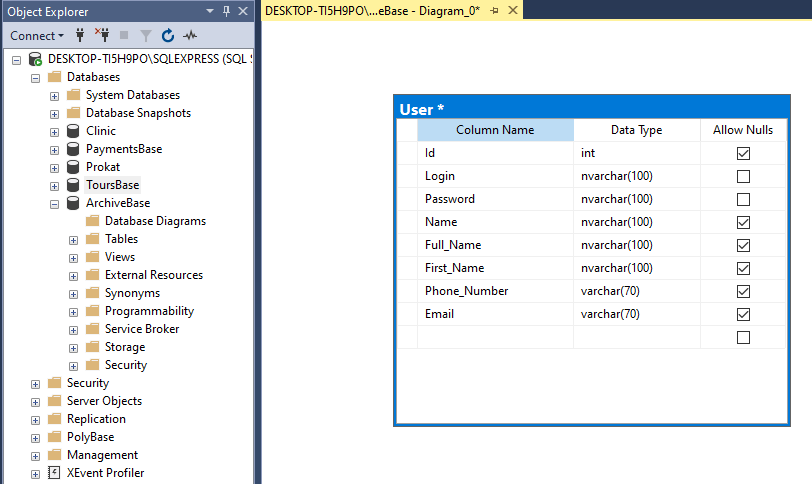


Рисунок 2.8 Заполненные поля таблицы

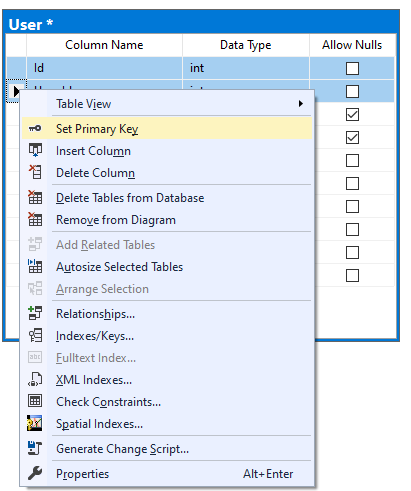


Рисунок 2.9 Присвоение первичных ключей

Проделать тоже самое с остальными таблицами и необходимо сделать связи, для того чтобы сделать связи следует выбрать первичны ключ таблицы и потянуть мышкой до вторичного ключа другой таблицы, связь должна выглядеть как на рисунке 2.10.

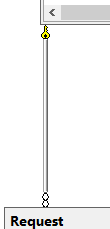


Рисунок 2.10 Связь между таблицами

Проделываем тоже самое с остальными связями и в итоге получится вот такая диаграмма (рисунок 2.11)

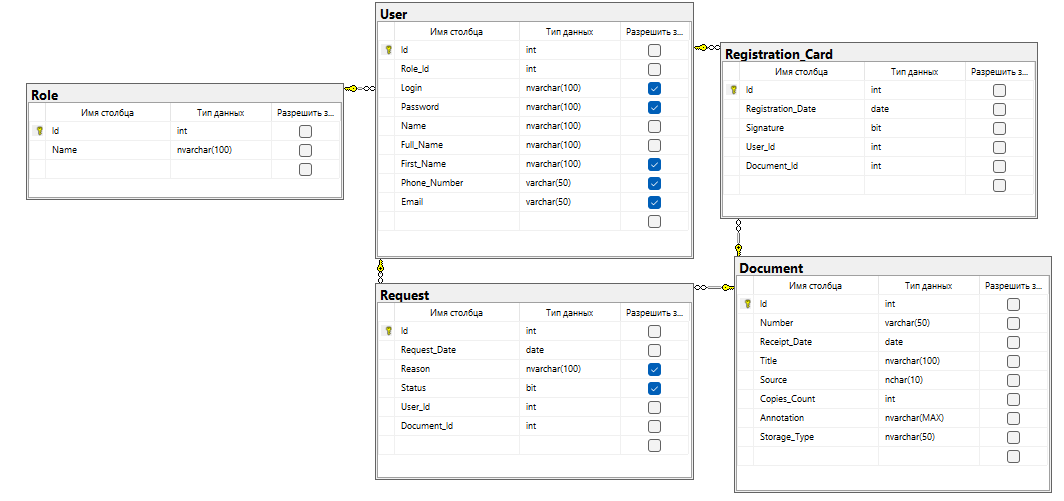


Рисунок 2.11 Диаграмма базы данных

В данной работе используется 5 таблиц базы данных. Сущность user (пользователь) имеет атрибуты: Id (уникальный ключ), Role\_Id (Внешний ключ), Login (Логин), Password (Пароль), Name (Имя), Full\_Name (Фамилия), First\_Name (Отчество), Phone\_Number (Номер телефона) и Email (почта). Сущность role (роль пользователя) имеет атрибуты: Id (Уникальный ключ) и Name (Название). Сущность document (документ) имеет атрибуты: Id (Уникальный ключ), Document\_Number (Номер документа), Receipt\_Date (Дата получения) Title (Заголовок), Source (Источник), Copies\_Count (Количество копий), Annotation (Аннотация) и Storage\_Type (Тип хранения). Сущность Registration\_Card (Регистрационная карта) имеет атрибуты: Id (Уникальный ключ), Registration\_Date (Дата регистрации), Signature (Подпись), User\_Id (Внешний ключ) и Document\_Id (Внешний ключ). Сущность request (запрос) имеет атрибуты: Id (Уникальный ключ), Request\_Date (Дата запроса), Reason (Основание), Status (Статус), User\_Id (Внешний ключ) и Document\_Id (Внешний ключ).

# ГЛАВА 3. РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ

1. 1. Разработка прототипа информационной системы

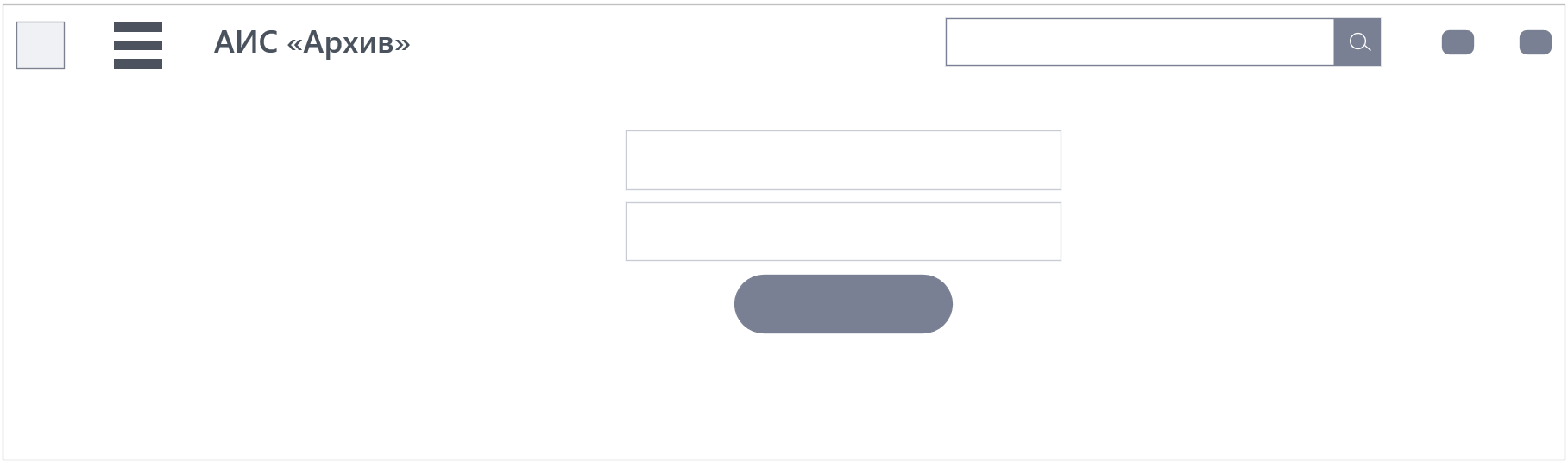


Рисунок 3.1.1 Окно «Авторизация»

На рисунке представлено окно «Авторизация», на котором показаны поля ввода и кнопка авторизации.

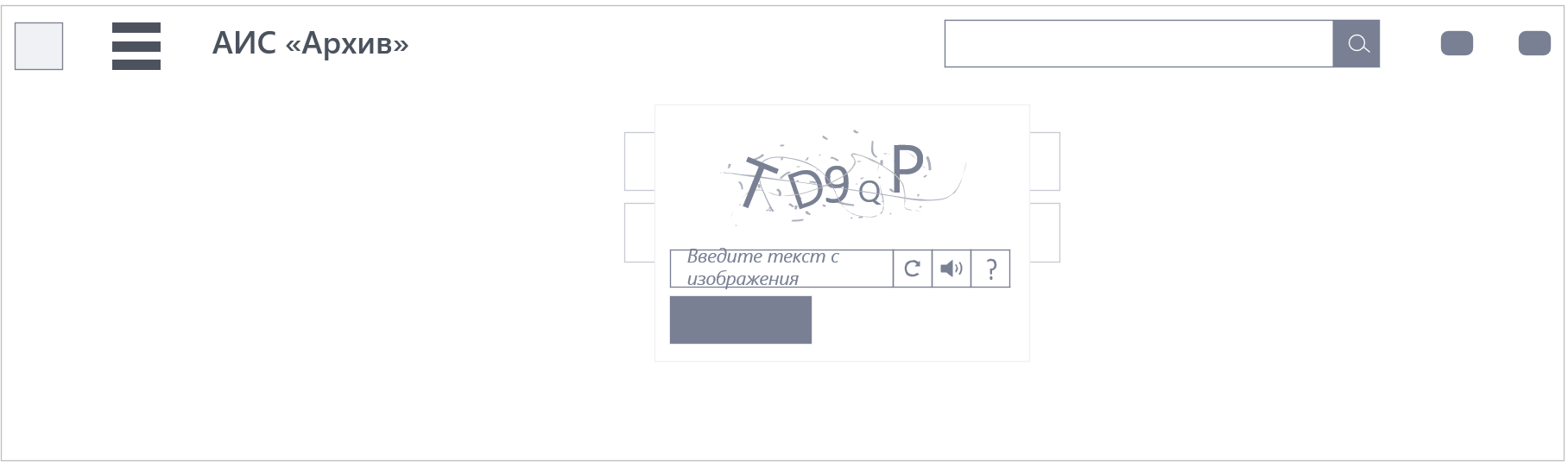


Рисунок 3.1.2 Окно «Капча»

На рисунке представлено окно «Капча», на котором показано диалоговое окно капчи.

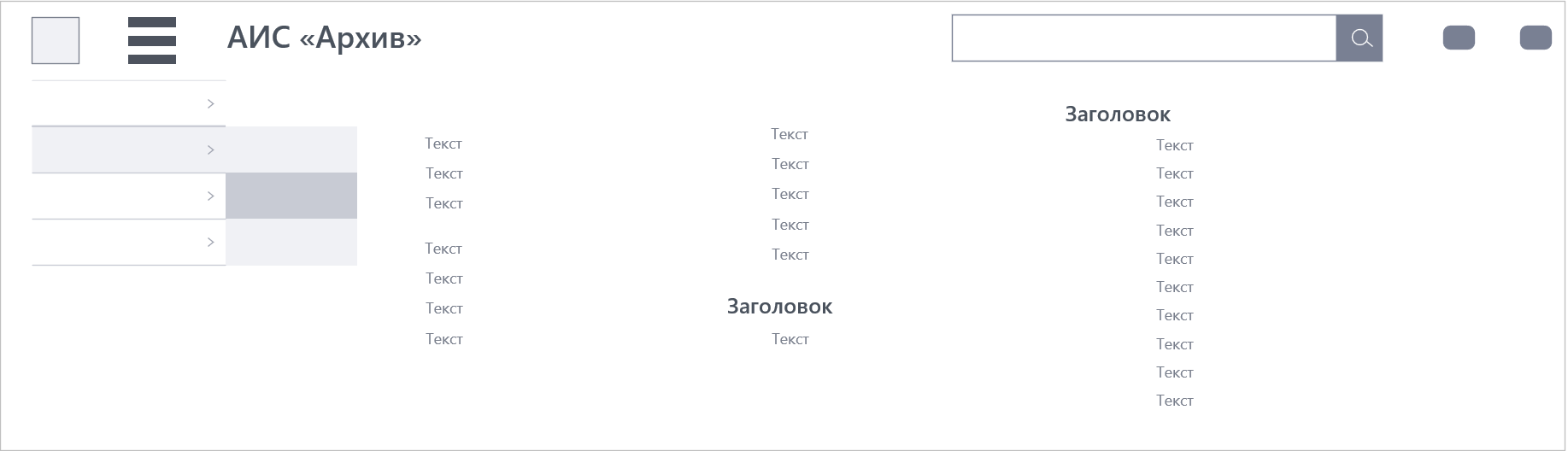


Рисунок 3.1.3 Окно «Главное меню»

На рисунке представлено окно «Главное меню», на котором показан основной функционал программы.

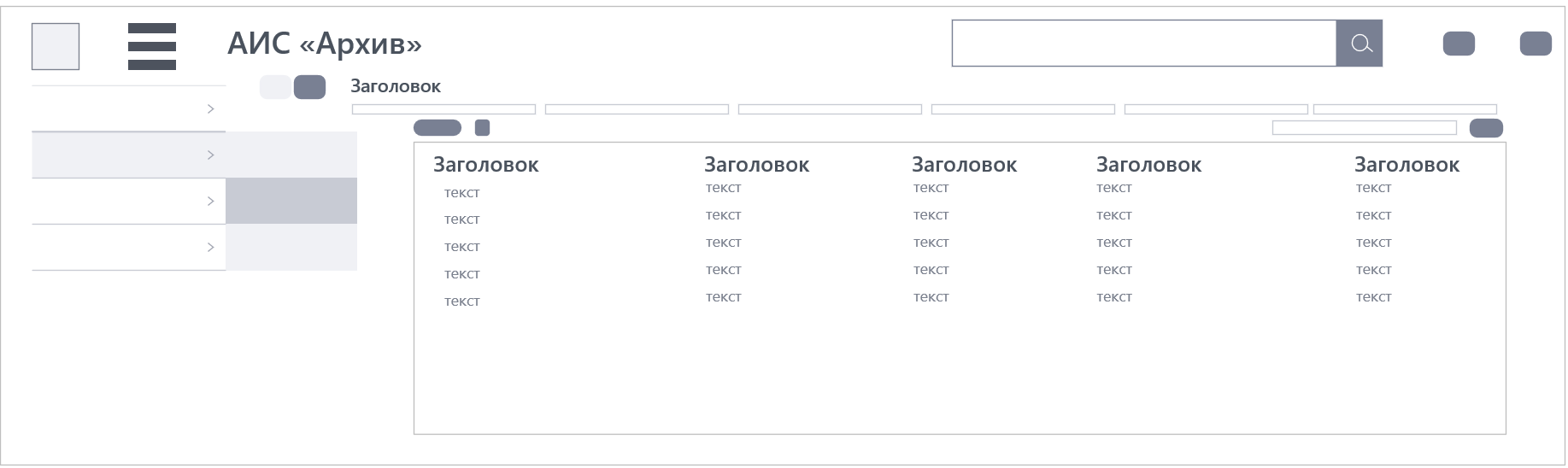


Рисунок 3.1.4 Окно «Документы»

На рисунке представлено окно «Документы», на котором показаны таблица с документами, поля поиска и сортировки.

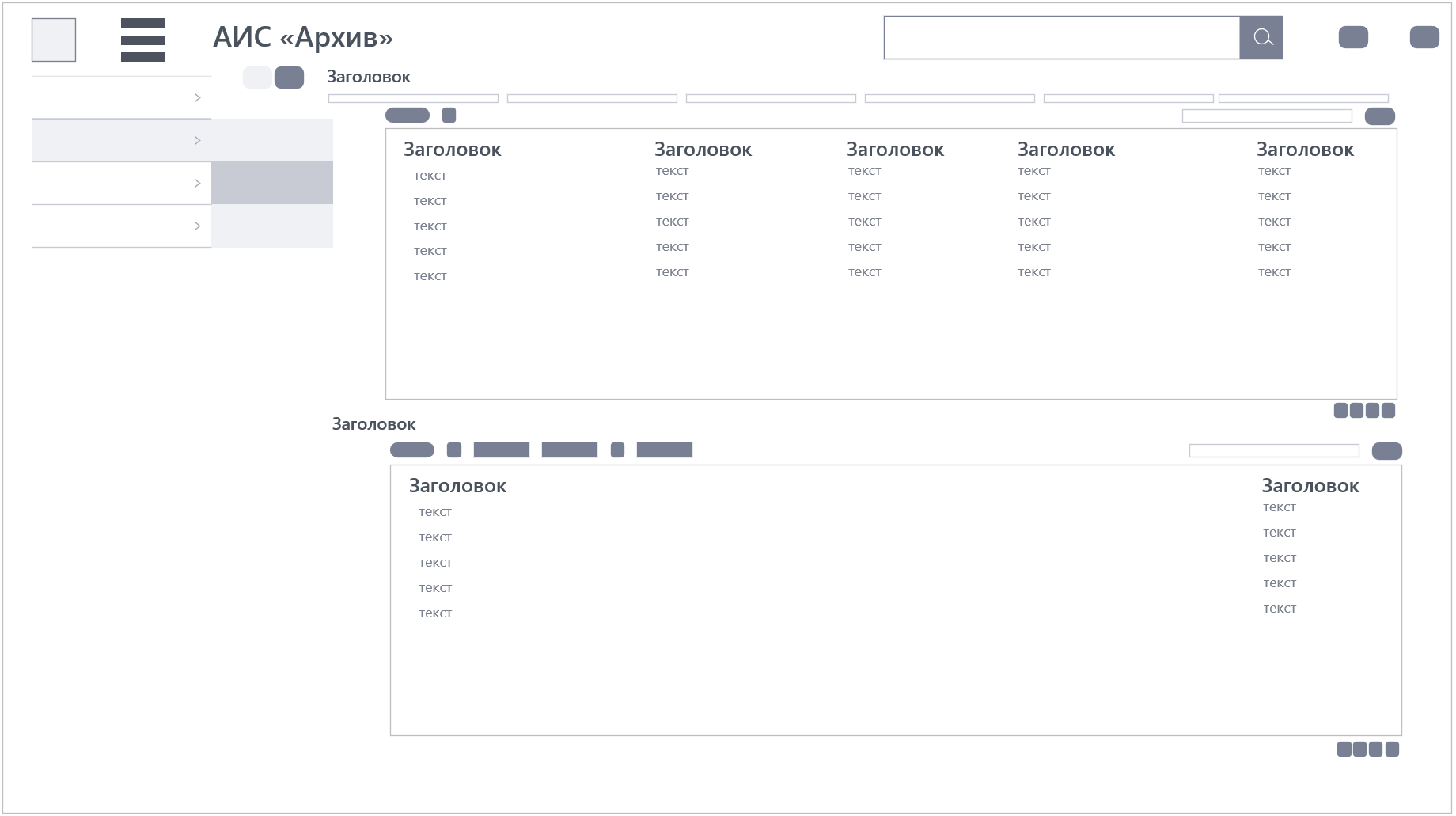


Рисунок 3.1.5 Окно «Документы и задачи»

На рисунке представлено окно «Документы и задачи», на котором показаны таблица с документами, таблица с задачами, поля поиска и сортировки данных.

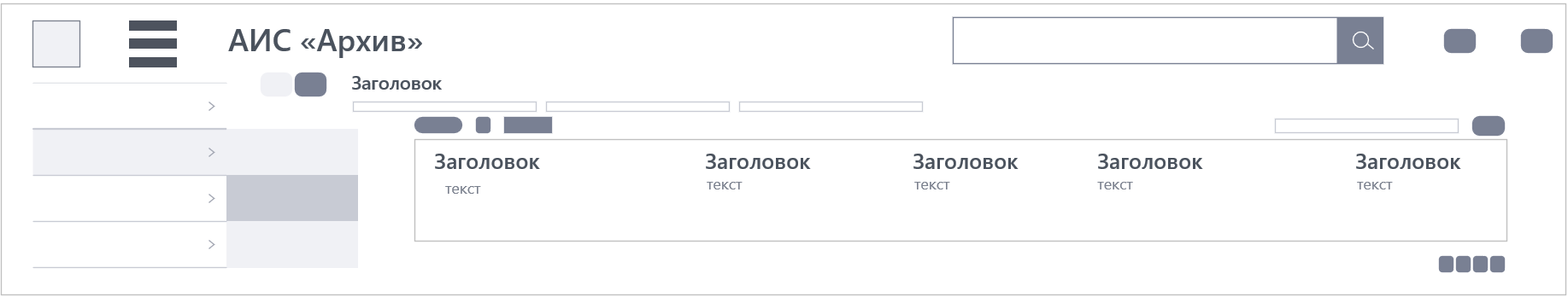


Рисунок 3.1.6 Окно «Запросы»

На рисунке представлено окно «Запросы», на котором показаны таблица запросов, поля поиска и сортировки данных, кнопки для штрих-кода и печати.

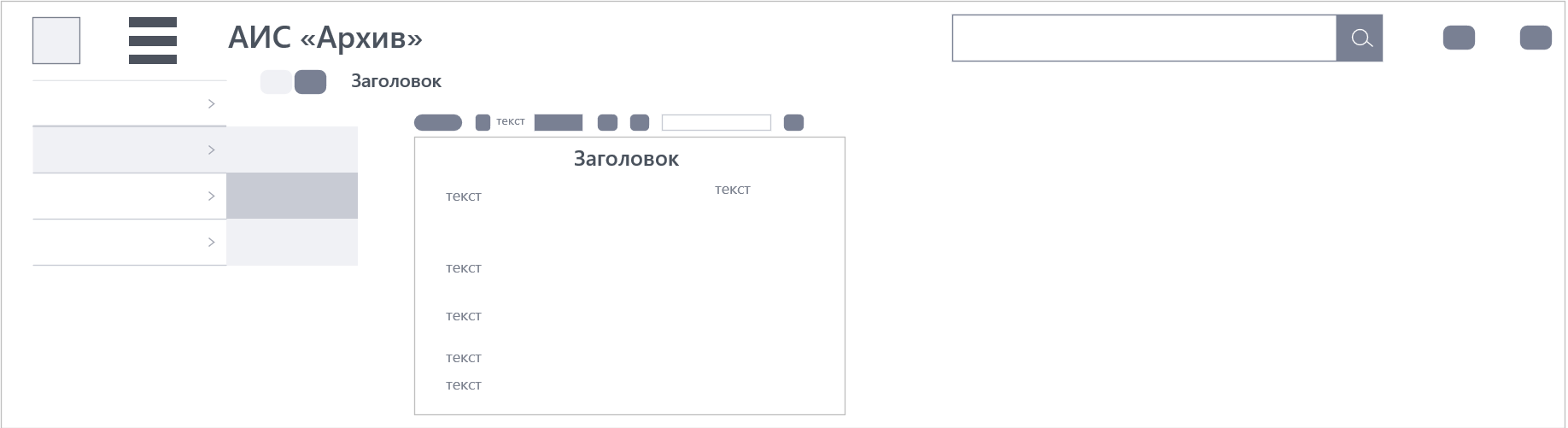


Рисунок 3.1.7 Окно «Содержание запроса»

На рисунке представлено окно «Содержание запроса», на котором показано описание запроса.

* 1. Программирование информационной системы
     1. Разработка модуля «Авторизация»

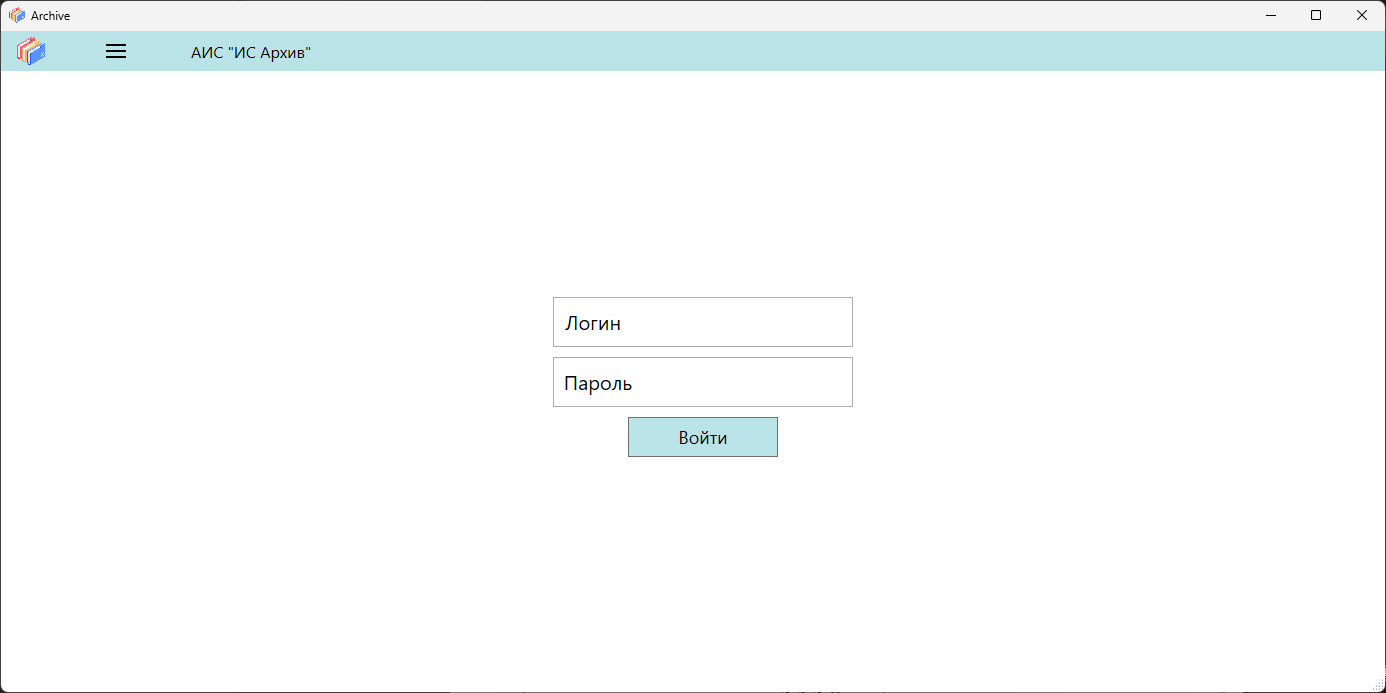


Рисунок 3.2.1.1 Окно «Авторизация»

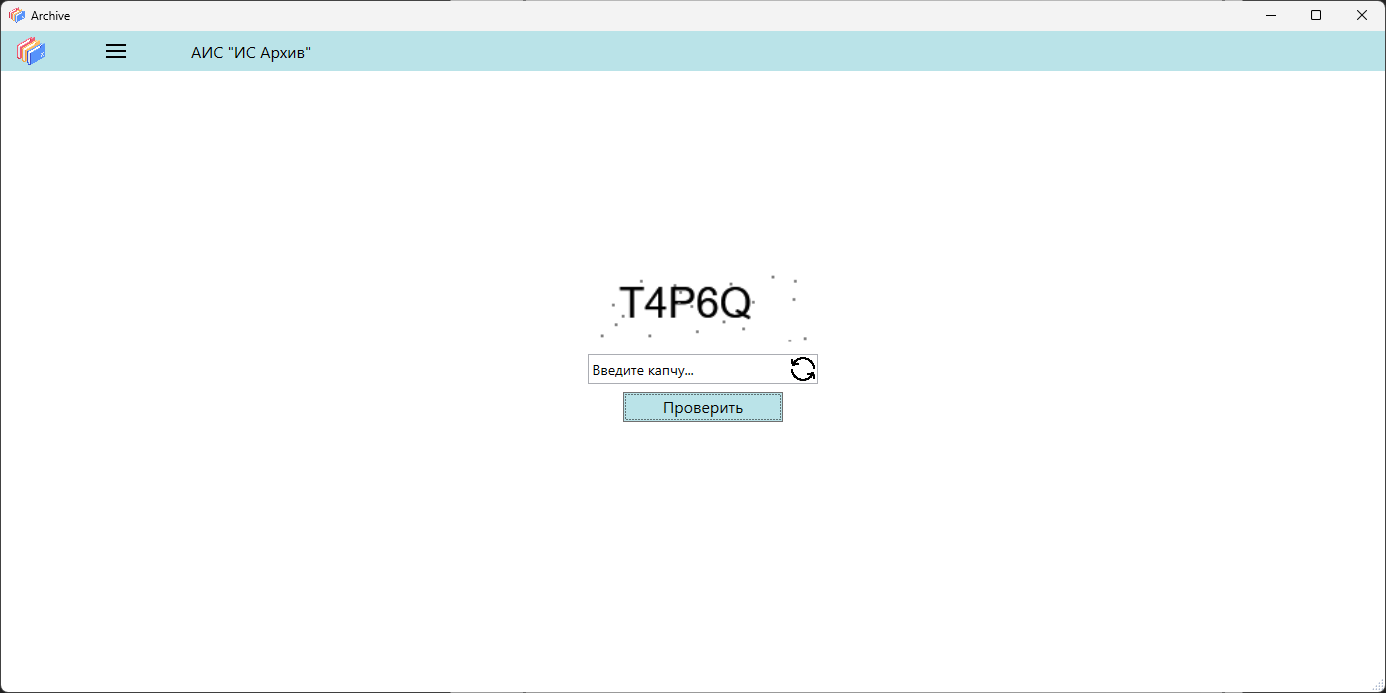


Рисунок 3.2.1.2 Окно «Капча»

Методы проверки логина, пароля и капчи:

private void VerifyCredentials()

{

string login = LoginBox.Text.Trim(); // Получение логина без пробелов

string password = PasswordBox.Password.Trim(); // Получение пароля без пробелов

StringBuilder errorMessage = new StringBuilder(); // Сбор ошибок ввода

if (string.IsNullOrWhiteSpace(password)) // Проверка пустого пароля

errorMessage.AppendLine("Введите пароль!");

if (string.IsNullOrWhiteSpace(login)) // Проверка пустого логина

errorMessage.AppendLine("Введите логин!");

if (errorMessage.Length > 0) // Показ ошибки, если поля пустые

{

ShowError(errorMessage.ToString());

return;

}

var user = ArchiveBaseEntities.GetContext().User.AsEnumerable() // Поиск пользователя

.FirstOrDefault(u => u.Login == login);

if (user == null || user.Password != password) // Проверка корректности данных

{

\_failedAttempts++; // Увеличение счетчика неудачных попыток

ShowError(user == null ? "Неверный логин!" : "Неверный пароль!"); // Сообщение об ошибке

\_pendingLogin = null; // Очистка неверного логина

\_pendingPassword = null; // Очистка неверного пароля

\_credentialsVerified = false; // Сброс флага проверки

if (IsCaptchaInGracePeriod()) // Пропуск капчи в режиме милосердия

return;

if (\_failedAttempts >= 3) // Показ капчи после 3 неудач

{

HideError(); // Скрытие ошибки

\_pendingLogin = login; // Сохранение логина для проверки

\_pendingPassword = password; // Сохранение пароля для проверки

RequestCaptcha(); // Запрос капчи

}

return;

}

\_pendingLogin = login; // Сохранение верного логина

\_pendingPassword = password; // Сохранение верного пароля

\_credentialsVerified = true; // Установка флага успешной проверки

AuthorizeUser(); // Авторизация пользователя

}

private void AuthorizeWithCaptcha()

{

string enteredCaptcha = CaptchaTextBox.Text.Trim(); // Получение введенной капчи

if (string.IsNullOrWhiteSpace(enteredCaptcha) || enteredCaptcha != \_captchaText) // Проверка корректности капчи

{

ShowError("Неверная капча! Попробуйте еще раз."); // Сообщение об ошибке

GenerateNewCaptcha(); // Обновление капчи

return;

}

\_captchaGraceUntil = DateTime.Now.Add(\_captchaGracePeriod); // Установка режима милосердия

if (string.IsNullOrWhiteSpace(\_pendingLogin) || string.IsNullOrWhiteSpace(\_pendingPassword)) // Проверка наличия данных

{

ShowError("Введите логин и пароль заново!"); // Сообщение об ошибке

HideCaptchaUI(); // Возврат к форме входа

return;

}

var user = ArchiveBaseEntities.GetContext().User.AsEnumerable() // Повторная проверка данных

.FirstOrDefault(u => u.Login == \_pendingLogin && u.Password == \_pendingPassword);

if (user == null) // Обработка неверных данных

{

\_failedAttempts++; // Увеличение счетчика неудач

ShowError("Неверный логин или пароль!"); // Сообщение об ошибке

HideCaptchaUI(); // Возврат к форме входа

return;

}

\_credentialsVerified = true; // Установка флага проверки

HideCaptchaUI(); // Скрытие капчи

HideError(); // Скрытие ошибки

AuthorizeUser(); // Авторизация пользователя

}

private void AuthorizeUser()

{

if (string.IsNullOrWhiteSpace(\_pendingLogin) || string.IsNullOrWhiteSpace(\_pendingPassword)) // Проверка наличия данных

{

ShowError("Ошибка авторизации: данные отсутствуют!"); // Сообщение об ошибке

return;

}

var user = ArchiveBaseEntities.GetContext().User // Поиск пользователя для авторизации

.FirstOrDefault(u => u.Login == \_pendingLogin && u.Password == \_pendingPassword);

if (user == null) // Обработка ошибки авторизации

{

ShowError("Ошибка авторизации: неверные данные!"); // Сообщение об ошибке

\_pendingLogin = null; // Очистка логина

\_pendingPassword = null; // Очистка пароля

\_credentialsVerified = false; // Сброс флага проверки

return;

}

Класс генерации капчи:

public static class CaptchaGenerator

{

private static Random \_random = new Random(); // Генератор случайных чисел

public static string GenerateCaptchaText(int length = 5)

{

const string chars = "ABCDEFGHJKLMNPQRSTUVWXYZ123456789abcdefghjklmnpqrstuvwxyz!@#$%^&\*()-\_=+"; // Допустимые символы без похожих (I, O, 1, 0)

char[] captcha = new char[length]; // Массив для хранения символов капчи

for (int i = 0; i < length; i++) // Заполнение массива случайными символами

captcha[i] = chars[\_random.Next(chars.Length)]; // Выбор случайного символа

return new string(captcha); // Преобразование массива в строку

}

public static BitmapImage GenerateCaptchaImage(string captchaText)

{

int width = 150, height = 50; // Размеры изображения капчи

DrawingVisual visual = new DrawingVisual(); // Объект для рисования

using (DrawingContext dc = visual.RenderOpen()) // Открытие контекста рисования

{

dc.DrawRectangle(Brushes.White, null, new Rect(0, 0, width, height)); // Отрисовка белого фона

Typeface typeface = new Typeface("Arial"); // Шрифт для текста капчи

FormattedText formattedText = new FormattedText( // Форматированный текст капчи

captchaText, // Текст для отображения

CultureInfo.InvariantCulture, // Инвариантная культура

FlowDirection.LeftToRight, // Направление текста слева направо

typeface, // Выбранный шрифт

30, // Размер шрифта

Brushes.Black, // Цвет текста

1.0); // Плотность пикселей

dc.DrawText(formattedText, new Point(15, 5)); // Отрисовка текста на изображении

for (int i = 0; i < 20; i++) // Добавление шума в виде случайных точек

{

double x = \_random.Next(width); // Случайная X-координата

double y = \_random.Next(height);// Случайная Y-координата

dc.DrawRectangle(Brushes.Gray, null, new Rect(x, y, 2, 2)); // Отрисовка серой точки

}

}

RenderTargetBitmap bitmap = new RenderTargetBitmap( // Создание растрового изображения

width, height, // Размеры изображения

96, 96, // Разрешение DPI

PixelFormats.Pbgra32); // Формат пикселей с альфа-каналом

bitmap.Render(visual); // Рендеринг визуального объекта в битмап

return ConvertBitmapToBitmapImage(bitmap); // Конвертация в BitmapImage

}

private static BitmapImage ConvertBitmapToBitmapImage(BitmapSource bitmap)

{

using (MemoryStream memory = new MemoryStream()) // Создание потока памяти

{

PngBitmapEncoder encoder = new PngBitmapEncoder(); // Кодировщик для PNG формата

encoder.Frames.Add(BitmapFrame.Create(bitmap)); // Добавление кадра из битмапа

encoder.Save(memory); // Сохранение в поток памяти

BitmapImage bitmapImage = new BitmapImage(); // Создание объекта BitmapImage

bitmapImage.BeginInit(); // Начало инициализации

bitmapImage.StreamSource = new MemoryStream(memory.ToArray()); // Установка источника данных

bitmapImage.CacheOption = BitmapCacheOption.OnLoad; // Загрузка в память сразу

bitmapImage.EndInit(); // Завершение инициализации

bitmapImage.Freeze(); // Заморозка для потокобезопасности

return bitmapImage; // Возврат готового изображения

}

}

}

В данном модуле было разработано окно авторизации и капча с проверкой логина и пароля пользователя, если 3 раза пользователь ввел неверный логин или пароль, то показывается капча если же он с первого раза ввел верный логин и пароль, то пользователь входит в систему.

* + 1. Разработка модуля «Главное меню»

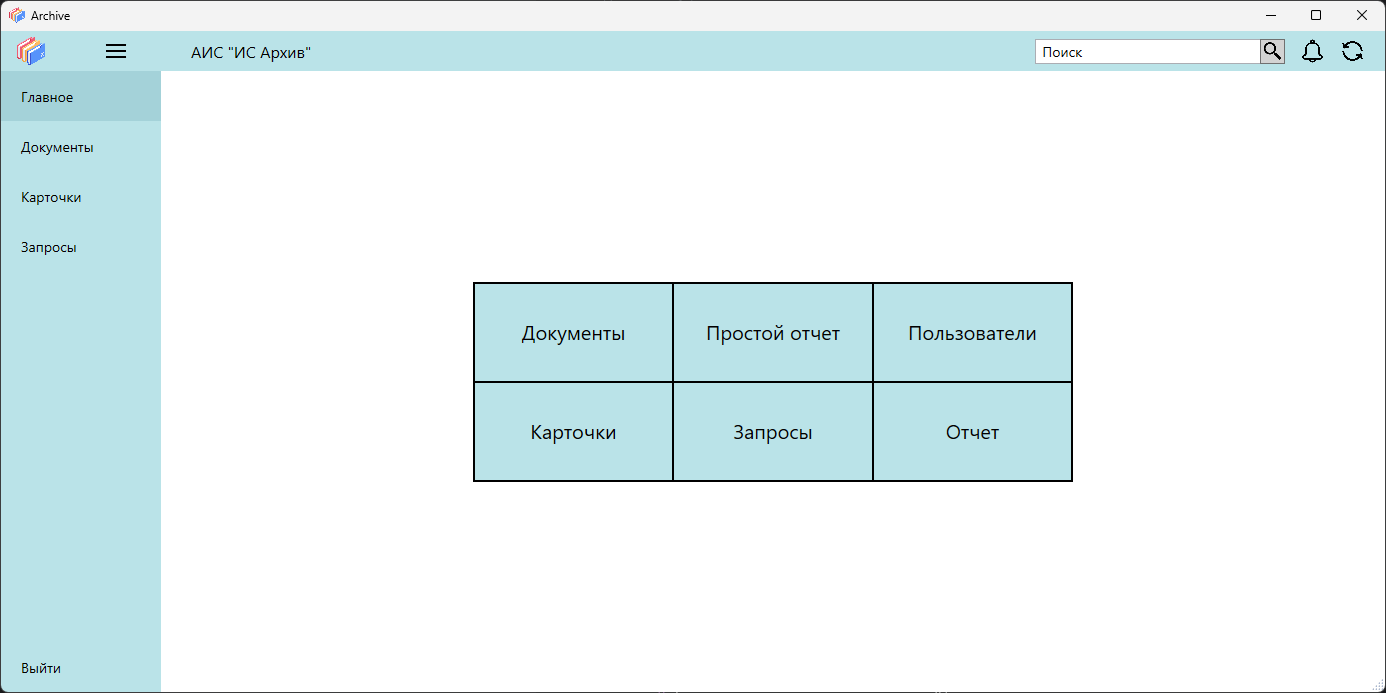


Рисунок 3.2.2 Окно «Главное меню»

Методы разграничения прав пользователей:

private void SetPermissionsBasedOnRole()

{

switch (\_Role) // Настройка интерфейса в зависимости от роли

{

case "Администратор": // Полный доступ для администратора

AdminControlsVisibility(true);

ClerkControlsVisibility(true);

ArchivariusControlsVisibility(true);

break;

case "Делопроизводитель": // Ограниченный доступ для делопроизводителя

AdminControlsVisibility(false);

ClerkControlsVisibility(true);

ArchivariusControlsVisibility(false);

break;

case "Архивариус": // Ограниченный доступ для архивариуса

AdminControlsVisibility(false);

ClerkControlsVisibility(true);

ArchivariusControlsVisibility(true);

break;

default: // Обработка неизвестной роли

MessageBox.Show("Неизвестная роль!");

break;

}

}

private void AdminControlsVisibility(bool isVisible)

{

Visibility visibility = isVisible ? Visibility.Visible : Visibility.Collapsed; // Установка видимости

ReportBtn.Visibility = visibility; // Кнопка отчета

UserBtn.Visibility = visibility; // Кнопка пользователей

RegCardBtn.Visibility = visibility; // Кнопка карточек

RequestBtn.Visibility = visibility; // Кнопка запросов

}

private void ClerkControlsVisibility(bool isVisible)

{

Visibility visibility = isVisible ? Visibility.Visible : Visibility.Collapsed; // Установка видимости

RegCardBtn.Visibility = visibility; // Кнопка карточек

DocumentBtn.Visibility = visibility;// Кнопка документов

SimpleRepBtn.Visibility = visibility;// Кнопка простого отчета

}

private void ArchivariusControlsVisibility(bool isVisible)

{

Visibility visibility = isVisible ? Visibility.Visible : Visibility.Collapsed; // Установка видимости

RequestBtn.Visibility = visibility; // Кнопка запросов

}

В данном модуле было разработано главное меню с разграничением прав пользователей и кнопками навигации по страницам, если роль пользователя «Администратор», то доступны все возможности, если роль пользователя «Архивариус» или «Делопроизводитель», то только часть функций.

* + 1. Разработка модуля «Документы»

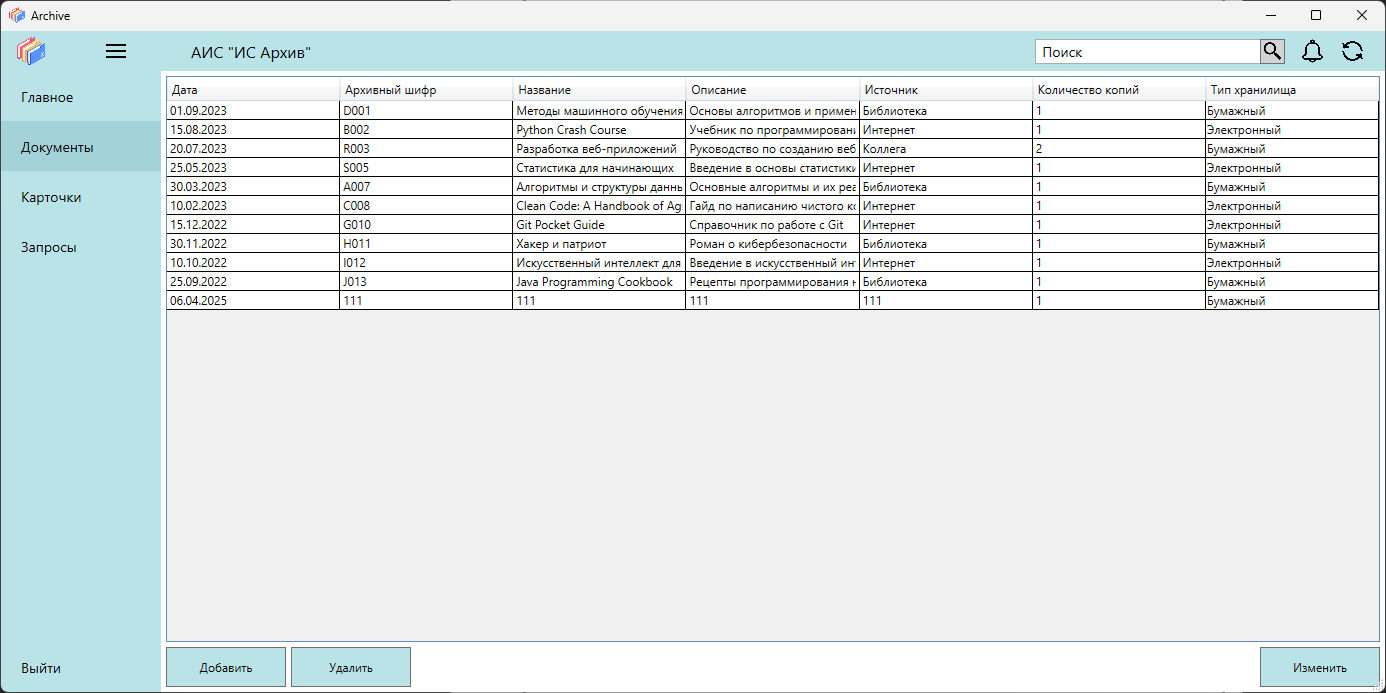


Рисунок 3.3 Окно «Документы»

Методы создания таблицы и загрузки данных в эту таблицу с возможностью изменения, удаления и добавления новых данных:

private void LoadData()

{

using (var context = new ArchiveBaseEntities()) // Подключение к базе данных

{

DataGridTable.ItemsSource = context.Document.ToList(); // Загрузка документов в DataGrid

}

DataGridTable.IsReadOnly = true; // Установка режима "только чтение"

}

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

{

var documentsForRemoving = DataGridTable.SelectedItems.Cast<Document>().ToList(); // Получение выбранных документов

if (documentsForRemoving.Count == 0) // Проверка наличия выбранных элементов

{

MessageBox.Show("Выберите хотя бы один элемент для удаления!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (MessageBox.Show($"Вы точно хотите удалить {documentsForRemoving.Count} элементов?", // Подтверждение удаления

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

{

try

{

using (var context = new ArchiveBaseEntities()) // Подключение к базе данных

{

foreach (var doc in documentsForRemoving) // Удаление каждого документа

{

var docToRemove = context.Document.Find(doc.Id); // Поиск документа в базе

if (docToRemove != null)

{

context.Registration\_Card.RemoveRange(docToRemove.Registration\_Card); // Удаление связанных карточек

context.Request.RemoveRange(docToRemove.Request); // Удаление связанных запросов

context.Document.Remove(docToRemove); // Удаление документа

}

}

context.SaveChanges(); // Сохранение изменений

}

MessageBox.Show("Данные удалены!"); // Уведомление об успешном удалении

LoadData(); // Перезагрузка данных

}

catch (Exception ex) // Обработка ошибок

{

MessageBox.Show($"Ошибка при удалении: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

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

{

if (DataGridTable.IsReadOnly) // Переключение в режим редактирования

{

DataGridTable.IsReadOnly = false; // Разрешение редактирования

EditBtn.Content = "Сохранить"; // Изменение текста кнопки

}

else // Сохранение изменений

{

DataGridTable.IsReadOnly = true; // Блокировка редактирования

EditBtn.Content = "Изменить"; // Восстановление текста кнопки

SaveChanges(); // Сохранение изменений

}

}

private void SaveChanges()

{

using (var context = new ArchiveBaseEntities()) // Подключение к базе данных

{

if (newDocument != null && isAddingNewRow) // Добавление нового документа

{

if (string.IsNullOrWhiteSpace(newDocument.Title) || // Проверка обязательных полей

string.IsNullOrWhiteSpace(newDocument.Number) ||

string.IsNullOrWhiteSpace(newDocument.Source) ||

string.IsNullOrWhiteSpace(newDocument.Storage\_Type))

{

RemoveEmptyRow(); // Удаление пустой строки при ошибке

return;

}

context.Document.Add(newDocument); // Добавление нового документа в базу

}

foreach (var item in DataGridTable.Items) // Обновление существующих документов

{

if (item is Document doc && doc != newDocument) // Проверка типа и исключение нового документа

{

var docToUpdate = context.Document.Find(doc.Id); // Поиск документа в базе

if (docToUpdate != null) // Обновление полей документа

{

docToUpdate.Number = doc.Number;

docToUpdate.Receipt\_Date = doc.Receipt\_Date;

docToUpdate.Title = doc.Title;

docToUpdate.Annotation = doc.Annotation;

docToUpdate.Source = doc.Source;

docToUpdate.Copies\_Count = doc.Copies\_Count;

docToUpdate.Storage\_Type = doc.Storage\_Type;

}

}

}

context.SaveChanges(); // Сохранение изменений в базе

}

isAddingNewRow = false; // Сброс флага добавления

newDocument = null; // Очистка нового документа

LoadData(); // Перезагрузка данных

}

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

{

if (isAddingNewRow) return; // Защита от повторного добавления

isAddingNewRow = true; // Установка флага добавления

newDocument = new Document // Создание нового документа

{

Receipt\_Date = DateTime.Now, // Текущая дата

Number = "", // Пустой номер

Title = "", // Пустое название

Source = "", // Пустой источник

Copies\_Count = 0, // Количество копий по умолчанию

Annotation = "", // Пустая аннотация

Storage\_Type = StorageTypes.FirstOrDefault() // Первый тип хранения

};

var items = DataGridTable.ItemsSource as List<Document>; // Получение текущего списка

if (items != null) // Добавление нового документа в список

{

items.Add(newDocument);

DataGridTable.ItemsSource = null; // Сброс источника данных

DataGridTable.ItemsSource = items; // Переустановка источника данных

}

DataGridTable.SelectedItem = newDocument; // Установка фокуса на новую строку

foreach (var item in DataGridTable.Items) // Блокировка других строк

{

if (item is Document doc && doc != newDocument)

{

var row = DataGridTable.ItemContainerGenerator.ContainerFromItem(doc) as DataGridRow;

if (row != null) row.IsEnabled = false; // Отключение строки

}

}

DataGridTable.IsReadOnly = false; // Разрешение редактирования

EditBtn.Content = "Сохранить"; // Изменение текста кнопки

}

В данном модуле была разработана таблица документов, которая берет данные из базы данных и имеется возможность их изменить, удалить или добавить.

* + 1. Разработка модуля «Карточки»

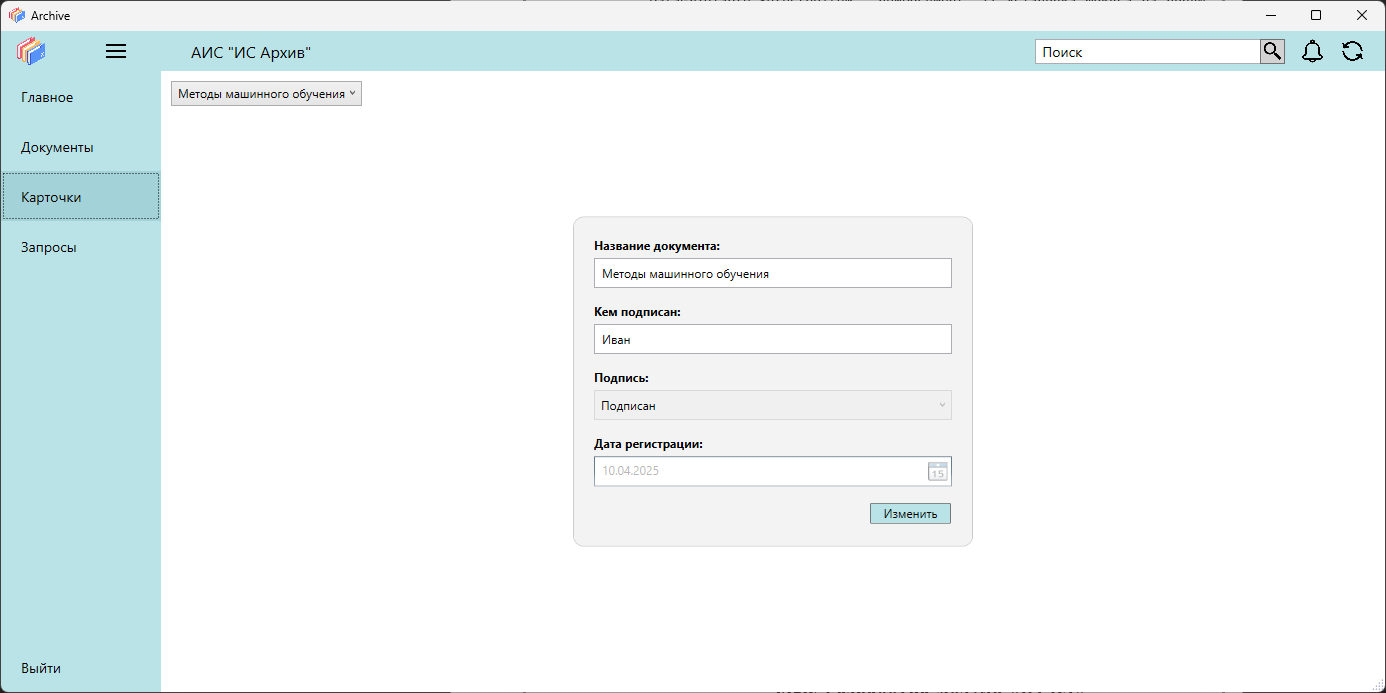


Рисунок 3.2.4 Окно «Карточки»

Методы разграничения прав для кнопки «Изменить»:

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

{

if (!isEditMode) // Переключение в режим редактирования

{

if (currentUserRole != "Администратор") // Проверка прав для не-администраторов

{

if (selectedRegCard != null && selectedRegCard.Signature == true) // Блокировка редактирования подписанных документов

{

MessageBox.Show("Документ уже подписан и не может быть изменен");

return;

}

}

isEditMode = true; // Включение режима редактирования

EditBtn.Content = "Сохранить"; // Изменение текста кнопки

TitleTextBox.IsReadOnly = false; // Разрешение редактирования названия

SignatureСomboBox.IsEnabled = true; // Разрешение выбора статуса

RegistrationDatePicker.IsEnabled = true; // Разрешение выбора даты

if (currentUserRole != "Администратор") // Установка имени для не-администраторов

{

SignedByTextBox.Text = Users.FirstOrDefault(u => u.Id == currentUserId)?.Name ?? "Неизвестно";

}

if (selectedRegCard == null || selectedRegCard.Signature == false || currentUserRole == "Администратор") // Установка текущей даты

{

RegistrationDatePicker.SelectedDate = DateTime.Now;

}

}

else // Сохранение изменений

{

if (string.IsNullOrWhiteSpace(TitleTextBox.Text) || // Проверка заполнения полей

SignatureСomboBox.SelectedIndex == -1 ||

!RegistrationDatePicker.SelectedDate.HasValue)

{

MessageBox.Show("Поля не должны быть пустыми. Изменения отменены.");

return;

}

using (var context = new ArchiveBaseEntities()) // Подключение к базе данных

{

if (DocumentComboBox.SelectedItem is Document selectedDoc) // Обработка выбранного документа

{

var doc = context.Document.Find(selectedDoc.Id); // Поиск документа в базе

if (doc != null)

doc.Title = TitleTextBox.Text; // Обновление названия документа

var regCard = context.Registration\_Card.FirstOrDefault(rc => rc.Document\_Id == selectedDoc.Id); // Поиск карточки

if (regCard != null) // Обновление существующей карточки

{

regCard.Signature = (bool)SignatureСomboBox.SelectedValue; // Обновление статуса подписи

regCard.Registration\_Date = RegistrationDatePicker.SelectedDate.Value; // Обновление даты

if (currentUserRole != "Администратор") // Обновление ID пользователя для не-администраторов

regCard.User\_Id = currentUserId;

}

else // Создание новой карточки

{

regCard = new Registration\_Card

{

Document\_Id = selectedDoc.Id,

User\_Id = currentUserId,

Signature = (bool)SignatureСomboBox.SelectedValue,

Registration\_Date = RegistrationDatePicker.SelectedDate.Value

};

context.Registration\_Card.Add(regCard); // Добавление карточки в базу

}

context.SaveChanges(); // Сохранение изменений в базе

}

}

MessageBox.Show("Изменения сохранены."); // Уведомление о сохранении

isEditMode = false; // Выход из режима редактирования

EditBtn.Content = "Изменить"; // Восстановление текста кнопки

TitleTextBox.IsReadOnly = true; // Блокировка редактирования названия

SignatureСomboBox.IsEnabled = false;// Блокировка выбора статуса

RegistrationDatePicker.IsEnabled = false; // Блокировка выбора даты

LoadDocuments(); // Перезагрузка документов

LoadRegistrationCards(); // Перезагрузка карточек

DocumentComboBox\_SelectionChanged(DocumentComboBox, null); // Обновление UI

}

}

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

{

if (DocumentComboBox.SelectedItem is Document selectedDocument) // Обработка выбранного документа

{

TitleTextBox.Text = selectedDocument.Title; // Отображение названия документа

selectedRegCard = RegCards.FirstOrDefault(rc => rc.Document\_Id == selectedDocument.Id); // Поиск карточки

if (selectedRegCard != null) // Если карточка существует

{

var user = Users.FirstOrDefault(u => u.Id == selectedRegCard.User\_Id); // Поиск пользователя карточки

SignedByTextBox.Text = user?.Name ?? "Неизвестно"; // Отображение имени подписавшего

SignatureСomboBox.SelectedValue = selectedRegCard.Signature; // Установка статуса подписи

RegistrationDatePicker.SelectedDate = selectedRegCard.Registration\_Date; // Установка даты

bool isSigned = selectedRegCard.Signature == true; // Проверка статуса подписи

if (currentUserRole == "Администратор") // Настройка для администратора

{

EditBtn.IsEnabled = true; // Всегда доступно редактирование

EditBtn.Content = "Изменить"; // Текст кнопки

}

else // Настройка для не-администраторов

{

EditBtn.IsEnabled = !isSigned; // Доступно только для неподписанных

EditBtn.Content = isSigned ? "Подписано" : "Изменить"; // Текст кнопки

}

TitleTextBox.IsReadOnly = true; // Блокировка редактирования названия

SignatureСomboBox.IsEnabled = false;// Блокировка выбора статуса

RegistrationDatePicker.IsEnabled = false; // Блокировка выбора даты

}

else // Если карточка не найдена

{

SignedByTextBox.Text = ""; // Очистка имени

SignatureСomboBox.SelectedIndex = -1; // Сброс статуса

RegistrationDatePicker.SelectedDate = null; // Сброс даты

EditBtn.IsEnabled = true; // Разрешение создания новой карточки

EditBtn.Content = "Изменить"; // Текст кнопки

TitleTextBox.IsReadOnly = true; // Блокировка редактирования названия

SignatureСomboBox.IsEnabled = false; // Блокировка выбора статуса

RegistrationDatePicker.IsEnabled = false; // Блокировка выбора даты

}

}

}

В данном модуле была разработана страница карточек документов с разграничением прав и возможностью их изменения.

* + 1. Разработка модуля «Запросы»

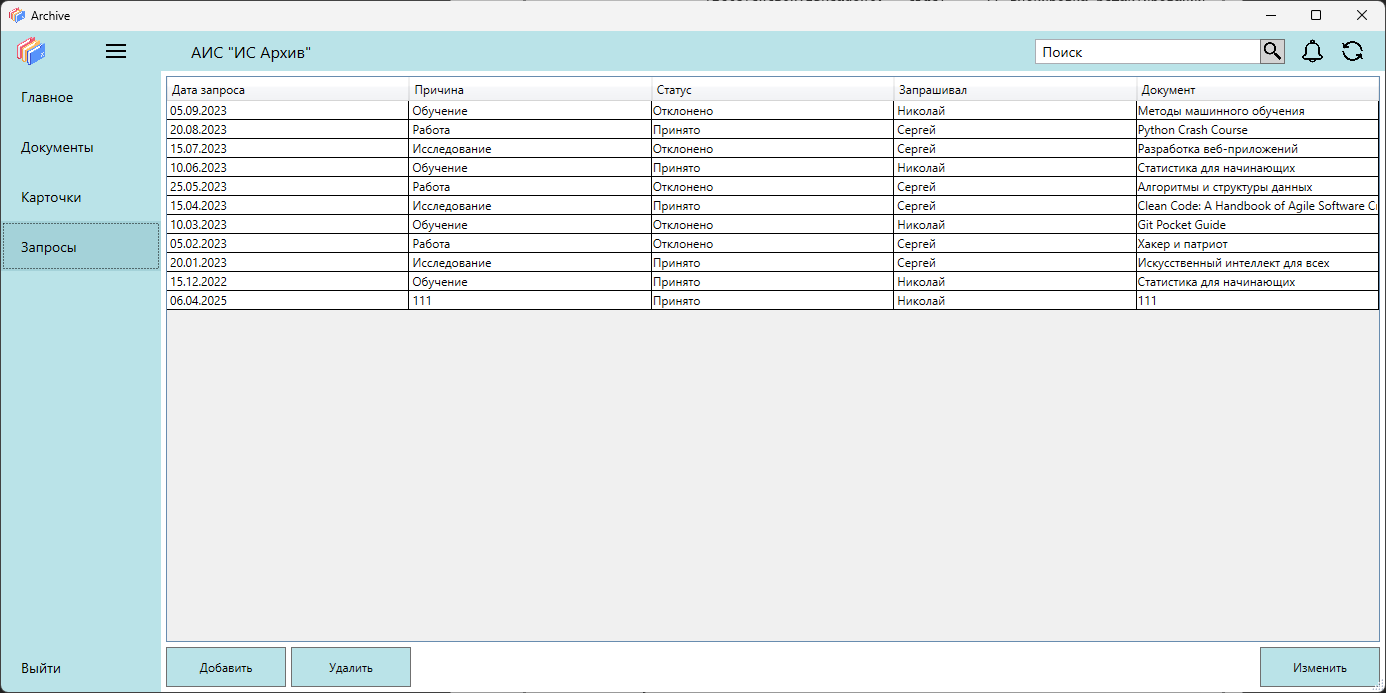


Рисунок 3.2.5 Окно «Запросы»

Методы создания таблицы и удаления, изменения и добавления данных в эту таблицу:

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

{

var selectedRequests = DataGridTable.SelectedItems.Cast<Request>().ToList(); // Получение выбранных запросов

if (selectedRequests.Count == 0) // Проверка наличия выбранных элементов

{

MessageBox.Show("Выберите хотя бы один элемент для удаления!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (MessageBox.Show($"Вы точно хотите удалить {selectedRequests.Count} элементов?", // Подтверждение удаления

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

{

try

{

using (var context = new ArchiveBaseEntities()) // Подключение к базе данных

{

foreach (var req in selectedRequests) // Удаление каждого запроса

{

var reqToRemove = context.Request.Find(req.Id); // Поиск запроса в базе

if (reqToRemove != null)

context.Request.Remove(reqToRemove); // Удаление запроса

}

context.SaveChanges(); // Сохранение изменений

}

MessageBox.Show("Данные удалены!"); // Уведомление об успешном удалении

LoadData(); // Перезагрузка данных

}

catch (Exception ex) // Обработка ошибок

{

MessageBox.Show($"Ошибка при удалении: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

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

{

if (DataGridTable.IsReadOnly) // Переключение в режим редактирования

{

DataGridTable.IsReadOnly = false; // Разрешение редактирования

EditBtn.Content = "Сохранить"; // Изменение текста кнопки

}

else // Сохранение изменений

{

DataGridTable.IsReadOnly = true; // Блокировка редактирования

EditBtn.Content = "Изменить"; // Восстановление текста кнопки

SaveChanges(); // Сохранение изменений

}

}

private void SaveChanges()

{

try

{

using (var context = new ArchiveBaseEntities()) // Подключение к базе данных

{

if (isAddingNewRow && newRequest != null) // Добавление нового запроса

{

if (string.IsNullOrWhiteSpace(newRequest.Reason) || newRequest.Document\_Id == 0) // Проверка обязательных полей

{

RemoveEmptyRow(); // Удаление пустой строки при ошибке

return;

}

var requestToAdd = new Request // Создание нового запроса

{

Request\_Date = newRequest.Request\_Date,

Reason = newRequest.Reason,

Status = newRequest.Status,

User\_Id = newRequest.User\_Id,

Document\_Id = newRequest.Document\_Id

};

context.Request.Add(requestToAdd); // Добавление запроса в базу

context.SaveChanges(); // Сохранение изменений

}

foreach (var req in Requests.Where(r => r.Id != 0)) // Обновление существующих запросов

{

var reqToUpdate = context.Request.Find(req.Id); // Поиск запроса в базе

if (reqToUpdate != null) // Обновление полей

{

reqToUpdate.Request\_Date = req.Request\_Date;

reqToUpdate.Reason = req.Reason;

reqToUpdate.Status = req.Status;

reqToUpdate.Document\_Id = req.Document\_Id;

}

}

context.SaveChanges(); // Сохранение всех изменений

}

isAddingNewRow = false; // Сброс флага добавления

newRequest = null; // Очистка нового запроса

LoadData(); // Перезагрузка данных

}

catch (Exception ex) // Обработка ошибок

{

MessageBox.Show($"Ошибка при сохранении: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

if (isAddingNewRow) return; // Защита от повторного добавления

isAddingNewRow = true; // Установка флага добавления

var currentUser = Users.FirstOrDefault(u => u.Id == currentUserId); // Поиск текущего пользователя

newRequest = new Request // Создание нового запроса

{

Id = 0, // ID=0 для новой записи

Request\_Date = DateTime.Now, // Текущая дата

Reason = "", // Пустое основание

Status = null, // Статус не установлен

User\_Id = currentUserId, // ID текущего пользователя

Document\_Id = 0, // Документ не выбран

Document = null, // Связанный документ не установлен

User = currentUser // Данные текущего пользователя

};

Requests.Add(newRequest); // Добавление в коллекцию

DataGridTable.SelectedItem = newRequest; // Установка фокуса на новую строку

DataGridTable.ScrollIntoView(newRequest); // Прокрутка к новой строке

DataGridTable.IsReadOnly = false; // Разрешение редактирования

EditBtn.Content = "Сохранить"; // Изменение текста кнопки

}

В данном модуле была разработана таблица запросов с возможностью изменения, добавления и удаления данных из базы данных.

* + 1. Разработка модуля «Отчет»

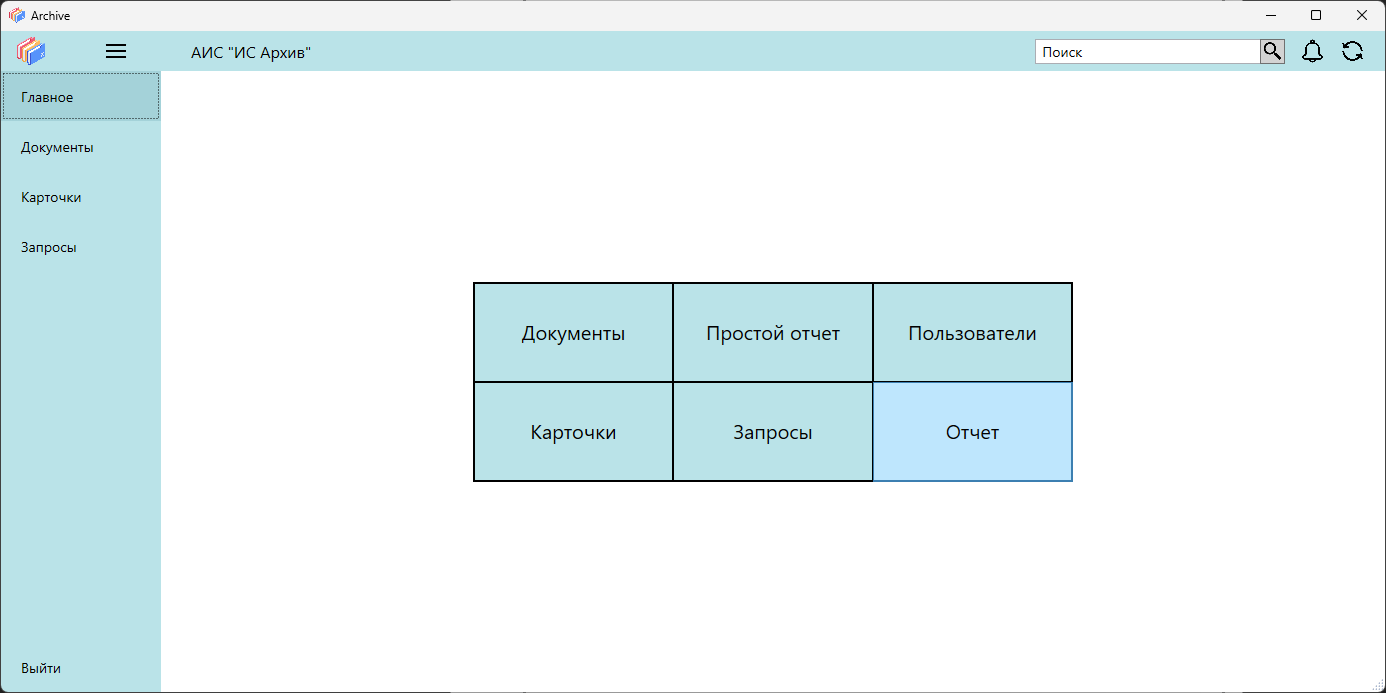


Рисунок 3.2.6 Кнопка «Отчет»

Класс экспорта данных из базы данных в виде таблиц в Word документ:

public static void ExportToWord(string filePath)

{

try

{

using (var context = new ArchiveBaseEntities()) // Подключение к базе данных

{

var data = new // Загрузка всех данных из базы

{

Documents = context.Document.ToList(),

Requests = context.Request.Include("User").Include("Document").ToList(),

Users = context.User.Include("Role").ToList(),

RegistrationCards = context.Registration\_Card.Include("User").Include("Document").ToList()

};

if (File.Exists(filePath)) File.Delete(filePath); // Удаление существующего файла

Word.Application wordApp = new Word.Application(); // Создание приложения Word

wordApp.Visible = false; // Скрытие интерфейса Word

Word.Document doc = wordApp.Documents.Add(); // Создание нового документа

SetDocumentStyles(doc); // Настройка стилей документа

AddTitle(doc, "Полный отчет архива документов"); // Добавление заголовка

ExportDocumentsToWord(doc, data.Documents); // Экспорт документов

if (data.Requests.Any() || data.Users.Any() || data.RegistrationCards.Any()) // Условный разрыв страницы

AddPageBreak(doc);

if (data.Requests.Any()) // Экспорт запросов

{

ExportRequestsToWord(doc, data.Requests);

if (data.Users.Any() || data.RegistrationCards.Any()) AddPageBreak(doc);

}

if (data.Users.Any()) // Экспорт пользователей

{

ExportUsersToWord(doc, data.Users);

if (data.RegistrationCards.Any()) AddPageBreak(doc);

}

if (data.RegistrationCards.Any()) // Экспорт регистрационных карточек

ExportRegistrationCardsToWord(doc, data.RegistrationCards);

doc.SaveAs2(filePath); // Сохранение документа

doc.Close(); // Закрытие документа

wordApp.Quit(); // Закрытие приложения Word

ReleaseWordObjects(doc, wordApp); // Освобождение ресурсов

OpenExportedFile(filePath); // Открытие файла

}

}

catch (Exception ex) // Обработка ошибок

{

MessageBox.Show($"Ошибка при экспорте в Word: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private static void SetDocumentStyles(Word.Document doc)

{

doc.Content.Font.Name = "Times New Roman"; // Установка шрифта

doc.Content.Font.Size = 14; // Установка размера шрифта

doc.Content.ParagraphFormat.LineSpacing = 18f; // Межстрочный интервал

doc.Content.ParagraphFormat.SpaceBefore = 0; // Отступ перед абзацем

doc.Content.ParagraphFormat.SpaceAfter = 0; // Отступ после абзаца

doc.Content.ParagraphFormat.Alignment = Word.WdParagraphAlignment.wdAlignParagraphCenter; // Выравнивание по центру

}

private static void AddTitle(Word.Document doc, string text)

{

Word.Paragraph title = doc.Paragraphs.Add(); // Добавление абзаца для заголовка

title.Range.Text = text; // Установка текста заголовка

title.Range.Font.Bold = 1; // Жирный шрифт

title.Range.Font.Size = 16; // Размер шрифта заголовка

title.Format.SpaceBefore = 0; // Отступ перед заголовком

title.Format.SpaceAfter = 0; // Отступ после заголовка

title.Format.Alignment = Word.WdParagraphAlignment.wdAlignParagraphCenter; // Выравнивание по центру

title.Range.InsertParagraphAfter(); // Добавление пустого абзаца после

}

private static void ExportDocumentsToWord(Word.Document doc, List<Document> documents)

{

AddTableTitle(doc, "Документы"); // Добавление заголовка таблицы

Word.Table table = CreateWordTable(doc, new string[] { "ID", "Номер", "Название", "Источник", "Копии", "Тип хранения" }); // Создание таблицы

foreach (var item in documents) // Заполнение таблицы данными документов

{

AddRowToWordTable(table, new string[] {

item.Id.ToString(),

item.Number,

item.Title,

item.Source,

item.Copies\_Count.ToString(),

item.Storage\_Type

});

}

FinalizeWordTable(table); // Финальное форматирование таблицы

}

private static Word.Table CreateWordTable(Word.Document doc, string[] headers)

{

Word.Table table = doc.Tables.Add(doc.Range(doc.Content.End - 1), 1, headers.Length); // Создание таблицы с одной строкой

for (int i = 0; i < headers.Length; i++) // Заполнение заголовков

{

table.Cell(1, i + 1).Range.Text = headers[i]; // Установка текста заголовка

table.Cell(1, i + 1).Range.Font.Bold = 1; // Жирный шрифт для заголовков

table.Cell(1, i + 1).Range.ParagraphFormat.Alignment = Word.WdParagraphAlignment.wdAlignParagraphCenter; // Выравнивание по центру

}

return table; // Возврат созданной таблицы

}

private static void AddRowToWordTable(Word.Table table, string[] values)

{

table.Rows.Add(); // Добавление новой строки

int rowIndex = table.Rows.Count; // Индекс новой строки

for (int i = 0; i < values.Length; i++) // Заполнение ячеек строки

{

table.Cell(rowIndex, i + 1).Range.Text = values[i] ?? ""; // Установка текста ячейки

table.Cell(rowIndex, i + 1).Range.Font.Bold = 0; // Обычный шрифт

table.Cell(rowIndex, i + 1).Range.ParagraphFormat.Alignment = Word.WdParagraphAlignment.wdAlignParagraphCenter; // Выравнивание по центру

}

}

В данном модуле был разработан экспорт таблиц базы данных в **Word**.

# ЗАКЛЮЧЕНИЕ

В рамках выполнения курсового проекта была разработана автоматизированная информационная система "Архив" для управления документами в архивных учреждениях. При создании системы использовались современные технологии: платформа .NET, язык программирования C#, WPF для интерфейса, SQL Server в качестве СУБД и Entity Framework для работы с данными.

Система включает модули авторизации с проверкой логина, пароля и капчи, главное меню с адаптацией под роли пользователей (администратор, архивариус, делопроизводитель), функционал для работы с документами и запросами, а также формирование отчетов с возможностью экспорта в Word. База данных построена по реляционной модели и содержит таблицы для пользователей, документов, регистрационных карточек и запросов, что отражено в ER-диаграмме. Система полностью соответствует требованиям технического задания, включая разграничение прав доступа, контроль вводимых данных и защиту от несанкционированного доступа.

Внедрение системы позволит сократить время на регистрацию документов с 8-10 минут до 1-2 минут, а формирование отчетов - с 6-12 часов до 3-4 минут, что значительно повысит эффективность работы архива. Для дальнейшего развития системы рекомендуется интеграция с MS Office и расширение функционала, например, добавление мобильного или веб-интерфейса. Разработанная информационная система успешно реализована, соответствует поставленным задачам и готова к внедрению, обеспечивая автоматизацию ключевых процессов архивного учета.

Все этапы разработки - от выбора инструментария до программирования модулей - выполнены в полном объеме, что подтверждает достижение целей курсового проекта.

[Репозиторий GitHub](https://github.com/5sway/Kursach)

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Введение в Entity Framework Core [Электронный ресурс] — режим доступа: <https://metanit.com/sharp/efcore/1.1.php>
2. Введение в MS SQL Server и T-SQL [Электронный ресурс] — режим доступа: <https://metanit.com/sql/sqlserver/1.1.php>
3. Документация по C# [Электронный ресурс] — режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/overview>
4. Документация по Microsoft SQL [Электронный ресурс] — режим доступа: <https://learn.microsoft.com/ru-ru/ssms/sql-server-management-studio-ssms>
5. Полное руководство по языку программирования С# 13 и платформе .NET 9. [Электронный ресурс] — режим доступа: <https://metanit.com/sharp/tutorial/1.1.php>
6. Руководство по WPF [Электронный ресурс] — режим доступа: <https://metanit.com/sharp/wpf/1.php>
7. Что такое ER-диаграмма и как ее создать? [Электронный ресурс] — режим доступа: <https://www.lucidchart.com/pages/ru/erd-diagram>