Министерство цифрового развития государственного управления информационных технологий и связи Республики Татарстан

государственное автономное профессиональное образовательное учреждение

«Международный центр компетенций –

Казанский техникум информационных технологий и связи»

|  |  |
| --- | --- |
| ВКР выполнена и защищена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Председатель ГЭК\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (ФИО)  «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_\_ г | ДОПУСТИТЬ К ЗАЩИТЕ  Заместитель директора по учебной работе  \_\_\_\_\_\_\_\_\_\_\_\_ О. С. Тимофеева |

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

(ДИПЛОМНЫЙ ПРОЕКТ)

Разработка информационной системы МБДОУ Балтасинский детский сад №3 общеразвивающего вида

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Исполнитель ВКР,  студент группы 421 ПКС | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *подпись, дата* | Р.Н. Шакиров |

|  |  |  |
| --- | --- | --- |
| Руководитель ВКР,  преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *подпись, дата* | Ф.К. Сулимов |

Казань 2020 г.

Министерство цифрового развития государственного управления информационных технологий и связи Республики Татарстан

государственное автономное профессиональное образовательное учреждение

«Международный центр компетенций –

Казанский техникум информационных технологий и связи»

УТВЕРЖДАЮ

Заместитель директора

по учебной работе

\_\_\_\_\_\_\_\_\_\_\_\_ О. С. Тимофеева

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

ЗАДАНИЕ

на выпускную квалификационную работу

(дипломная проект)

студенту Шакирову Рузалю Нурмахаматовичу группы 421 ПКС  
по специальности 09.02.03 «Программирование в компьютерных системах»

1. Тема ВКР: Разработка информационной системы МБДОУ Балтасинский детский сад №3 общеразвивающего вида, утверждена приказом директора от «\_\_» \_\_\_\_\_\_\_\_\_ 2020 г. № \_\_\_\_\_\_\_
2. Срок сдачи студентом законченной ВКР: «8» июня 2020 г.
3. Исходные данные к ВКР: техническое задание, описание предметной области.
4. Краткое содержание ВКР или перечень подлежащих разработке вопросов: введение, анализ предметной области, диаграммы, обоснование выбора языка программирования, компоненты программы, описание пользовательского интерфейса, создание проекта, тестирование, заключение, источники информации.
5. Перечень графических материалов/Приложения: техническое задание, код программы, список файлов на диске.

План-график выполнения ВКР (дипломного проекта):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Наименование этапов ВКР | Срок выполнения этапов ВКР | | Подпись руководителя,  консультантов ВКР |
| План | Факт |
| 1 | Получение задания | 06.04.2020 | 06.04.2020 |  |
| 2 | Проектирование информационной системы | 10.04.2020-18.04.2020 | 10.04.2020-18.04.2020 |  |
| 3 | Разработка программного продукта | 20.04.2020-26.05.2020 | 20.04.2020-26.05.2020 |  |
| 4 | Работа над разделом «Введение» | 10.04.2020-15.04.2020 | 10.04.2020-15.04.2020 |  |
| 5 | Работа над разделом Проектирование» | 16.04.2020-20.04.2020 | 16.04.2020-20.04.2020 |  |
| 6 | Работа над разделом «Реализация проекта» | 20.04.2020-15.05.2020 | 20.04.2020-15.05.2020 |  |
| 7 | Работа над разделом  «Тестирование программного продукта» | 15.05.2020-20.05.2020 | 15.05.2020-20.05.2020 |  |
| 8 | Работа над разделом «Заключение» | 20.05.2020- 26.05.2020 | 20.05.2020- 26.05.2020 |  |
| 9 | Предварительная защита |  |  |  |
| 10 | Защита дипломного проекта |  |  |  |

Рассмотрено на заседании ЦК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Протокол № \_\_\_\_ от «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

Председатель ЦК \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*подпись инициалы и фамилия*

Дата выдачи задания:  
 «10» апреля 2020 г.

Руководитель ВКР:

Преподаватель ГАПОУ «МЦК-КТИТС» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ф.К. Сулимов

*подпись*

Задание принял к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Р.Н. Шакиров

*подпись*

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc42470536)

[1 РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА 6](#_Toc42470537)

[1.1 Цель и задачи дипломного проекта 6](#_Toc42470538)

[1.2 Анализ предметной области 7](#_Toc42470539)

[1.3 ER-Диаграмма 8](#_Toc42470540)

[1.4 Диаграмма вариантов использования 9](#_Toc42470541)

[1.5 Диаграмма потоков данных 11](#_Toc42470542)

[2 РЕАЛИЗАЦИЯ ПРОЕКТА 12](#_Toc42470543)

[2.1 Обоснование выбора СУБД 12](#_Toc42470544)

[2.2 Обоснование выбора языка и среды программирования 13](#_Toc42470545)

[2.3 Компоненты, используемые в программном проекте 14](#_Toc42470546)

[2.4 Описание пользовательского интерфейса 15](#_Toc42470547)

[2.5 Создание проекта 20](#_Toc42470548)

[3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА 28](#_Toc42470549)

[3.1 Основные понятия и принципы тестирования ПП 28](#_Toc42470550)

[3.2 Тестирование методом черного ящика 29](#_Toc42470551)

[4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 32](#_Toc42470552)

[4.1 Запуск программы 32](#_Toc42470553)

[4.2 Работа с программой. Заведующий 33](#_Toc42470554)

[4.3 Работа с программой. Старший воспитатель 37](#_Toc42470555)

[ЗАКЛЮЧЕНИЕ 40](#_Toc42470556)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 41](#_Toc42470557)

[ПРИЛОЖЕНИЕ А 42](#_Toc42470558)

[ПРИЛОЖЕНИЕ Б 46](#_Toc42470559)

[ПРИЛОЖЕНИЕ В 78](#_Toc42470560)

# ВВЕДЕНИЕ

Целью работы является проектирование базы данных и разработка информационной системы для использования в детском саду муниципального учреждения.

Основным направлением работы руководства детского сада является организация и ведение образовательного процесса.

**Каждое занятие, прогулку, дополнительную работу нужно поминутно расписывать вручную.** Во время сончаса воспитатели постоянно ведут занятия: вырезают, клеют пишут конспекты и планы, пополняют картотеку игр, подбирают новые упражнения для гимнастики, неустанно ищут

в интернете детские картинки и печатают их, чтобы потом торжественно продемонстрировать малышам.

В любое время дня педагога могут проверить — посмотреть, как он проводит занятие, соблюдает ли режим, ведет ли документы, все ли оборудование у него работает.

В таких сложных и напряженных для обслуживающего персонала условиях становится актуальным вопрос создания информационной системы для детского сада с применением современных методов ведения документооборота с введением базы данных.

Основным путем достижения поставленных задач является автоматизация деятельности старшего воспитателя, путем введения элек­тронного документооборота, автоматического составления сетки занятий, автоматический подсчет оплаты платных (дополнительных) услуг по каждому ребенку. Это позволит упростить работу методиста и предоставит дополнительное время для выполнения других обязанностей.

# 1 РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА

# 1.1 Цель и задачи дипломного проекта

Цель выпускной квалификационной работы (дипломного проекта)- повышение производительности персонала и сокращение времени на ведение отчетности.

Требуется разработать программный продукт и выполнить проектирование базы данных. С информационной системой должны работать следующие базовые группы пользователей:

* заведующий детским садом;
* старший воспитатель;
* воспитатель;
* педагог дополнительного образования.

Подсистема для заведующего детским садом должна иметь возможность решать следующие задачи:

* добавлять, редактировать, удалять данные о педагогов, родителях, детях, ролях, помещениях, статусах посещении;
* настраивать шкалу времени расписания;
* просматривать доходы и расходы;
* составлять отчеты;
* все возможности старшего воспитателя, воспитателей и педагогов.

Подсистема для старшего воспитателя должна иметь возможность решать следующие задачи:

* заполнять сетку расписании;
* добавления, удаления шаблонов абонементов;
* просматривать профили сотрудников;
* все возможности воспитателей и педагогов.

Воспитатели и педагоги имеют возможность вести учет посещении, просматривать профили способностей детей, участие родителей, а также просматривать расписание

# 1.2 Анализ предметной области

В проекте рассматривается предметная область детского сада.

Детский сад — учреждение для общественного воспитания детей дошкольного возраста. Детские сады как тип учреждений существуют в большинстве стран и являются обычно первым звеном в системе народного образования (не считая образования, полученного от родителей). Также в детском саду детей дают основу культурного поведения. Кроме подготовки к школе там учат доброте и милосердию, рассказывают о моральных нормах и культурных ценностях.

Самая главная проблема воспитателей и педагогов в детских садах - это бумажная работа. Зачастую им некогда заниматься детьми. Нужно постоянно писать отчеты, проводить мониторинги в начале года и в конце, разрабатывать конспекты, придумывать мероприятия — причем, иногда это случается только на бумаге, а с детьми некогда взаправду проводить, и они занимаются непонятно чем.

Перед началом каждого учебного года старшему воспитателю передается список детей по возрастам, а также учебно-методический план на год от заведующей детским садом. На основе данной информации методист формирует группы, составляет сетку занятий и передает эту информацию воспитателям и педагогам.

Воспитатели проводят опрос среди родителей каждого ребенка об интересующих их дополнительных занятиях, предоставляемых данным учреждением, и составляют списки детей. Эту информацию они передают старшему воспитателю, который в свою очередь, основываясь на полученной информации, заполняет «Абонементы» каждого ребенка.

Каждый воспитатель ежедневно отмечает присутствие или отсутствие детей (во втором случае отмечаются причины).

В конце каждого месяца заведующая ДОУ формирует отчет о посещаемости детей на основе информации, полученных от воспитателей.

Должностным лицом, принимающим решения, является заведующая ДОУ. Исполнителем являются воспитатели и педагоги.

# 1.3 ER-Диаграмма

ER-диаграмма [1] — это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы (Рисунок 1.1). 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.

ER-диаграммы применяются для моделирования и проектирования реляционных баз данных, причем как в плане логических и бизнес-правил (логические модели данных), так и в плане внедрения конкретных технологий (физические модели данных). В сфере разработки программного обеспечения ER-диаграмма, как правило, служит первым шагом в определении требований проекта по созданию информационных систем. На дальнейших этапах работы ER-диаграммы также применяются для моделирования конкретных баз данных. Реляционная база данных сопровождается соответствующей реляционной таблицей и при необходимости может быть представлена в этом формате.

ER-диаграммы часто используются в сочетании с диаграммами DFD, которые схематично показывают движение потоков информации в рамках процесса или системы.

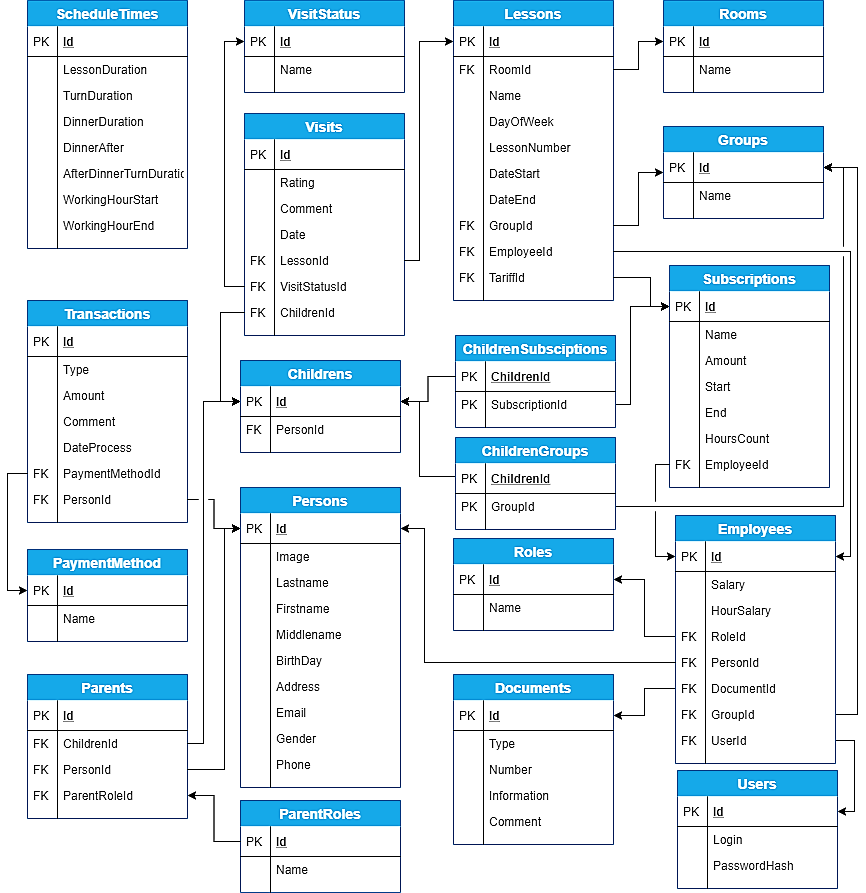


Рисунок 1.1 – ER диаграмма информационной системы

# 1.4 Диаграмма вариантов использования

Визуальное моделирование в UML[2] (Рисунок 1.2) можно представить как некоторый процесс поуровневого спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования, которая описывает функциональное назначение системы или то, что система будет делать в процессе своего функционирования.

Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Разработка диаграммы вариантов использования преследует цели:

* определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
* сформулировать общие требования к функциональному поведению проектируемой системы;
* разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне.

Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик.

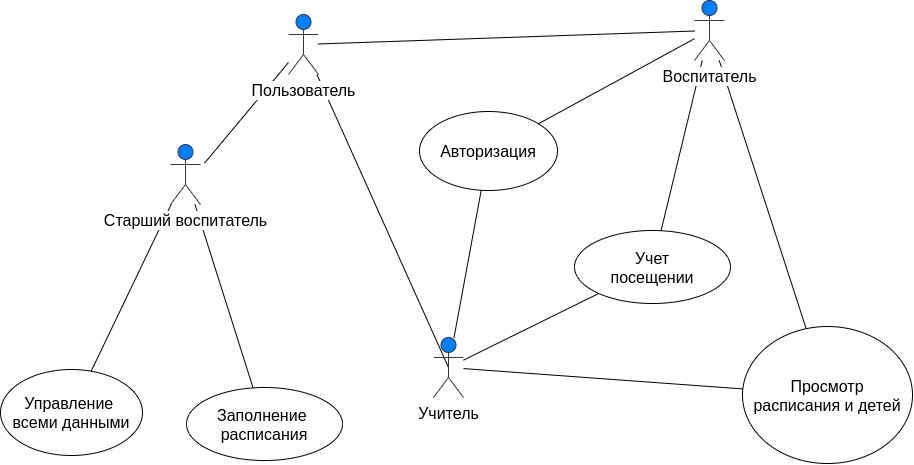


Рисунок 1.2 - Варианты использования системы

# 1.5 Диаграмма потоков данных

DFD [3] (Рисунок 1.3) — общепринятое сокращение от англ. data flow diagrams — диаграммы потоков данных. Так называется методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ.

Диаграмма потоков данных (data flow diagram, DFD) — один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML. Несмотря на имеющее место в современных условиях смещение акцентов от структурного к объектно-ориентированному подходу к анализу и проектированию систем, «старинные» структурные нотации по-прежнему широко и эффективно используются как в бизнес-анализе, так и в анализе информационных систем.

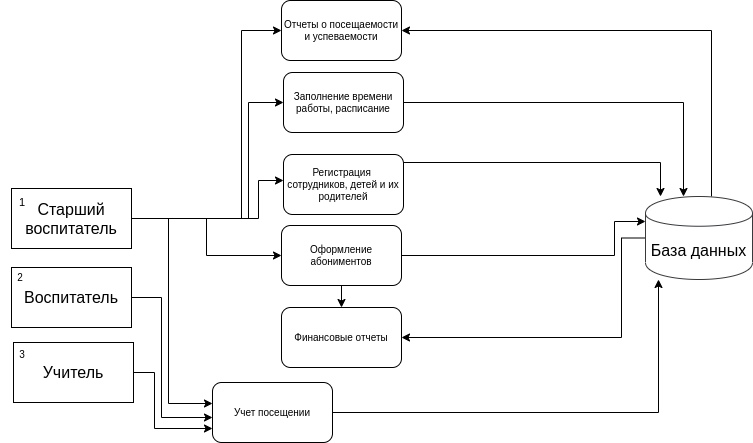


Рисунок 1.3 – Диаграмма потоков данных

# 2 РЕАЛИЗАЦИЯ ПРОЕКТА

# 2.1 Обоснование выбора СУБД

СУБД [4]— комплекс программ, позволяющих создать базу данных (БД) и манипулировать данными (вставлять, обновлять, удалять и выбирать). Система обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД. В качестве средства разработки БД была выбрана система MS SQL Server. Microsoft SQL Server — система управления базами данных (СУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

Плюсы Microsoft SQL Server:

* СУБД масштабируется, поэтому работать с ней можно на портативных ПК или мощной мультипроцессорной технике. Процессор может одновременно обрабатывать большой объем запросов;
* размер страниц – до 8 кб, поэтому данные извлекаются быстро, подробную и сложную информацию хранить удобнее. Система позволяет обрабатывать транзакции в интерактивном режиме, есть динамическая блокировка;
* рутинные административные задачи автоматизированы: это управление блокировками, памятью, редактура размеров файлов. У системы продуманы настройки, можно создать профили пользователей;
* реализован поиск по фразам, тексту, словам, можно создавать ключевые индексы;
* в SQL Server есть репликации через интернет, предусмотрена синхронизация. Есть полноценный веб-ассистент для форматирования страниц;
* в систему интегрирован сервер интерактивного анализа для принятия решений, создания корпоративных отчетов. Есть службы преобразования информации;
* запросы можно формулировать на английском языке, без программирования;
* СУБД поддерживает работу с другими продуктами Microsoft: Access, MS Excel.

# 2.2 Обоснование выбора языка и среды программирования

C# [5]— объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET. Компилятор с C# входит в стандартную установку самой .NET, поэтому программы на нём можно создавать и компилировать даже без инструментальных средств, вроде Visual Studio. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов, делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML. Переняв многое от своих предшественников — языков C++, Java, Delphi — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов.

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает FCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR.

Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем. (Однако эта закономерность была нарушена с выходом C# 3.0, представляющим собой расширения языка, не опирающиеся на расширения платформы .NET.) CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

# 2.3 Компоненты, используемые в программном проекте

* Button представляет элемент управления «кнопка Windows». Можно щелкнуть мышью, клавишей Ввод или пробел, если кнопка находится в фокусе;
* ComboBox образует выпадающий список. Для хранения элементов списка в ComboBox также предназначено свойство Items;
* ListView можно отображать сложные данные в различных столбцах, можно задавать данным изображения и пиктограммы;
* TextBlock элемент предназначен для вывода текстовой информации, для создания простых надписей;
* TextBox для ввода и редактирования текста. Текст элемента TextBox можно установить или получить с помощью свойства Text;
* GridView представляет режим просмотра, который отображает данные в столбцах для элемента управления ListView;
* DataGrid более расширенная версия Grandview, дает возможность добавлять новые строки, а также вставлять другие компоненты в ячейки таблицы;
* StackPanel располагает все элементы в ряд либо по горизонтали, либо по вертикали в зависимости от ориентации.;
* WrapPanel адаптивный StackPanel;
* RadioButton позволяет пользователю выбрать одну опцию из предопределенного набора;
* Frame представляет собой элемент управления содержимым, обычно используется в качестве контейнера для страницы;
* Border поддерживают рисование рамок вокруг своих границ, в отличие от панелей;
* ScrollViewer обеспечивает удобный способ прокрутки содержимого, в случае если размеры содержимого превышают размеры окна или страницы;
* DatePicker представляет интерфейс для выбора даты;
* Image контейнер для изображении.

А также были использованы следующие пакеты NuGet:

* MvvmLight представляет собой набор компонентов для использования шаблона MVVM в Silverlight, WPF и Windows Phone приложениях;
* EntityFrameworkCore представляет собой объектно-ориентированную технологию для доступа к данным;
* Fody позволяет сокращать полные свойства, с реализацией INotifyPropertyChanged;
* ControlzEx предоставляет набор дополнительных компонентов;
* SharpVectors предоставляет возможность использовать svg файлы;
* ClosedXML.Report для удобного составления отчетов на Excel;

# 2.4 Описание пользовательского интерфейса

Пользовательский интерфейс [6] (UI), в промышленный образец области взаимодействия человека с компьютером, это пространство, где происходит взаимодействие между людьми и машинами. Целью этого взаимодействия является обеспечение эффективного функционирования и управления машиной из человеческого конца, в то время как машина одновременно подает обратно информацию, которая помогает Операторам принятию решений процесса. Примеры этой широкой концепции пользовательских интерфейсов включают интерактивные аспекты компьютерной операционной системы, ручные инструменты, тяжелая техника управления оператора и процесса управления. Конструктивные соображения применимы при создании пользовательских интерфейсов связаны или связаны такие дисциплины, как эргономика и психологии.

Как правило, цель дизайна пользовательского интерфейса является создание пользовательского интерфейса, что делает его легким, эффективным и приятным (удобно).

Принципами разработки пользовательского интерфейса являются ясность и простота.

Ясность – это первая и главнейшая цель любого интерфейса. Для того, чтобы эффективно использовать разработанный интерфейс, люди должны легко выявлять его, осознавать, зачем они его используют, понимать, взаимодействие с каким объектом он обеспечивает, представлять, что произойдет при его использовании.

Простота – интерфейс должен быть простым. При этом имеется в виду не упрощение функционала программы, а обеспечение легкости в его изучении и использовании. Кроме того, он должен предоставлять доступ ко всему перечню функциональных возможностей, предусмотренных данной программой. Реализация доступа к широким функциональным возможностям и обеспечение простоты работы противоречат друг другу. Один из возможных путей поддержания простоты - представление на экране информации, минимально необходимой для выполнения пользователем очередного шага задания. В частности, необходимо избегать многословных команд или сообщений. Непродуманные или избыточные фразы затрудняют пользователю извлечение существенной информации.

В данном приложении был использован графический пользовательский интерфейс с применением технологии WPF.

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

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;
* 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.

Пользовательский интерфейс разрабатываемой информационной системы состоит из: SpalshScreen (Рисунок 2.4.1), окна авторизации (Рисунок 2.4.2), основного окна (Рисунок 2.4.3), окон редактирования данных (Рисунок 2.4.4) и окна уведомления (Рисунок 2.4.5).



Рисунок 2.4.1 – SplashScreen

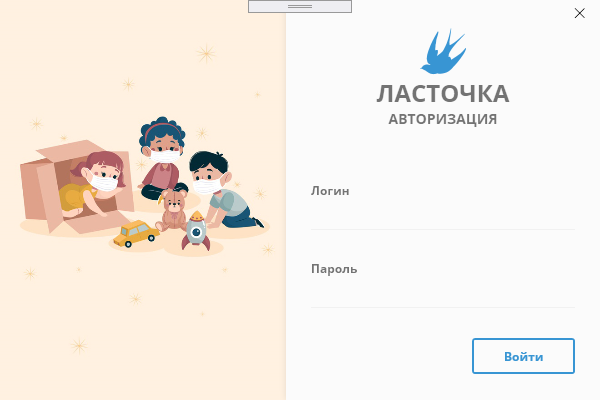


Рисунок 2.4.2 – Окно авторизации

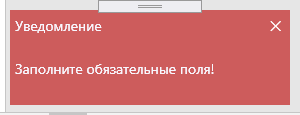


Рисунок 2.4.5 – Уведомления

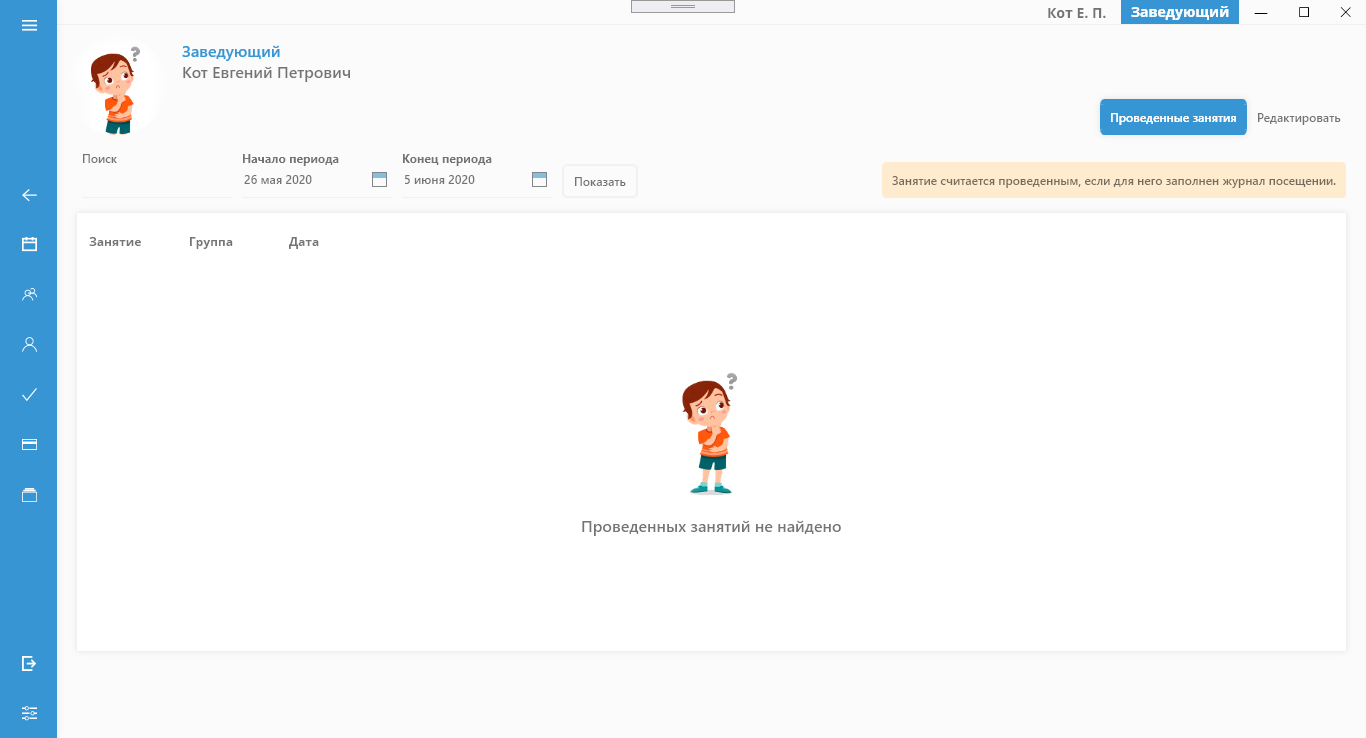


Рисунок 2.4.3 – Основное окно и страница проведенных занятии

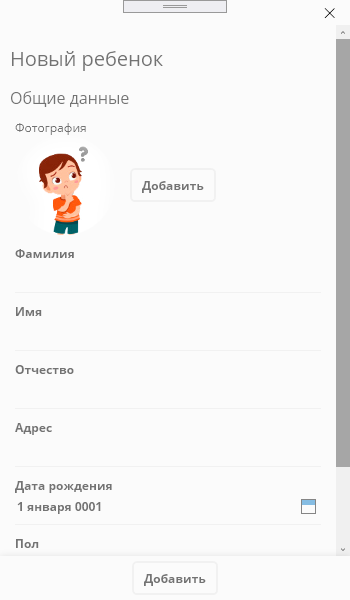


Рисунок 2.4.4 – Пример окна редактирования данных

# 2.5 Создание проекта

Для информационной системы была спроектирована ER-диаграмма (Рисунок 1.1). Далее по этой ER-диаграмме были созданы классы (Рисунок 2.5.1) и контекст базы данных (Рисунок 2.5.2). Это ускоряло разработку базы данных, так как при необходимости внести изменения в базу данных, не нужно было открывать MSSQL, запуск которой занимает много времени. Чтобы внести изменения в базу данных, надо было внести изменения в необходимый класс, создать миграцию и обновить базу данных (Рисунок 2.5.3).

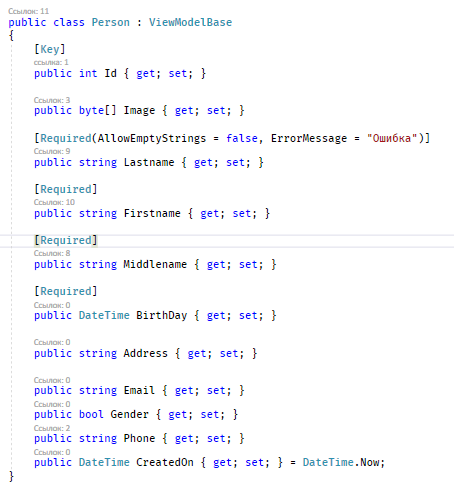


Рисунок 2.5.1 – Пример класса



Рисунок 2.5.2 – Контекст базы данных

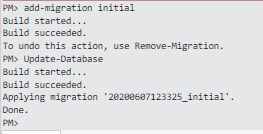


Рисунок 2.5.3 – Создание миграции и обновление базы данных

При разработке пользовательского интерфейса системы в проект был добавлен SplashScreen. SplashScreen (заставка) - это графический элемент управления, состоящий из окна, содержащего изображение, логотип и текущую версию программного обеспечения. Заставка обычно появляется во время запуска игры или программы (Рисунок 2.4.1).

Следующим шагом было разработать окно авторизации (Рисунок 2.5.4) и ее логику;

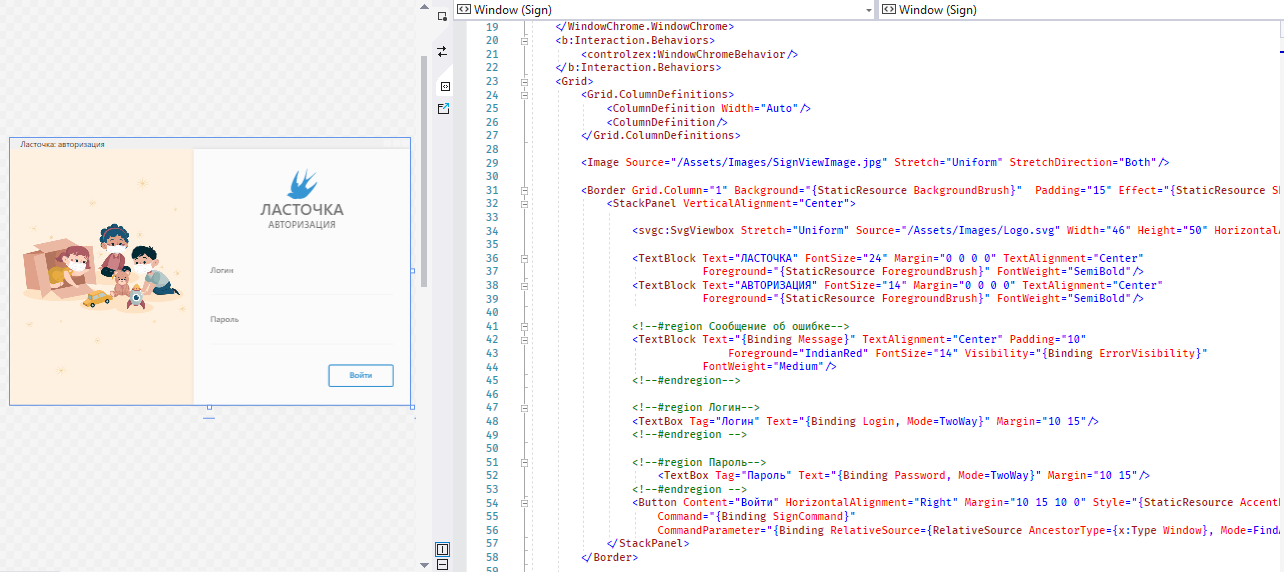


Рисунок 2.5.4 – Разработка окна авторизации

Далее был создан основное окно (Рисунки 2.5.5, 2.5.6). Из-за того, что почти вся разметка основного окна сделана как стиль, меню и кнопки не отображаются в предварительном просмотре.

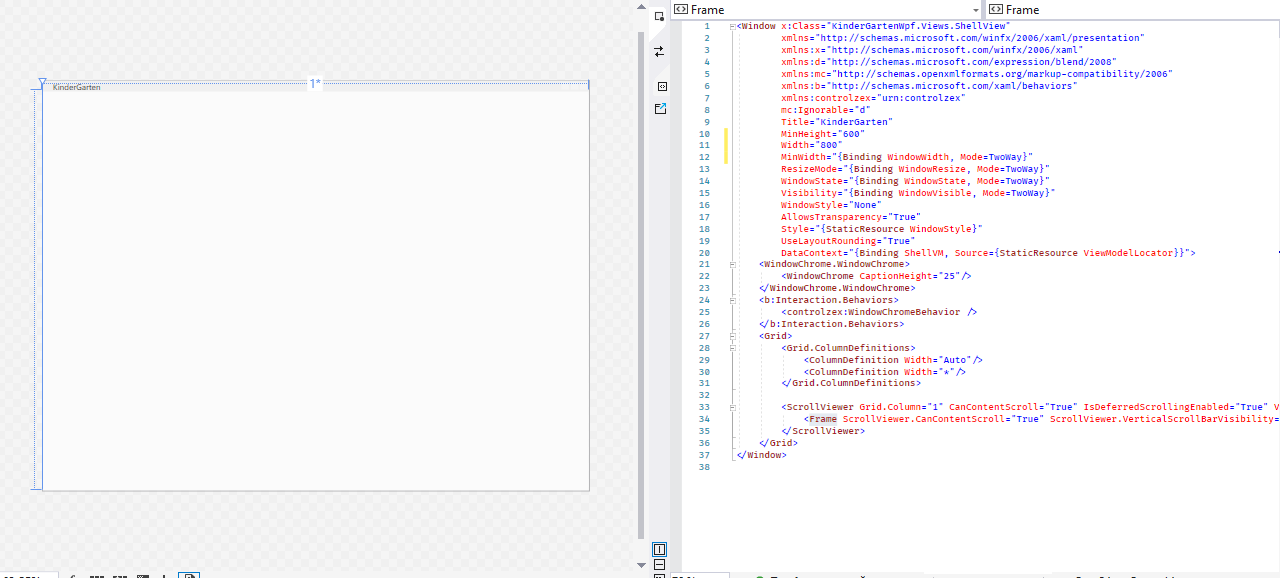


Рисунок 2.5.5 – Разработка основного окна

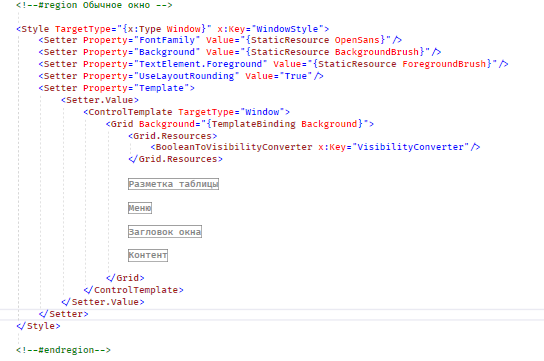


Рисунок 2.5.6 – Разработка стиля для основного окна

Также требовалось разработать окна для редактирования данных (Рисунок 2.5.7) и окно уведомлении (Рисунок 2.5.8).

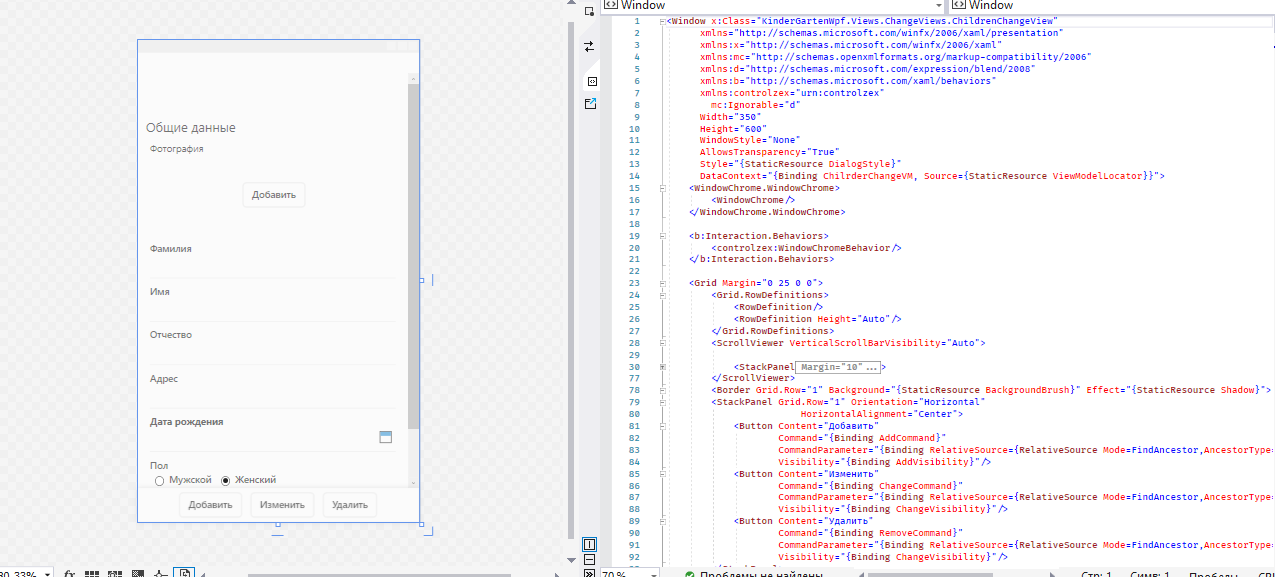


Рисунок 2.5.7 – Разработка окон редактирования данных

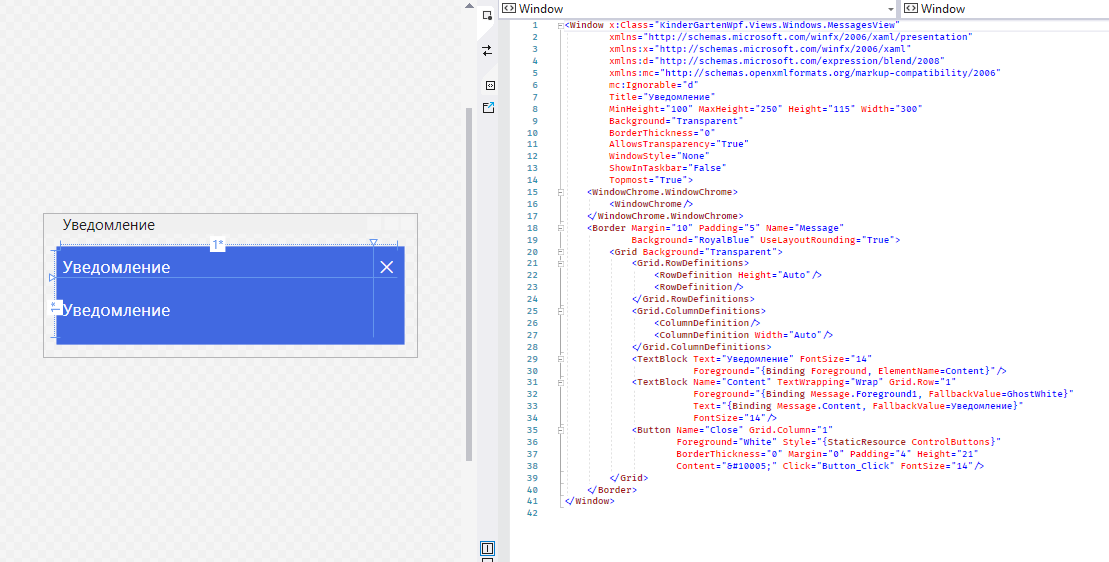


Рисунок 2.5.8 – Разработка окна уведомления

ViewModelLocator.cs – В этом классе используется ServiceLocator из пакета CommonServiceLocator и SimpleIoc из пакета MvvmLight для реализации доступа к сервисам указанных в конструкторах ViewModel’ей (Рисунок 2.5.9).

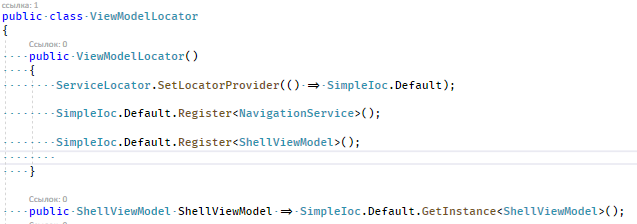


Рисунок 2.5.9 – ViewModelLocator.cs

В информационной системе для защиты пароля используется ее хеширование, поэтому для авторизации, регистрации и изменения пароля был создан специальный сервис AuthenticationService (Рисунок 2.5.10) Она использует класс PasswordHasher из пакета Microsoft.AspNet.Identity.

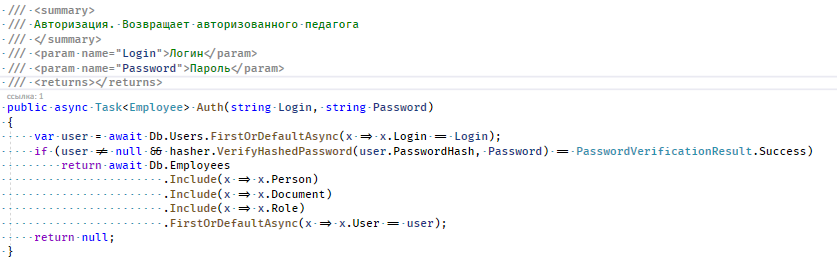


Рисунок 2.5.10 – AuthenticationService.cs

А для уведомлении был создан сервис MessageService, который создает экземпляр окна уведомлении (Рисунок 2.5.11). Она принимает 2 аргумента: тип(ошибка, успех, информация) и сообщение. В зависимости от типа, цвет уведомления меняетя.

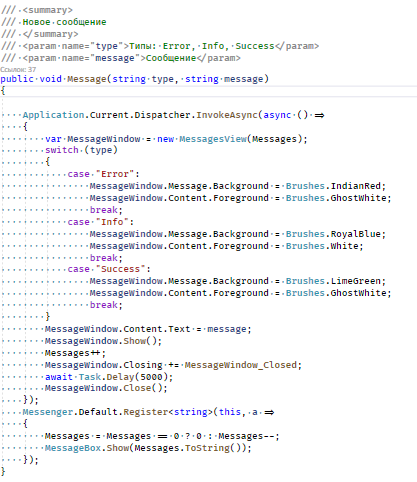


Рисунок 2.5.11 – MessageService

Все ViewModel’и наследуются из класса ViewModelBase из пакета MvvmLight, который реализует интерфейс INotifyPropertyChanged, и в основном эти ViewModel’и состоят из свойств (Рисунок 2.5.12), команд (Рисунок 2.5.13) и конструктора (Рисунок 2.5.14).

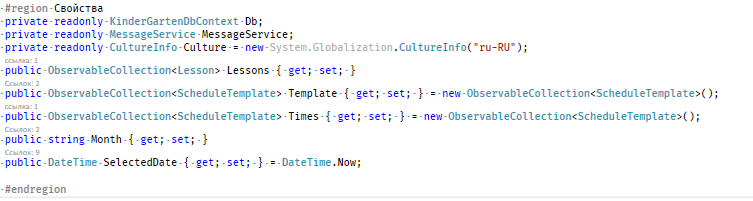


Рисунок 2.5.12 - Свойства ViewModel’ей

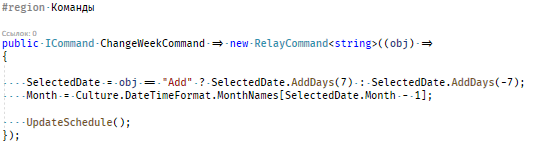


Рисунок 2.5.13 - Команды

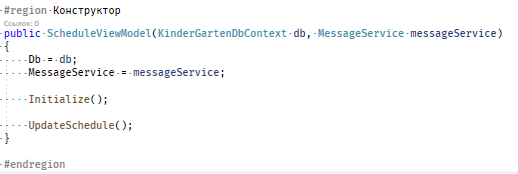


Рисунок 2.5.14 – Конструктор

Для ViewModel, создается страница, которая привязывается к нему и его свойствам (Рисунок 2.5.15).

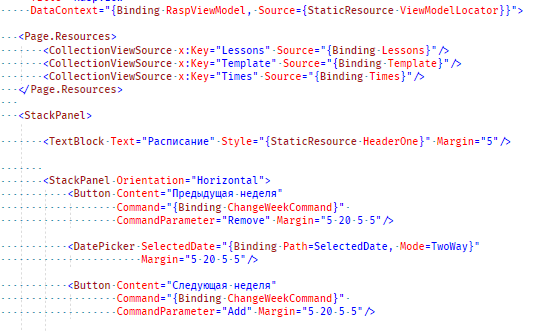


Рисунок 2.5.15 – Пример привязки данных

# 3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА

# 3.1 Основные понятия и принципы тестирования ПП

Тестирование программного продукта [7] - это процесс выполнения программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется тестовым или просто тестом. Тестирование программ является одной из составных частей более общего понятия - «отладка программ». Под отладкой по­нимается процесс, позволяющий получить программу, функциони­рующую с требующимися характеристиками в заданной области изменения входных данных.

Процесс отладки включает:

* действия, направленные на выявление ошибок (тестирование);
* диагностику и локализацию ошибок (определение характера ошибок и их местонахождение);
* внесение исправлений в программу с целью устранения ошибок.

Из трех перечисленных видов работ самым трудоемким и дорогим является тестирование, затраты на которое приближаются к 45 % общих затрат на разработку ПС.

Невозможно гарантировать отсутствие ошибок в программе. В лучшем случае можно попытаться показать наличие ошибок. Если программа правильно ведет себя для большого набора тестов, нет оснований утверждать, что в ней нет ошибок. Если считать, что набор тестов способен с большой вероятностью обнаружить возможные ошибки, то можно говорить о некотором уровне уверенности (надежности) в правильности работы программы, устанавливаемом этими тестами. Сформулируем следующее высказывание: если ваша цель - показать отсутствие ошибок, вы их найдете не слишком много. Если же ваша цель - показать наличие ошибок, вы найдете значительную их часть.

Надежность невозможно внести в программу в результате тестирования, она определяется правильностью этапов проектирования. Наилучшее решение проблемы надежности - с самого начала не допускать ошибок в программе. Однако вероятность того, что удастся безупречно спроектировать большую программу, мала. Роль тестирования состоит в том, чтобы определить местонахождение немногочисленных ошибок, оставшихся в хорошо спроектированной программе. Попытки с помощью тестирования достичь надежности плохо спроектированной программы безнадежны.

Тестирование оказывается довольно необычным процессом (поэтому и считается трудным), так как этот процесс разрушительный. Ведь цель проверяющего (тестовика) - заставить программу сбиться.

Программы, как объекты тестирования, имеют ряд особенно­стей, которые отличают процесс их тестирования от общепринято­го, применяемого при разработке аппаратуры и других технических изделий. Особенностями тестирования ПС являются:

* отсутствие эталона (программы), которому должна соответ­ствовать тестируемая программа;
* высокая сложность программ и принципиальная невозможность исчерпывающего тестирования;
* практическая невозможность создания единой методики тестирования (формализация процесса тестирования) в силу большого разнообразия программных изделий (ПИ) по их сложности, функциональному назначению, области использования и т.д.

# 3.2 Тестирование методом черного ящика

Метод тестирования «черного ящика»[8] – процедура получения и выбора тестовых случаев на основе анализа спецификации (функциональной или нефункциональной), компонентов или системы без ссылки на их внутреннее устройство.

Техники тестирования «черным ящиком»:

* эквивалентное разбиение;
* анализ граничных значений.

Эквивалентное разбиение. Эта техника включает в себя разделение входных значений на допустимые и недопустимые разделы и выбор репрезентативных значений из каждого раздела в качестве тестовых данных. Она может быть использована для уменьшения количества тестовых случаев.

Допустим, у нас есть целая переменная N в диапазоне от -99 до 99: позитивными классами эквивалентности будут [-99, -10], [-9, -1], 0, [1, 9], [10, 99], а недействительными (негативными) – <-99, >99, пустое значение, нечисловые строки.

Анализ граничных значений. Техника, которая включает в себя определение границ входных значений и выбор в качестве тестовых данных значений, находящихся на границах, внутри и вне границ. Многие системы имеют тенденцию вести себя некорректно при граничных значениях, поэтому оценка значений границ приложения очень важна. При проверке мы берем следующие величины: минимум, (минимум-1), максимум, (максимум+1), стандартные значения. Например, в том же случае -99 <= N <= 99 будет использоваться набор: -100, -99, -98, -10, -9 -1, 0, 1, 9, 10, 98, 99, 100.

Таблица 3.2.1 – Тестирование

|  |  |
| --- | --- |
| Общая информация о тестировании | |
| Название проекта | Информационная система для детского сада «Ласточка» |
| Номер версии | 0.0.1a |
| Имя тестера | Шакиров Рузаль Нурмахаматович |
| Дата тестирования | 27.05.2020 |
| Test Case #1 | |
| Test Case# | TC\_1 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка ввода специальных символов в поле номер телефона |
| Резюме испытания | Проверка невозможности ввода специальных символов |
|  |  |
|  | Продолжение таблицы |
| Шаги тестирования | Ввод специальных символов в поле номера телефона |
| Данные тестирования | +=\_?:%;»№!ф#\/ |
| Ожидаемый результат | Невозможность вписать специальные символы в поле номер телефона |
| Фактический результат | Специальные символы не вводятся в поле номер телефона |
| Предпосылки | Добавление ребенка |
| Статус (Pass/Fail) | Pass |
| Test Case #2 | |
| Test Case# | TC\_2 |
| Приоритет тестирования | Высокий |
| Название тестирования/Имя | Проверка авторизации |
| Резюме испытания | Проверка авторизации при кореектно и не корректно введенных данных |
| Шаги тестирования | Ввод данных в поля логин и пароль |
| Данные тестирования | 1. admin, admin; 2. admin, 123123; 3. user, user13231. |
| Ожидаемый результат | 1. успешная авторизация; 2. сообщение об ошибке; 3. сообщение об ошибке; |
| Фактический результат | 1. успешная авторизация; 2. сообщение об ошибке; 3. сообщение об ошибке; |
| Предпосылки | Авторизация |
| Статус (Pass/Fail) | Pass |

# 4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

# 4.1 Запуск программы

Запуск программы осуществляется двойным кликом по ярлыку. Пользователя встречает окно авторизации (Рисунок 2.4.1). Если пользователь не зарегистрирован или ввел неверные данные, то выводится соответствующее сообщение об ошибке (Рисунок 4.1).

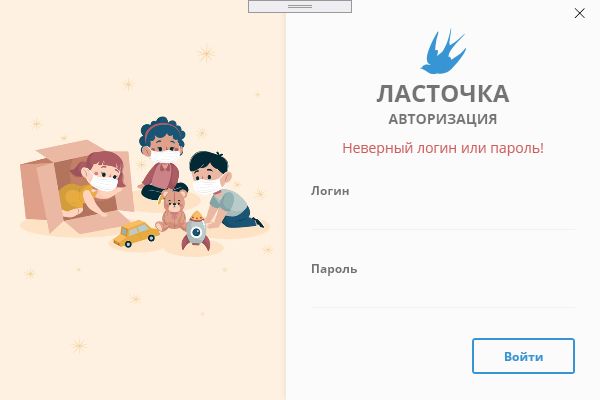


Рисунок 4.1 – Сообщение об ошибке

Если пользователь ввел правильные данные, то открывается основное окно (Рисунок 2.3)

Основное окно состоит из:

* верхней горизонтальной панели;
* меню;
* рабочей области.

Чтобы перейти в личный кабинет, необходимо нажать кнопку с инициалами пользователя, в верхней панели справа. В личном кабинете можно отредактировать данные профиля (Рисунок 4.2).

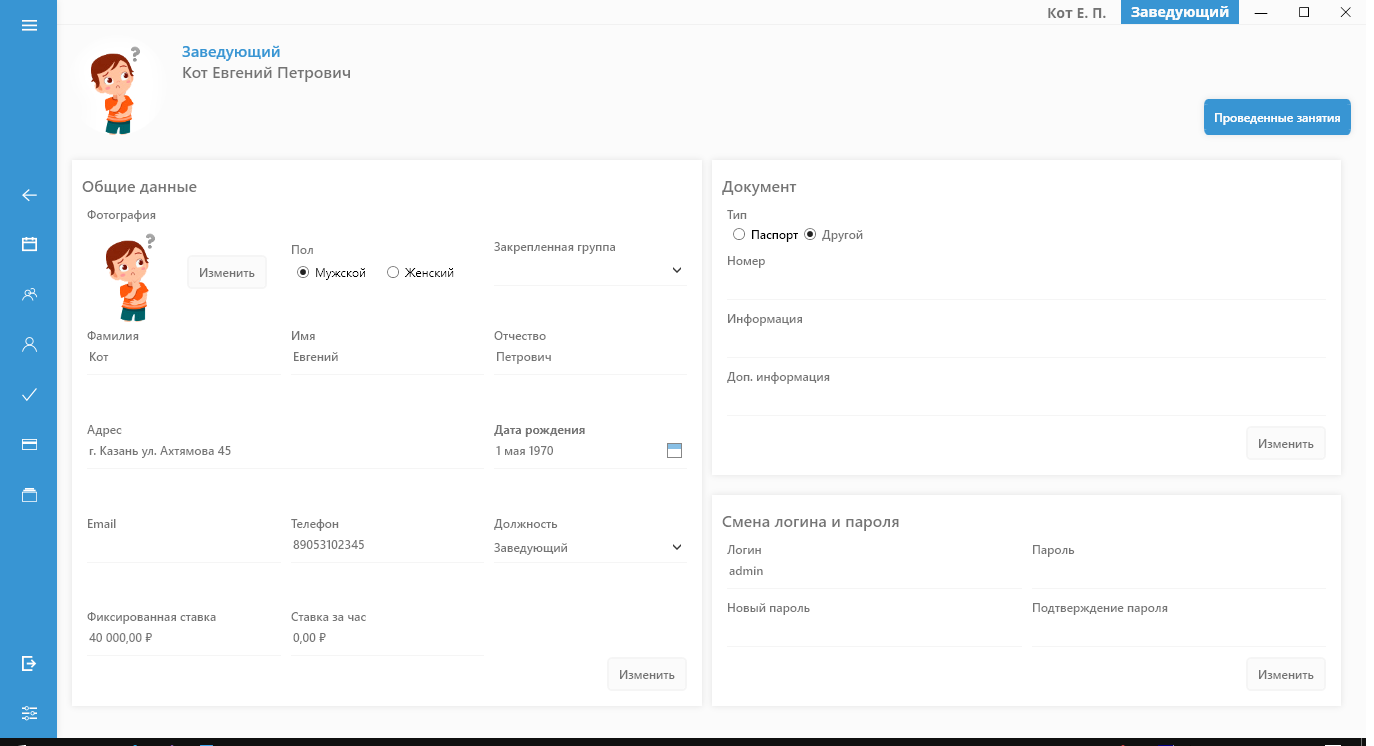


Рисунок 4.2 – Редактирование профиля

Навигация в системе осуществляется при помощи меню (синяя панель слева). Чтобы раскрыть меню необходимо нажать на первую кнопку, с тремя вертикально расположенными дефисами (Рисунок 4.3).

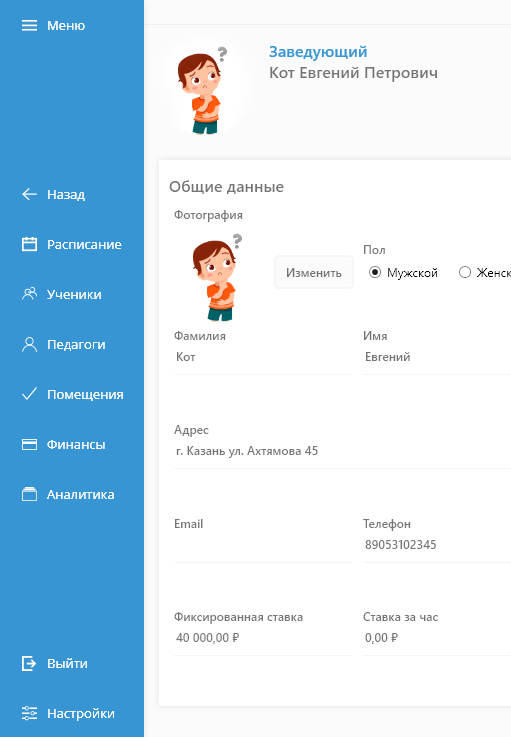


Рисунок 4.3 – Раскрытое меню

# 4.2 Работа с программой. Заведующий

После установки при первом запуске, чтобы авторизоваться необходимо ввести в поля логина и пароля: «admin», а также заполнить профиль в личном кабинете.

Следующим шагом необходимо заполнить соответствующие данные в настройках (Рисунок 4.4)

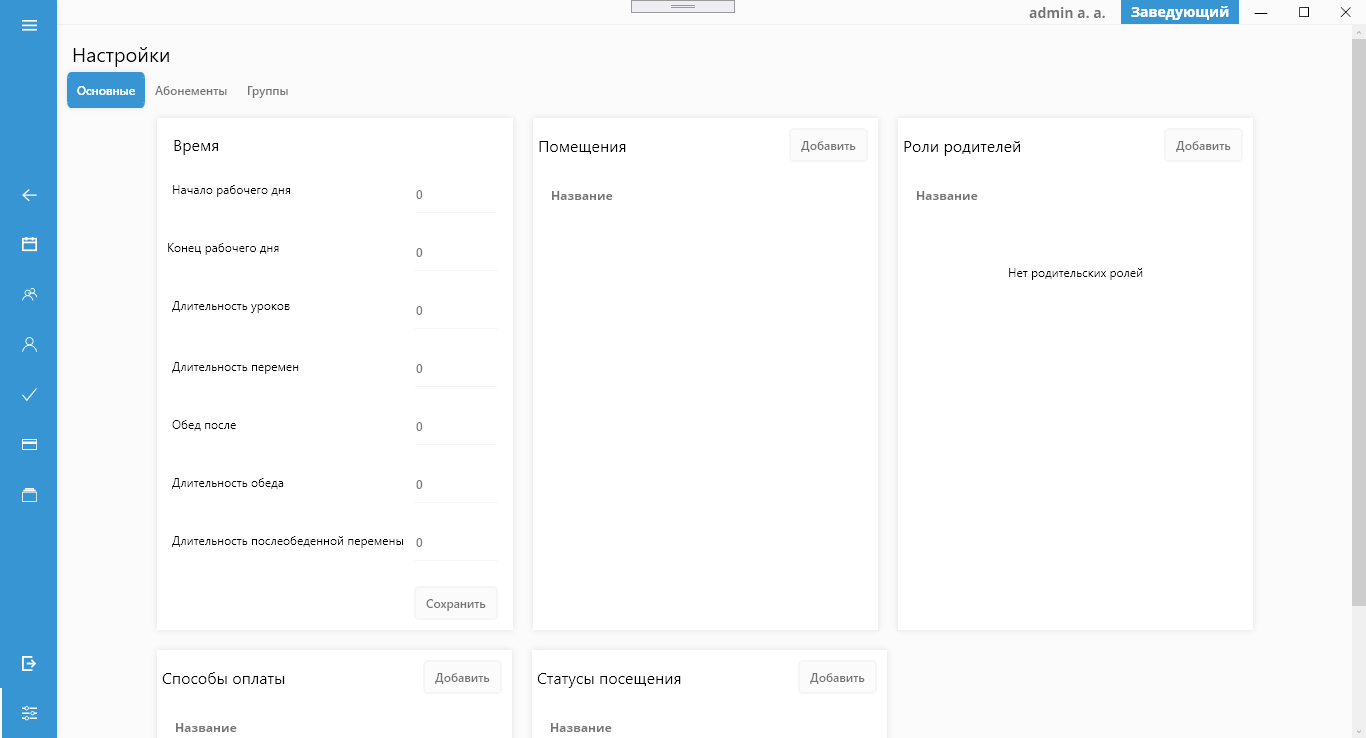


Рисунок 4.4 – Страница настроек

Чтобы добавить новых пользователей необходимо перейти на страницу педагоги (Рисунок 4.5). Нажать на кнопку «Добавить педагога», после чего откроется форма регистрации нового пользователя (Рисунок 4.6).

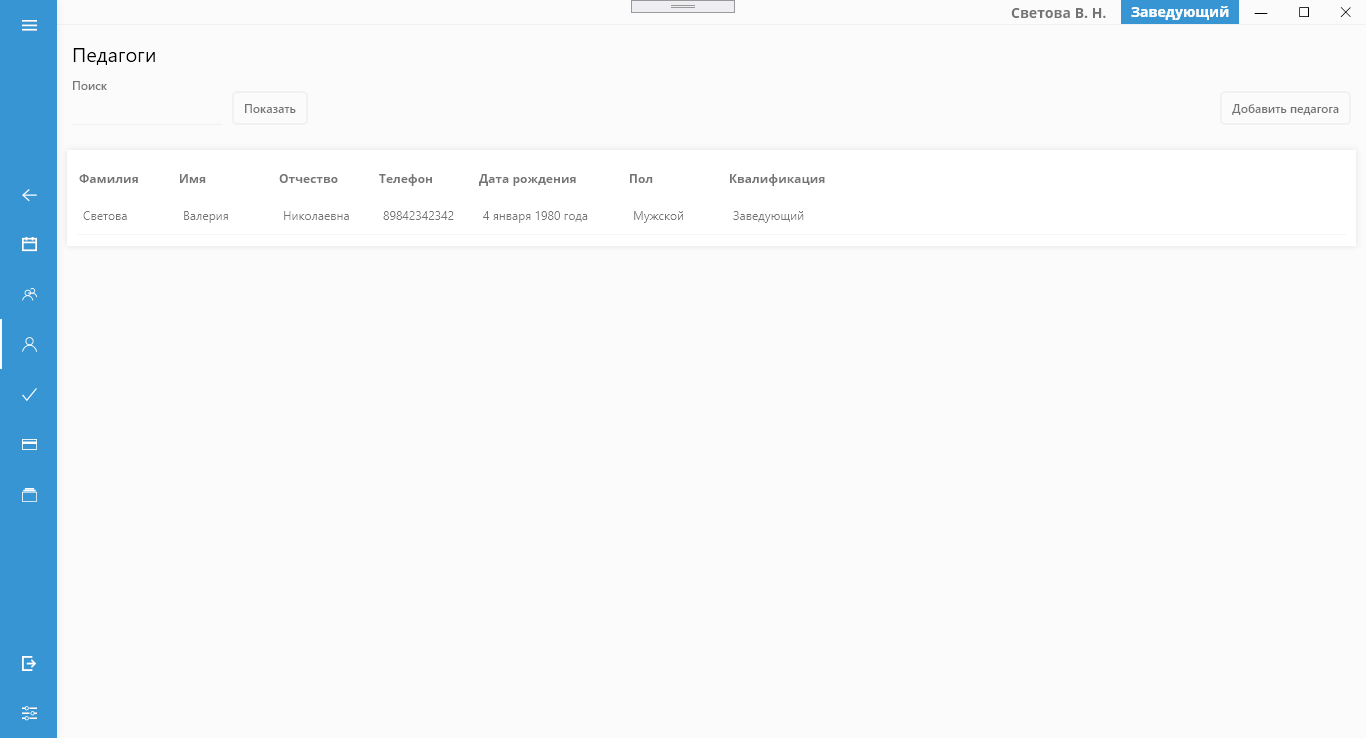


Рисунок 4.5 – Страница педагогов

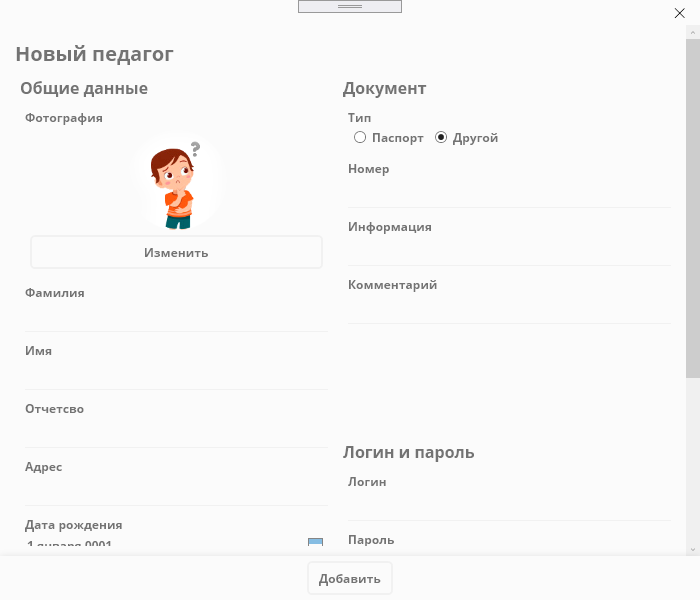


Рисунок 4.6 – Форма регистрации нового пользователя

В случае успешной или не успешной операции, система уведомляет об этом пользователя. (Рисунок 2.6).

Чтобы посмотреть список финансовых операции необходимо перейти на страницу финансы (Рисунок 4.7).

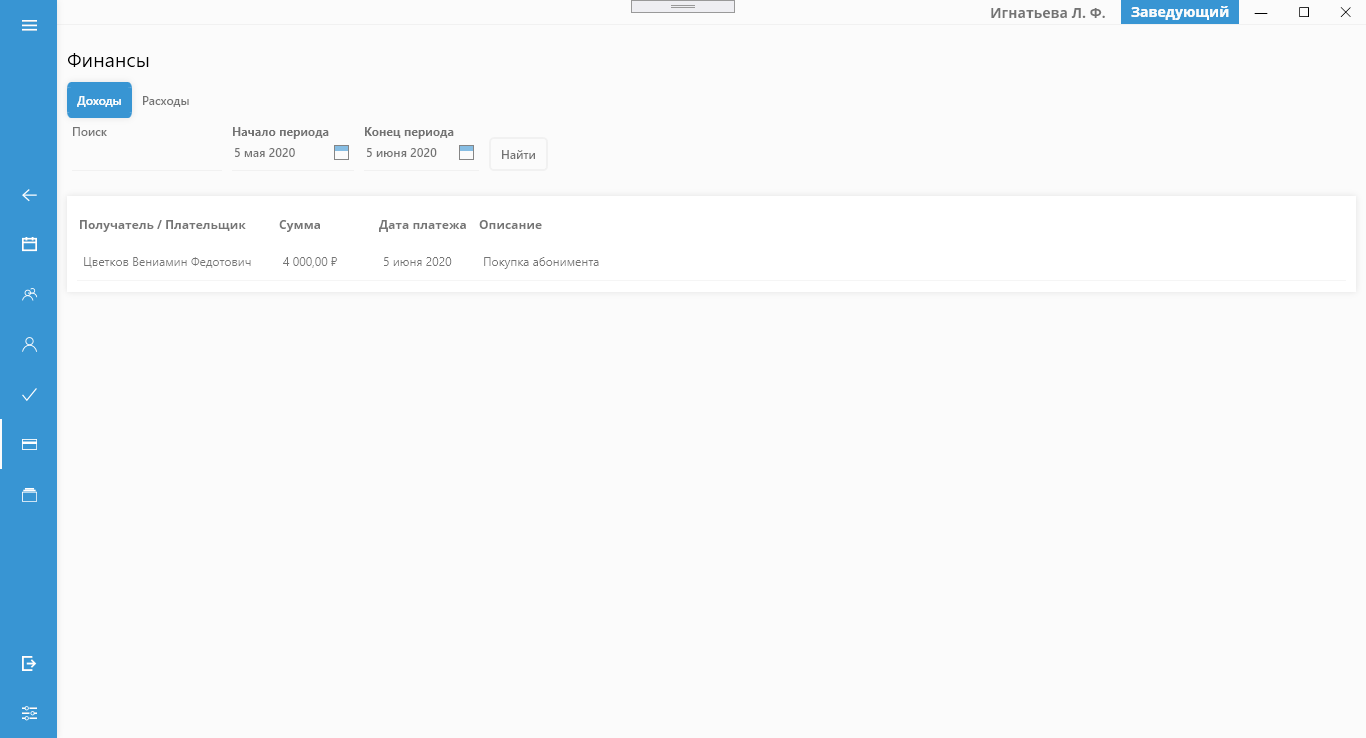


Рисунок 4.7 – Страница финансы

Чтобы выплатить сотрудникам зарплату, необходимо перейти во вкладку «Расходы» и нажать кнопку «Выплатить зарплату». Далее откроется форма выдачи зарплаты, в которой нужно выбрать сотрудника, тип ставки, метод оплаты, после чего нажать кнопку «Выплатить» (Рисунок 4.8)

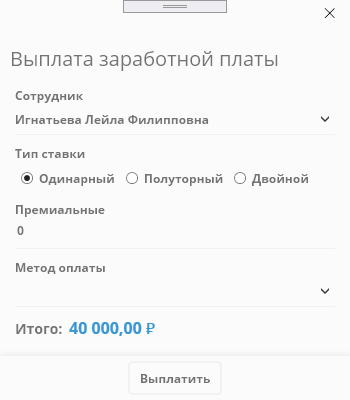


Рисунок 4.8 – Форма выплаты заработной платы

Для составления финансового отчета и отчета посещении необходимо перейти на страницу «Аналитика» (Рисунок 4.9). Далее выбрать соответствующую вкладку, указать период и нажать на кнопку «Составить отчет в Excel». Система автоматический создаст отчет в каталоге «Reports». Чтобы открыть составленный отчет (Рисунок 4.10), необходимо выбрать его в панели «Составленные отчеты».

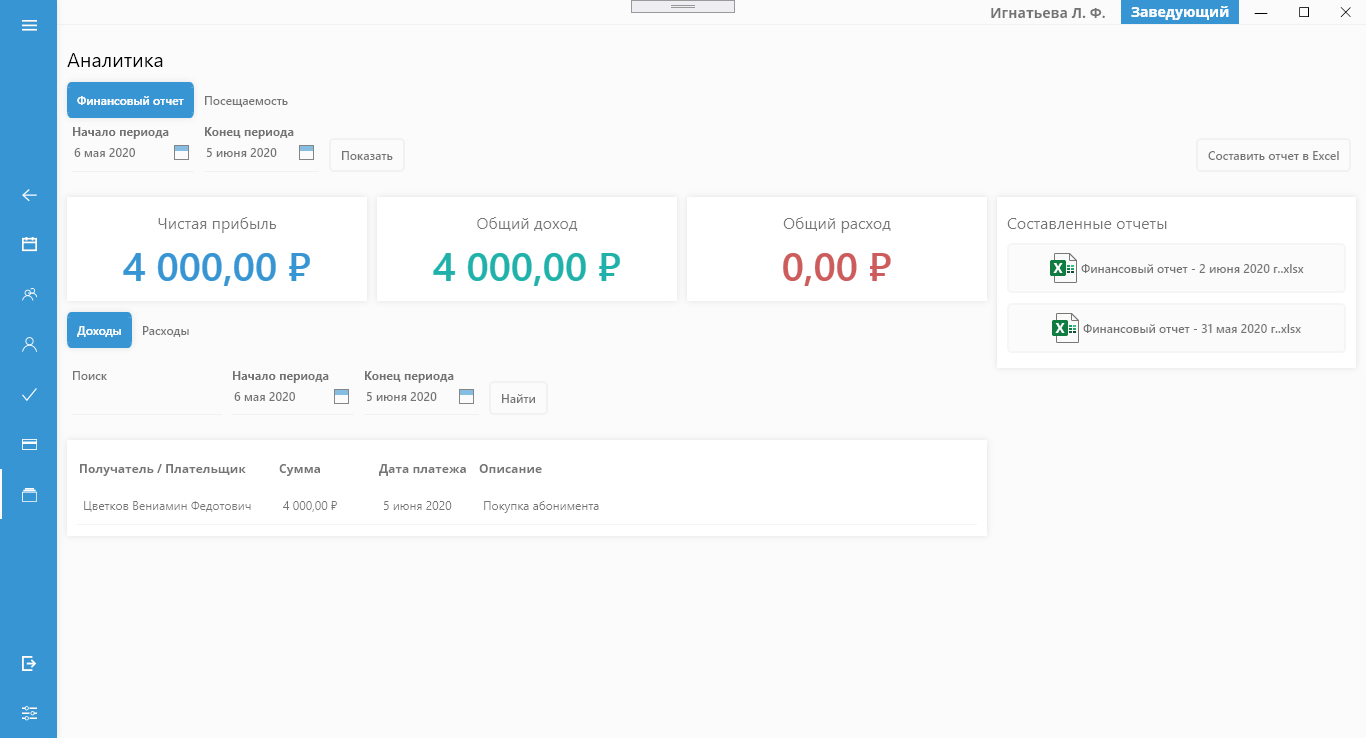


Рисунок 4.9 – Страница аналитики

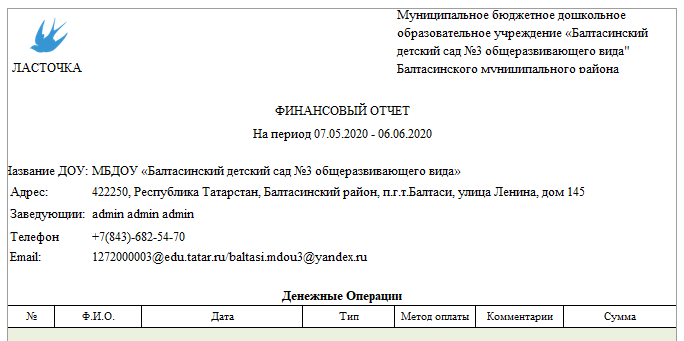


Рисунок 4.10 – Пример составленного отчета

# 4.3 Работа с программой. Старший воспитатель

Чтобы оформить абонемент, необходимо перейти на страницу дети и двойным щелчком выбрать ребенка из списка. Далее откроется профиль ребенка (Рисунок 4.11), со страницей абонементов. Затем нажать на кнопку «Оформить абонемент» и в открывшейся форме (Рисунок 4.12) заполнить соответствующие поля.

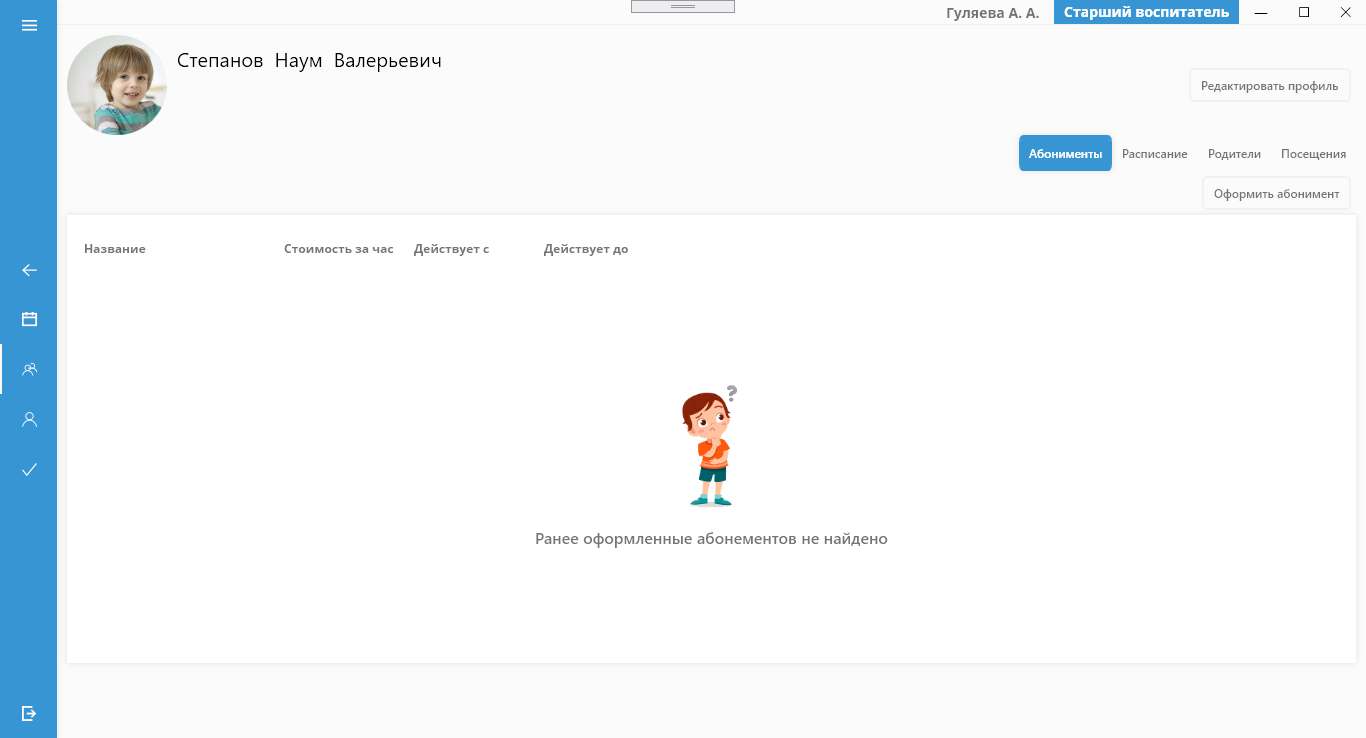


Рисунок 4.11 – Профиль ребенка

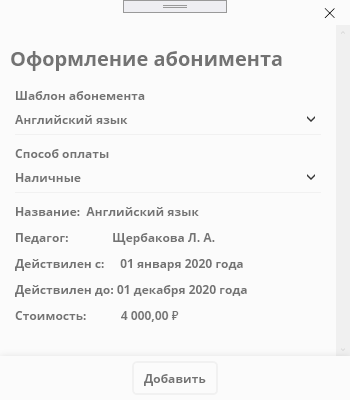


Рисунок 4.12 – Форма оформления абонемента

Для добавления нового занятия в расписание, необходимо перейти на страницу журнал посещении (Рисунок 4.13), нажать на кнопку «Добавить занятие» и заполнить поля в открывшейся форме (Рисунок 4.14)

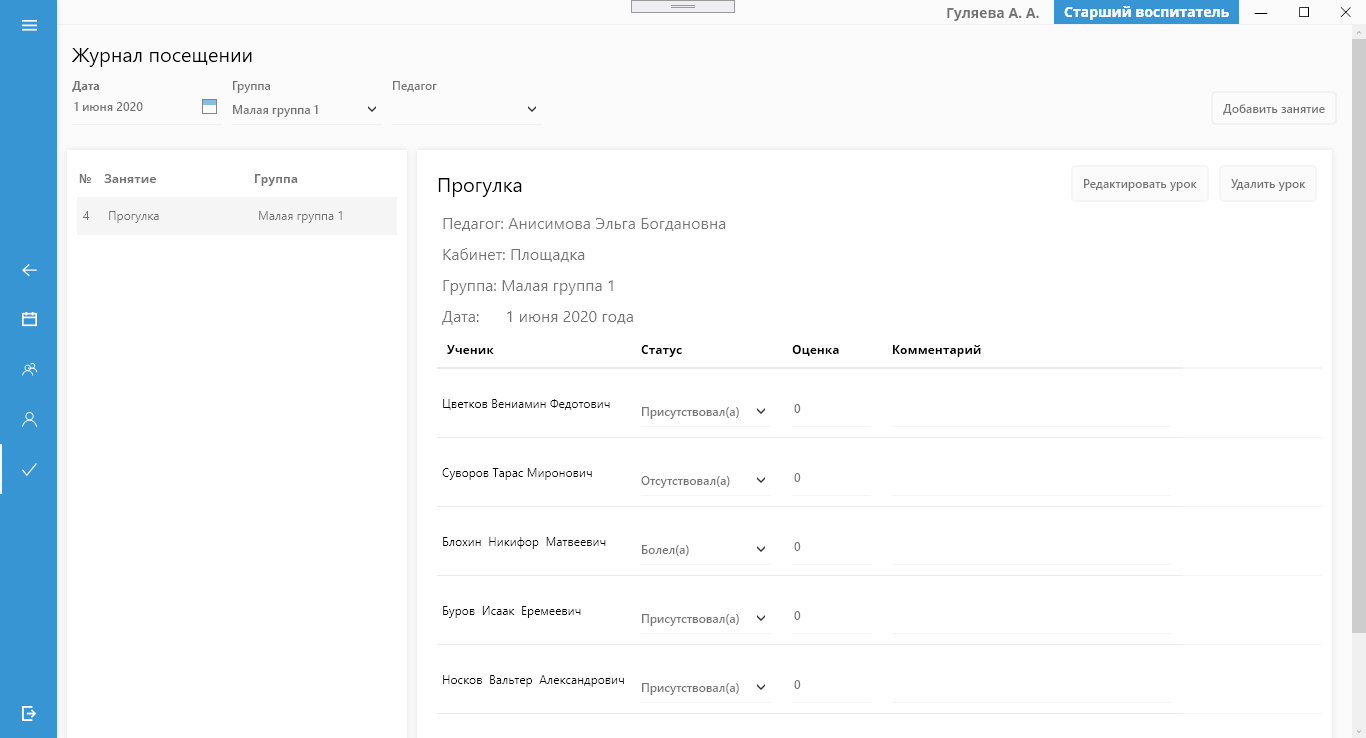


Рисунок 4.13 – Журнал посещении

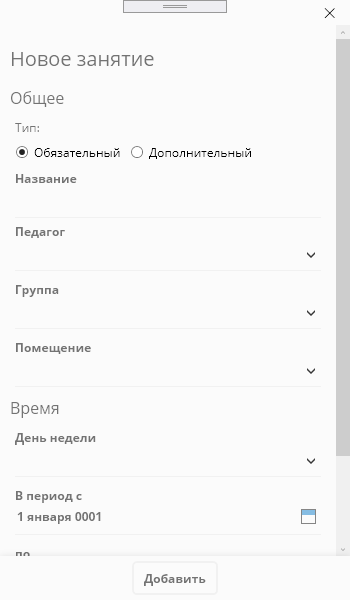


Рисунок 4.14 – Форма добавления занятия

Чтобы редактировать данные о занятии или вовсе удалить занятие из расписания, необходимо выбрать нужное занятие в журнале посещении, в списке слева. После этого, большая панель справа, заполнится данными о выбранном занятии и появятся 2 кнопки: «Редактировать занятие» и «Удалить занятие».

# ЗАКЛЮЧЕНИЕ

Целью данной выпускной квалификационной работы являлось разработка информационной системы детского сада.

В результате разработки были решены следующие задачи:

* проанализированы требования к программному продукту;
* выделены проблемы в управлении ДОУ;
* выявлены методы решения проблем;
* составлены ER, UML, DFD диаграммы;
* спроектирована база данных;
* разработана информационная система детского сада;
* протестирована информационная система;
* произведены апробация и внедрение информационной системы в ДОУ.

Изучение проблем в управлении ДОУ позволило сделать следующие выводы:

* воспитателям и педагогам, помимо того, чтобы просто следить за детьми, надо каждое занятие, прогулку, дополнительную работу нужно поминутно расписывать вручную;
* заведующей необходимо распределять детей в группы, получать записи воспитателей, и проанализировав их, составлять отчеты.

Все это занимает очень много времени, сил и нервов.

В целях совершенствования в управлении ДОУ была разработана информационная система со следующим функционалом:

* ведение списка детей, родителей и персонала;
* ведение сетки расписание и журнала посещении;
* ведение финансовых операции;
* составление финансовых отчетов и отчетов посещения.

Что позволило упростить работу методиста и предоставило дополнительное время для выполнения других обязанностей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ER-диаграмма: определение и обзор. Режим доступа - <https://www.lucidchart.com/pages/ru/erd-диаграмма>;
2. Элементы графической нотации диаграммы вариантов использования. НОУ ИНТУИТ. Режим доступа - <https://www.intuit.ru/studies/courses/32/32/lecture/1004>;
3. DFD – Википедия. Свободная энциклопедия. Режим доступа - <https://ru.wikipedia.org/wiki/DFD>;
4. Система управления базами данных. Свободная энциклопедия. Режим доступа - <https://ru.wikipedia.org/wiki/Система_управления_базами_данных>;
5. C Sharp. Свободная энциклопедия. Режим доступа -<https://vlab.wikia.org/ru/wiki/C_Sharp>;
6. Пользовательский интерфейс - User interface. Свободная энциклопедия. Режим доступа - <https://ru.qwe.wiki/wiki/User_interface>;
7. Тестирование и отладка. Научная электронная библиотека. Режим доступа - <https://www.monographies.ru/en/book/section?id=4632>;
8. Особенности тестирования «черного ящика». Свободная энциклопедия. Режим доступа - <https://quality-lab.ru/blog/key-principles-of-black-box-testing/>;

# ПРИЛОЖЕНИЕ А

1 Обоснование необходимости разработки программ

1.1 Введение

Настоящее техническое задание распространяется на разработку программного продукта: Информационная система детского сада.

1.2 Наименование программного продукта

Наименование программного продукта: Информационная система для детского сада.

1.3 Постановка задачи

Программный продукт должна иметь 4 подсистемы:

* подсистема для заведующего;
* подсистема для старшего воспитателя;
* подсистема для воспитателя;
* подсистема для педагога;

Программный продукт предназначена для автоматизации:

* учета посещении;
* учета доходов и расходов;
* составления отчетов посещении и финансовых операции;
* составления сетки расписания.

1.4 Выбор и обоснование критериев эффективности и качества

Удобный и понятный интерфейс позволяют работать с программой без предварительной подготовки. Программный продукт должен обеспечивать возможность выполнения перечисленных ниже свойств:

* гибкость – настраивается на любую структуру, для широкого круга деятельности;
* безопасность – ограничение общего доступа к заранее определенным ресурсам базы данных;
* целостность и доступность – информация в базе данных должна быть представлена в общепринятых форматах, защищенных от изменений;
* возможность поиска – фильтрация и сортировка по базе данных.

2 Выполнение научно-исследовательских работ

2.1 Определение структуры входных и выходных данных

К входным данным относятся:

* информация о детях и их родителей;
* информация о сотрудниках;
* информация о абонементах;
* информация о финансовых операции;

К выходным данным относятся:

* отчет о посещениях;
* отчет о финансовых операциях.

2.2 Предварительный выбор методов решения задач

Данный программный продукт рекомендуется реализовывать средствами «WPF». «WPF» - это 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. Это система для разработки, включающая в себя набор инструментов, механизмов и сервисов, обеспечивающих все необходимое для разработки прикладных решений, их работы, администрирования и обслуживания.

2.3 Определение требований к техническим средствам

В состав технических средств должен входить персональный компьютер, имеющий следующие характеристики:

* процессор Pentium-2.0Mz, не менее;
* оперативную память объемом 2 Гигабайт, не менее;
* HDD, 40 Гигабайт, не менее;
* операционную систему Windows 7, 8, 10;
* установленный «SQL Server».

2.4 Обоснование принципиальной возможности решения поставленных задач

Для эффективной работы с помощью данного программного продукта необходимо иметь ПК с вышеперечисленными характеристиками.

3 Разработка и утверждение технического задания

3.1 Определение требований к программному продукту

Основные возможности программы:

* просмотр/добавление/изменение/удаление информации о детях;
* просмотр/добавление/изменение/удаление информации об сотрудниках;
* просмотр/добавление/изменение/удаление сведений о занятии;
* формирование отчетной документации о посещениях;
* формирование отчетной документации о финансовых операциях;
* просмотр/добавление/изменение/удаление информации о родителях;

3.2 Определение стадий, этапов и сроков разработки программного продукта и документации на нее

При создании программного продукта можно выделить следующие основные этапы:

* исследование данной предметной области;
* разработка информационной системы;
* написание руководства пользователя;
* тестирование программного продукта.
* апробация и внедрение.

Программный продукт должен быть полностью разработан, протестирован внедрен и задокументирован до 8 июня 2020 г.

3.3 Выбор средств разработки

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, элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление.

Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Windows Forms, где используется GDI/GDI+. Производительность WPF выше, чем у GDI+ за счёт использования аппаратного ускорения графики через DirectX.

3.4 Согласование и утверждение технического задания

На основании договора на оказание услуг по разработке программного обеспечения, Исполнитель совместно с Заказчиком подписывает Акт приемки-сдачи программного продукта в эксплуатацию.

Дата: «\_\_» \_\_\_\_\_\_\_\_ 2020г.

Руководитель ВКР:

Преподаватель ГАПОУ «МЦК-КТИТС» \_\_\_\_\_\_\_\_\_\_\_\_ Ф.К. Сулимов

(подпись)

Руководитель учреждения:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

(должность) (подпись) (инициалы и фамилия)

Принял к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_ Р.Н. Шакиров

(подпись)

М.П.

# ПРИЛОЖЕНИЕ Б

using System;

using System.Globalization;

using System.Windows.Data;

namespace KinderGartenWpf.Converters

{

public class DateToColumnConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

return (int)((DateTime)value).DayOfWeek;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

return System.Windows.DependencyProperty.UnsetValue;

}

}

}

using System;

using System.Globalization;

using System.Windows.Data;

namespace KinderGartenWpf.Converters

{

public class DayConventer : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

int i = (int)value;

return i - 1;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

int i = (int)value;

return i + 1;

}

}

}

using System;

using System.Drawing;

using System.Globalization;

using System.Windows;

using System.Windows.Data;

using System.Windows.Interop;

using System.Windows.Media.Imaging;

namespace KinderGartenWpf.Converters

{

public class FullNameToIconConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

using (Icon ico = Icon.ExtractAssociatedIcon((string)value))

return Imaging.CreateBitmapSourceFromHIcon(ico.Handle, Int32Rect.Empty, BitmapSizeOptions.FromEmptyOptions());

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

return DependencyProperty.UnsetValue;

}

}

}

using System;

using System.Globalization;

using System.Windows;

using System.Windows.Data;

namespace KinderGartenWpf.Converters

{

public class GenderConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

var gender = (bool)value ? "Мужской" : "Женский";

return gender;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

return DependencyProperty.UnsetValue;

}

}

}

using KinderGartenWpf.Models.Objects;

using System;

using System.Globalization;

using System.Linq;

using System.Windows;

using System.Windows.Data;

namespace KinderGartenWpf.Converters

{

class PersonNameConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

var Person = (Person)value;

if (value != null)

{

return $"{Person.Lastname} {Person.Firstname.First()}. {Person.Middlename.First()}.";

}

return null;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

return DependencyProperty.UnsetValue;

}

}

}

using System;

using System.Globalization;

using System.Windows;

using System.Windows.Data;

using System.Windows.Media;

namespace KinderGartenWpf.Converters

{

public class TariffToBackgroundConventer : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

var DefaultBG = new SolidColorBrush { Color = new Color { R = 0, G = 0, B = 0, A = 10 } };

if (value is null)

return DefaultBG;

return new SolidColorBrush { Color = Colors.LightGoldenrodYellow };

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

return DependencyProperty.UnsetValue;

}

}

}

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using Microsoft.AspNet.Identity;

using Microsoft.EntityFrameworkCore;

using System.Threading.Tasks;

namespace KinderGartenWpf.Services

{

public class AuthenticationService

{

#region Свойства

private readonly IPasswordHasher hasher = new PasswordHasher();

private readonly KinderGartenDbContext Db;

#endregion

#region Конструктор

public AuthenticationService(KinderGartenDbContext db)

{

Db = db;

}

#endregion

#region Методы

/// <summary>

/// Авторизация. Возвращает авторизованного педагога

/// </summary>

/// <param name="Login">Логин</param>

/// <param name="Password">Пароль</param>

/// <returns></returns>

public async Task<Employee> Auth(string Login, string Password)

{

var user = await Db.Users.FirstOrDefaultAsync(x => x.Login == Login);

if (user != null && hasher.VerifyHashedPassword(user.PasswordHash, Password) == PasswordVerificationResult.Success)

return await Db.Employees

.Include(x => x.Person)

.Include(x => x.Document)

.Include(x => x.Role)

.FirstOrDefaultAsync(x => x.User == user);

return null;

}

/// <summary>

/// Регистрация. Регистрирует и возвращает нового пользователя

/// </summary>

/// <param name="login">Логин</param>

/// <param name="password">Пароль</param>

/// <returns></returns>

public User Reg(string login, string password)

{

var user = new User

{

Login = login,

PasswordHash = hasher.HashPassword(password)

};

return user;

}

/// <summary>

/// Изменение пароля, возвращает булевый тип, true - успешно

/// </summary>

/// <param name="UserId">Id пользователя</param>

/// <param name="Password">Старый пароль</param>

/// <param name="newPassword">Новый пароль</param>

/// <returns></returns>

public async Task<bool> UpdatePassword(int UserId, string Password, string newPassword)

{

var user = await Db.Users.FirstOrDefaultAsync(x => x.Id == UserId);

if (hasher.HashPassword(Password) == user.PasswordHash)

{

user.PasswordHash = hasher.HashPassword(newPassword);

await Db.SaveChangesAsync();

return true;

}

return false;

}

#endregion

}

}

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Views.Windows;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Media;

namespace KinderGartenWpf.Services

{

public class MessageService

{

public int Messages { get; set; }

/// <summary>

/// Новое сообщение

/// </summary>

/// <param name="type">Типы: Error, Info, Success</param>

/// <param name="message">Сообщение</param>

public void Message(string type, string message)

{

Application.Current.Dispatcher.InvokeAsync(async () =>

{

var MessageWindow = new MessagesView(Messages);

switch (type)

{

case "Error":

MessageWindow.Message.Background = Brushes.IndianRed;

MessageWindow.Content.Foreground = Brushes.GhostWhite;

break;

case "Info":

MessageWindow.Message.Background = Brushes.RoyalBlue;

MessageWindow.Content.Foreground = Brushes.White;

break;

case "Success":

MessageWindow.Message.Background = Brushes.LimeGreen;

MessageWindow.Content.Foreground = Brushes.GhostWhite;

break;

}

MessageWindow.Content.Text = message;

MessageWindow.Show();

Messages++;

MessageWindow.Closing += MessageWindow\_Closed;

await Task.Delay(5000);

MessageWindow.Close();

});

Messenger.Default.Register<string>(this, a =>

{

Messages = Messages == 0 ? 0 : Messages--;

});

}

private void MessageWindow\_Closed(object sender, System.EventArgs e)

{

Messages--;

}

}

}

using GalaSoft.MvvmLight.Messaging;

namespace KinderGartenWpf.Services

{

public class NavigationService

{

/// <summary>

/// Переход на другую страницу

/// </summary>

/// <param name="Page"></param>

public void Navigate(string Page)

{

Messenger.Default.Send(new NotificationMessage<string>(Page, "Navigate"));

}

/// <summary>

/// Назад

/// </summary>

public void Back()

{

Messenger.Default.Send(new NotificationMessage<string>(null, "Back"));

}

}

}

using GalaSoft.MvvmLight.Command;

using System.Windows;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels.Base

{

public class DialogViewModel : ViewModelBase

{

#region Команды

//Команда закрыть окно

public ICommand CloseCommand =>

new RelayCommand<Window>((obj) => { obj.Close(); });

// Команда перетаскивания окна

public ICommand WindowDragCommand =>

new RelayCommand<Window>((obj) =>

{

try { obj.DragMove(); }

catch { }

});

#endregion

}

}

using KinderGartenWpf.Models;

using KinderGartenWpf.Services;

namespace KinderGartenWpf.ViewModels.Base

{

public class ViewModelBase : GalaSoft.MvvmLight.ViewModelBase

{

public NavigationService NavService;

public KinderGartenDbContext Db;

public MessageService MessageService;

public int UserRole { get; set; }

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using Microsoft.Win32;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels.Dialog

{

public class ChildrenChangeViewModel : DialogViewModel

{

#region Свойства

public Visibility AddVisibility { get; set; }

public Visibility ChangeVisibility { get; set; }

public Children Children { get; set; }

public List<Group> Groups { get; set; }

public string Title { get; set; }

#endregion

#region Конструктор

public ChildrenChangeViewModel(KinderGartenDbContext db, MessageService messageService)

{

Db = db;

MessageService = messageService;

Groups = Db.Groups.ToList();

Messenger.Default.Register<string>(this, message =>

{

if (message == "UpdateGroups")

Groups = Db.Groups.ToList();

});

Messenger.Default.Register<NotificationMessage<Children>>(this, message =>

{

if (message.Notification == "Add")

{

Title = "Новый ребенок";

Children = new Children

{

Person = new Person(),

};

AddVisibility = Visibility.Visible;

ChangeVisibility = Visibility.Collapsed;

}

else if (message.Notification == "Change")

{

Children = message.Content;

Title = "Редактирование ребенка";

AddVisibility = Visibility.Collapsed;

ChangeVisibility = Visibility.Visible;

}

});

}

#endregion

#region Команды

// Команда добавить

public ICommand AddCommand => new RelayCommand<Window>((obj) =>

{

Db.Childrens.Add(Children);

Db.SaveChanges();

obj.DialogResult = true;

Children = null;

obj.Close();

MessageService.Message("Success", "Ребенок добавлен!");

});

// Команда изменить

public ICommand ChangeCommand => new RelayCommand<Window>((obj) =>

{

Db.SaveChanges();

obj.DialogResult = true;

Children = null;

obj.Close();

MessageService.Message("Success", "Данные успешно изменены!");

});

// Команда удалить

public ICommand RemoveCommand => new RelayCommand<Window>((obj) =>

{

Db.Childrens.Remove(Children);

Db.SaveChanges();

obj.DialogResult = true;

Children = null;

obj.Close();

MessageService.Message("Success", "Ребенок удален!");

});

public ICommand PhotoChangeCommand => new RelayCommand(() =>

{

OpenFileDialog dlg = new OpenFileDialog();

ImageConverter imageConverter = new ImageConverter();

dlg.InitialDirectory = "";

dlg.Filter = "Image files (\*.jpg,\*.png,\*.bmp)|\*.jpg;\*.png;\*.bmp";

if (dlg.ShowDialog() == true)

Children.Person.Image = (byte[])imageConverter.ConvertTo(System.Drawing.Image.FromFile(dlg.FileName), typeof(byte[]));

});

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using Microsoft.Win32;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels.Dialog

{

public class EmployeeAddViewModel : DialogViewModel

{

#region Свойства

private readonly AuthenticationService AuthService;

public Visibility AddVisibility { get; set; }

public Visibility ChangeVisibility { get; set; }

public Employee Employee { get; set; }

public string Title { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public string ConfirmPassword { get; set; }

public List<Role> Roles { get => Db.Roles.ToList(); }

#endregion

#region Конструктор

public EmployeeAddViewModel(KinderGartenDbContext db, AuthenticationService authService, MessageService messageService)

{

Db = db;

AuthService = authService;

MessageService = messageService;

Messenger.Default.Register<NotificationMessage<Employee>>(this, message =>

{

if (message.Sender is EmployeesViewModel)

{

Title = "Новый сотрудник";

Employee = new Employee

{

Document = new Document(),

Person = new Person()

};

AddVisibility = Visibility.Visible;

ChangeVisibility = Visibility.Collapsed;

}

});

}

#endregion

#region Команды

// Команда добавить

public ICommand AddCommand => new RelayCommand<Window>(async (obj) =>

{

if (Db.Users.FirstOrDefault(x => x.Login == Login) is null)

if (Password == ConfirmPassword)

{

try

{

Employee.User = AuthService.Reg(Login, Password);

await Db.Employees.AddAsync(Employee);

await Db.SaveChangesAsync();

obj.DialogResult = true;

Login = string.Empty;

Password = string.Empty;

ConfirmPassword = string.Empty;

Employee = null;

obj.Close();

MessageService.Message("Success", "Сотрудник добавлен!");

Messenger.Default.Send("UpdateEmployees");

}

catch

{

MessageService.Message("Error", "Заполните обязательные поля!");

}

}

else

MessageService.Message("Error", "Пароли не совпадают!");

else

MessageService.Message("Error", "Пользователь с таким логином уже существует!");

});

public ICommand PhotoChangeCommand => new RelayCommand(() =>

{

OpenFileDialog dlg = new OpenFileDialog();

ImageConverter imageConverter = new ImageConverter();

dlg.InitialDirectory = "";

dlg.Filter = "Image files (\*.jpg,\*.png,\*.bmp)|\*.jpg;\*.png;\*.bmp";

if (dlg.ShowDialog() == true)

Employee.Person.Image = (byte[])imageConverter.ConvertTo(System.Drawing.Image.FromFile(dlg.FileName), typeof(byte[]));

});

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels.Dialog

{

public class ParentChangeViewModel : DialogViewModel

{

#region Свойства

public bool AddVisibility { get; set; }

public bool ChangeVisibility { get; set; }

public Parent Parent { get; set; }

public List<ParentRole> ParentRoles { get; set; }

public int ChildrenId { get; set; }

public string Title { get; set; }

#endregion

#region Конструктор

public ParentChangeViewModel(KinderGartenDbContext db, MessageService messageService)

{

Db = db;

MessageService = messageService;

ParentRoles = Db.ParentRoles.ToList();

Messenger.Default.Register<NotificationMessage<Parent>>(this, message =>

{

if (message.Sender is ParentsViewModel)

{

if (message.Notification == "Change")

{

Parent = message.Content;

Title = "Редактирование родителя";

AddVisibility = false;

ChangeVisibility = true;

}

else

{

Title = "Новый родитель";

Parent = new Parent()

{

Person = new Person()

};

ChildrenId = int.Parse(message.Notification);

AddVisibility = true;

ChangeVisibility = false;

}

}

});

Messenger.Default.Register<string>(this, message =>

{

if (message == "UpdateParentRoles")

ParentRoles = Db.ParentRoles.ToList();

});

}

#endregion

#region Команды

// Команда добавить

public ICommand AddCommand => new RelayCommand<Window>((obj) =>

{

Parent.Children = Db.Childrens.FirstOrDefault(x => x.Id == ChildrenId);

Db.Parents.Add(Parent);

Db.SaveChanges();

obj.DialogResult = true;

Parent = null;

obj.Close();

MessageService.Message("Success", "Родитель добавлен");

});

// Команда изменить

public ICommand ChangeCommand => new RelayCommand<Window>(async (obj) =>

{

await Db.SaveChangesAsync();

obj.DialogResult = true;

Parent = null;

obj.Close();

MessageService.Message("Success", "Данные изменены");

});

// Команда удалить

public ICommand RemoveCommand => new RelayCommand<Window>(async (obj) =>

{

Db.Parents.Remove(Parent);

await Db.SaveChangesAsync();

obj.DialogResult = true;

Parent = null;

obj.Close();

MessageService.Message("Success", "Родитель удален");

});

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using Microsoft.EntityFrameworkCore;

using PropertyChanged;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels.Dialog

{

public class SalaryChangeViewModel : DialogViewModel

{

#region Свойства

public List<PaymentMethod> PaymentMethods { get; set; }

public PaymentMethod SelectedPaymentMethod { get; set; }

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

[AlsoNotifyFor("Amount")]

public Employee SelectedEmployee { get; set; }

private int Visits

{

get

{

if (SelectedEmployee != null)

{

return Db.Visits.Where(x => x.Lesson.Employee == SelectedEmployee &&

x.Date.Month == DateTime.Now.Month &&

x.Date.Year == DateTime.Now.Year && x.Lesson.Tariff == null).Count();

}

return 0;

}

}

public string Title { get; set; } = "Выплата заработной платы";

[AlsoNotifyFor("Amount")]

public decimal Premial { get; set; }

public decimal Amount

{

get

{

if (SelectedEmployee != null)

{

var Bid = SelectedEmployee.Salary;

switch (Convert.ToByte(Type))

{

case 1:

Bid += SelectedEmployee.HourSalary \* Visits;

break;

case 2:

Bid += (SelectedEmployee.HourSalary \* 1.5m) \* Visits;

break;

case 3:

Bid += (SelectedEmployee.HourSalary \* 2) \* Visits;

break;

}

var visits = Db.Visits.Include(x => x.Lesson).ThenInclude(x => x.Tariff).Where(x => x.Lesson.Employee == SelectedEmployee &&

x.Date.Month == DateTime.Now.Month &&

x.Date.Year == DateTime.Now.Year && x.Lesson.Tariff != null).ToList();

if (visits != null)

foreach (Visit i in visits)

{

Bid += i.Lesson.Tariff.Amount - (i.Lesson.Tariff.Amount \* 0.05m);

}

return Bid + Premial;

};

return 0;

}

}

[AlsoNotifyFor("Amount")]

public byte Type { get; set; }

#endregion

#region Конструктор

public SalaryChangeViewModel(KinderGartenDbContext db, MessageService messageService)

{

Db = db;

MessageService = messageService;

Employees = Db.Employees.Include(x => x.Person).ToList();

PaymentMethods = Db.PaymentMethods.ToList();

}

#endregion

#region Команды

public ICommand BidChangeCommand => new RelayCommand<string>((obj) => Type = Convert.ToByte(obj));

// Команда добавить

public ICommand AddCommand => new RelayCommand<Window>(async (obj) =>

{

try

{

if (await Db.Transactions.FirstOrDefaultAsync(x => x.Person == SelectedEmployee.Person && x.DateProcess.Month == DateTime.Now.Month && x.Type == false) is null)

{

await Db.Transactions.AddAsync(new Transaction

{

Amount = Amount,

Comment = "Зарплата",

DateProcess = DateTime.Now,

PaymentMethod = SelectedPaymentMethod,

Type = false,

Person = SelectedEmployee.Person

});

await Db.SaveChangesAsync();

MessageService.Message("Success", $"Заработная плата {SelectedEmployee.Person.Lastname} {SelectedEmployee.Person.Firstname[0]}. {SelectedEmployee.Person.Firstname[1]}. в сумме {Amount} руб. выплачена!");

obj.DialogResult = true;

obj.Close();

}

else

{

MessageService.Message("Error", "Ошибка при выплате заработной платы!");

}

}

catch

{

MessageService.Message("Error", "Ошибка при выплате заработной платы!");

}

});

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels.Dialog

{

public class SheduleChangeViewModel : DialogViewModel

{

#region Свойства

public Visibility AddVisibility { get; set; }

public Visibility ChangeVisibility { get; set; }

public Visibility SubscriptionsVisibility => DopLesson ? Visibility.Visible : Visibility.Collapsed;

public Visibility EmployeeVisibility => !DopLesson ? Visibility.Visible : Visibility.Collapsed;

public bool DopLesson { get; set; }

public List<Group> Groups { get; set; }

public List<Room> Rooms { get; set; }

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

public List<SubscriptionTemplate> Subscriptions { get; set; }

public Lesson Lesson { get; set; }

public string Title { get; set; }

#endregion

#region Конструктор

public SheduleChangeViewModel(KinderGartenDbContext db, MessageService messageService)

{

Db = db;

MessageService = messageService;

Groups = Db.Groups.ToList();

Rooms = Db.Rooms.ToList();

Subscriptions = Db.Subscriptions.Include(x => x.Employee).ToList();

Employees = Db.Employees.Include(x => x.Person).ToList();

Messenger.Default.Register<string>(this, message =>

{

switch (message)

{

case "UpdateGroups":

Groups = Db.Groups.ToList();

break;

case "UpdateRooms":

Rooms = Db.Rooms.ToList();

break;

case "UpdateSubscriptions":

Subscriptions = Db.Subscriptions.Include(x => x.Employee).ToList();

break;

case "UpdateEmployees":

Employees = Db.Employees.Include(x => x.Person).ToList();

break;

}

});

Messenger.Default.Register<NotificationMessage<Lesson>>(this, message =>

{

if (message.Notification == "Add")

{

Title = "Новое занятие";

Lesson = new Lesson();

AddVisibility = Visibility.Visible;

ChangeVisibility = Visibility.Collapsed;

}

else if (message.Notification == "Change")

{

Lesson = message.Content;

Title = "Редактирование занятия";

DopLesson = message.Content.Tariff != null;

AddVisibility = Visibility.Collapsed;

ChangeVisibility = Visibility.Visible;

}

});

}

#endregion

#region Команды

// Команда добавить

public ICommand AddCommand => new RelayCommand<Window>(async (obj) =>

{

if (DopLesson)

{

Lesson.Employee = Lesson.Tariff.Employee;

Lesson.Name = Lesson.Tariff.Name;

Lesson.DateStart = Lesson.Tariff.Start;

Lesson.DateEnd = Lesson.Tariff.End;

}

if ((DopLesson && (Db.Lessons.Where(x => x.Tariff == Lesson.Tariff && x.Group == Lesson.Group).Count() < Lesson.Tariff.HoursCount) || !DopLesson) &&

Db.Lessons.FirstOrDefault(x => x.DayOfWeek == Lesson.DayOfWeek && x.Group == Lesson.Group && x.LessonNumber == Lesson.LessonNumber &&

((Lesson.DateStart >= x.DateStart && Lesson.DateStart <= x.DateEnd) ||

(x.DateStart >= Lesson.DateStart && x.DateStart <= Lesson.DateEnd)) &&

((Lesson.DateEnd >= x.DateStart && Lesson.DateEnd <= x.DateEnd) ||

(x.DateEnd >= Lesson.DateStart && x.DateEnd <= Lesson.DateEnd))) == null)

{

await Db.Lessons.AddAsync(Lesson);

await Db.SaveChangesAsync();

obj.DialogResult = true;

Lesson = null;

obj.Close();

MessageService.Message("Success", "Занятие добавлено!");

}

else

{

MessageService.Message("Error", "Занятие пересекается с другим!");

}

});

// Команда изменить

public ICommand ChangeCommand => new RelayCommand<Window>(async (obj) =>

{

if (DopLesson)

{

Lesson.Employee = Lesson.Tariff.Employee;

Lesson.Name = Lesson.Tariff.Name;

}

await Db.SaveChangesAsync();

obj.DialogResult = true;

Lesson = null;

obj.Close();

MessageService.Message("Success", "Занятие успешно изменено!");

});

// Команда удалить

public ICommand RemoveCommand => new RelayCommand<Window>(async (obj) =>

{

Db.Lessons.Remove(Lesson);

await Db.SaveChangesAsync();

obj.DialogResult = true;

Lesson = null;

obj.Close();

MessageService.Message("Success", "Занятие удалено!");

});

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels.Dialog

{

public class SingleDialogViewModel : DialogViewModel

{

#region Свойства

public string Name { get; set; }

public string Text { get; set; }

public string Title { get; set; }

public Visibility TextVisibility { get; set; }

public Visibility ComboVisibility { get => TextVisibility == Visibility.Collapsed ? Visibility.Visible : Visibility.Collapsed; }

public List<Children> Childrens { get; set; }

public Children Children { get; set; }

private object Target { get; set; }

#endregion

#region Команды

public ICommand OkCommand => new RelayCommand<System.Windows.Window>((window) =>

{

window.DialogResult = true;

if (Target is ChildrensViewModel)

{

if (Children != null)

Messenger.Default.Send(new NotificationMessage<Children>(this, Target, Children, Text));

else

MessageService.Message("Error", "Выберите ученика");

}

else

Messenger.Default.Send(new NotificationMessage<string>(this, Target, Title, Text));

});

#endregion

#region Конструктор

public SingleDialogViewModel(KinderGartenDbContext db, MessageService messageService)

{

Db = db;

MessageService = messageService;

Messenger.Default.Register<NotificationMessage<string>>(this, message =>

{

if (message.Sender is ChildrensViewModel)

{

Title = "Добавление ученика";

Childrens = Db.Childrens.ToList();

Target = message.Sender;

TextVisibility = Visibility.Collapsed;

}

else

{

Target = message.Sender;

Title = message.Content;

Name = message.Notification;

}

});

}

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels.Dialog

{

public class SubscriptionChangeViewModel : DialogViewModel

{

#region Свойства

public Visibility AddVisibility { get; set; }

public Visibility ChangeVisibility { get; set; }

public List<SubscriptionTemplate> Subscriptions { get; set; }

public List<PaymentMethod> PaymentMethods { get; set; }

public PaymentMethod SelectedMethod { get; set; }

public SubscriptionTemplate SelectedItem { get; set; }

public Children Children { get; set; }

public string Title { get; set; }

public decimal Price { get => SelectedItem != null ? SelectedItem.Amount \* SelectedItem.HoursCount : 0; }

#endregion

#region Конструктор

public SubscriptionChangeViewModel(KinderGartenDbContext db, MessageService messageService)

{

Db = db;

MessageService = messageService;

Subscriptions = Db.Subscriptions.ToList();

PaymentMethods = Db.PaymentMethods.ToList();

Messenger.Default.Register<string>(this, message =>

{

switch (message)

{

case "UpdateSubscriptions":

Subscriptions = Db.Subscriptions.ToList();

break;

case "UpdatePaymentMethods":

PaymentMethods = Db.PaymentMethods.ToList();

break;

}

});

Messenger.Default.Register<NotificationMessage<Children>>(this, message =>

{

if (message.Notification == "Add")

{

Title = "Оформление абонемента";

Children = message.Content;

AddVisibility = Visibility.Visible;

ChangeVisibility = Visibility.Collapsed;

}

});

}

#endregion

#region Команды

// Команда добавить

public ICommand AddCommand => new RelayCommand<Window>(async (obj) =>

{

Children.ChildrenSubscriptions.Add(new ChildrenSubscription { Children = Children, Subscription = SelectedItem });

Db.Transactions.Add(new Transaction

{

Amount = Price,

Comment = "Покупка абонемента",

DateProcess = DateTime.Now,

Payment = true,

Person = Children.Person,

Type = true,

PaymentMethod = SelectedMethod

});

Db.Childrens.Update(Children);

await Db.SaveChangesAsync();

obj.DialogResult = true;

obj.Close();

MessageService.Message("Success", "Абонемент оформлен!");

});

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels.Dialog

{

public class TemplateChangeViewModel : DialogViewModel

{

#region Свойства

public Visibility AddVisibility { get; set; }

public Visibility ChangeVisibility { get; set; }

public SubscriptionTemplate Subscription { get; set; }

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

public string Title { get; set; }

#endregion

#region Конструктор

public TemplateChangeViewModel(KinderGartenDbContext db, MessageService messageService)

{

Db = db;

MessageService = messageService;

Employees = Db.Employees.Include(x => x.Person).ToList();

Messenger.Default.Register<string>(this, message =>

{

if (message == "UpdateEmployees")

{

Employees = Db.Employees.Include(x => x.Person).ToList();

}

});

Messenger.Default.Register<NotificationMessage<SubscriptionTemplate>>(this, message =>

{

if (message.Notification == "Add")

{

Title = "Новый шаблон";

Subscription = new SubscriptionTemplate();

AddVisibility = Visibility.Visible;

ChangeVisibility = Visibility.Collapsed;

}

else if (message.Notification == "Change")

{

Subscription = message.Content;

Title = "Редактирование шаблона";

AddVisibility = Visibility.Collapsed;

ChangeVisibility = Visibility.Visible;

}

});

}

#endregion

#region Команды

// Команда добавить

public ICommand AddCommand => new RelayCommand<Window>((obj) =>

{

Db.Subscriptions.Add(Subscription);

Db.SaveChanges();

obj.DialogResult = true;

Subscription = null;

obj.Close();

MessageService.Message("Success", "Шаблон добавлен!");

Messenger.Default.Send("UpdateSubscriptions");

});

// Команда изменить

public ICommand ChangeCommand => new RelayCommand<Window>((obj) =>

{

Db.SaveChanges();

obj.DialogResult = true;

Subscription = null;

obj.Close();

MessageService.Message("Success", "Данные успешно изменены!");

Messenger.Default.Send("UpdateSubscriptions");

});

// Команда удалить

public ICommand RemoveCommand => new RelayCommand<Window>((obj) =>

{

Db.Subscriptions.Remove(Subscription);

Db.SaveChanges();

obj.DialogResult = true;

Subscription = null;

obj.Close();

MessageService.Message("Success", "Шаблон удален!");

Messenger.Default.Send("UpdateSubscriptions");

});

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using KinderGartenWpf.ViewModels.Base;

using KinderGartenWpf.Views;

using System.Windows.Controls;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels

{

public class AnalyticsViewModel : ViewModelBase

{

#region Свойства

public Page TabView { get; set; } = new FinanceReportView();

#endregion

#region Команды

public ICommand ChangeTabCommand => new RelayCommand<string>((numb) =>

{

switch (int.Parse(numb))

{

case 1:

TabView = new FinanceReportView();

break;

case 3:

TabView = new VisitsReportView();

break;

}

});

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.ViewModels.Base;

using KinderGartenWpf.Views;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels

{

public class CabinetViewModel : ViewModelBase

{

#region Свойства

public Page TabView { get; set; } = new EmployeeChangeView();

public Employee User { get; set; }

public bool ChangeBtnVisibility { get => App.RoleId == 1 || App.UserId == User?.Id; }

#endregion

#region Команды

public ICommand ChangeTabCommand => new RelayCommand<string>((numb) =>

{

switch (int.Parse(numb))

{

case 6:

TabView = new EmployeeChangeView();

break;

case 5:

TabView = new SuccessLessonsView();

break;

case 4:

TabView = new BidsView();

break;

default:

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

break;

}

Messenger.Default.Send(new NotificationMessage<Employee>(User, null));

});

#endregion

#region Конструктор

public CabinetViewModel(KinderGartenDbContext db)

{

Db = db;

User = Db.Employees.Find(App.UserId);

Messenger.Default.Register<NotificationMessage<Employee>>(this, message =>

{

User = message.Content;

});

}

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using KinderGartenWpf.ViewModels.Dialog;

using KinderGartenWpf.Views.ChangeViews;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels

{

public class ChildrensViewModel : ViewModelBase

{

#region Свойства

public bool IsChildrenEmpty { get => Childrens?.Count() == 0; }

public bool ChangeBtnVisibility { get => SelectedItem != null && App.RoleId == 1; }

public bool AddGroupBtnVisibility { get => SelectedGroup != null && App.RoleId == 1; }

public bool AddBtnVisibility { get => App.RoleId == 1; }

public List<Children> Childrens { get; set; }

public List<Group> Groups { get; set; }

public Group SelectedGroup { get; set; }

public string Search { get; set; }

public Children SelectedItem { get; set; }

#endregion

#region Конструктор

public ChildrensViewModel(KinderGartenDbContext db, MessageService messageService, NavigationService navigationService)

{

Db = db;

MessageService = messageService;

NavService = navigationService;

Groups = Db.Groups.ToList();

Messenger.Default.Register<NotificationMessage<Group>>(this, message =>

{

if (message.Sender is GroupsViewModel)

{

if (message.Content != null)

SelectedGroup = Groups.FirstOrDefault(x => x == message.Content);

}

});

Messenger.Default.Register<string>(this, message =>

{

if (message == "UpdateGroups")

Groups = Db.Groups.ToList();

});

Messenger.Default.Register<NotificationMessage<Children>>(this, message =>

{

if (message.Sender is SingleDialogViewModel)

{

Db.ChildrenGroups.Add(new ChildrenGroup { Children = message.Content, Group = SelectedGroup });

Db.SaveChanges();

MessageService.Message("Success", "Ребенок добавлен в группу!");

Update();

}

});

Update();

MessageService.Message("Info", "Чтобы посмотреть подробную информацию, дважды щелкните по ученику");

}

#endregion

#region Команды

//Команда поиска

public ICommand SearchCommand => new RelayCommand(() => Update());

//Команда добавить

public ICommand ChildrenAdd => new RelayCommand(() =>

{

var Dialog = new ChildrenChangeView();

Messenger.Default.Send(new NotificationMessage<Children>(this, null, "Add"));

Dialog.ShowDialog();

if (Dialog.DialogResult == true)

Update();

});

//Команда для просмотра подробной информации

public ICommand GridDoubleCommand => new RelayCommand(() =>

{

NavService.Navigate("Children");

Messenger.Default.Send(new NotificationMessage<Children>(this, SelectedItem, "Details"));

});

// Команда изменить

public ICommand ChildrenChange => new RelayCommand(() =>

{

var Dialog = new ChildrenChangeView();

Messenger.Default.Send(new NotificationMessage<Children>(this, SelectedItem, "Change"));

Dialog.ShowDialog();

if (Dialog.DialogResult == true)

Update();

});

public ICommand AddGroupChildren => new RelayCommand(() =>

{

var dialog = new DialogView();

Messenger.Default.Send(new NotificationMessage<string>(this, dialog.DataContext, "Добавление участника", "Название"));

dialog.ShowDialog();

});

#endregion

#region Методы

/// <summary>

/// Обновление таблицы учеников

/// </summary>

void Update()

{

Task.Run(() =>

{

if (string.IsNullOrWhiteSpace(Search))

Childrens = Db.Childrens

.Include(x => x.Person)

.Include(x => x.ChildrenSubscriptions)

.ThenInclude(x => x.Subscription)

.Include(x => x.ChildrenGroups)

.ThenInclude(x => x.Group)

.ToList();

else

Childrens = Db.Childrens.Include(x => x.Person).Include(x => x.ChildrenSubscriptions).ThenInclude(x => x.Subscription).Include(x => x.ChildrenGroups).ThenInclude(x => x.Group)

.Where(x => x.Person.Firstname.Contains(Search)

|| x.Person.Lastname.Contains(Search)

|| x.Person.Middlename.Contains(Search)

|| x.Person.Phone.Contains(Search))

.ToList();

if (SelectedGroup != null)

Childrens = Childrens.Where(x => x.ChildrenGroups.FirstOrDefault(x => x.Group == SelectedGroup) != null).ToList();

});

}

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.ViewModels.Base;

using KinderGartenWpf.Views;

using KinderGartenWpf.Views.ChangeViews;

using System.Windows.Controls;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels

{

public class ChildrenViewModel : ViewModelBase

{

#region Свойства

public Page TabView { get; set; } = new SubscriptionView();

public Children Children { get; set; }

public string ChildrenName { get; set; }

public string ChildrenGroups { get; set; }

#endregion

#region Конструктор

public ChildrenViewModel()

{

Messenger.Default.Register<NotificationMessage<Children>>(this, message =>

{

Children = message.Content;

if (Children != null)

ChildrenName = $"{Children.Person.Lastname} {Children.Person.Firstname} {Children.Person.Middlename}";

});

}

#endregion

#region Команды

public ICommand ChangeTabCommand => new RelayCommand<string>((numb) =>

{

switch (int.Parse(numb))

{

case 1:

TabView = new SubscriptionView();

Messenger.Default.Send(new NotificationMessage<Children>(this, Children, null));

break;

case 2:

TabView = new ScheduleView();

Messenger.Default.Send(new NotificationMessage<Children>(this, Children, null));

break;

case 3:

TabView = new ParentsView();

Messenger.Default.Send(new NotificationMessage<Children>(this, Children, null));

break;

}

});

public ICommand ChangeCommand => new RelayCommand(() =>

{

var Dialog = new ChildrenChangeView();

Messenger.Default.Send(new NotificationMessage<Children>(this, Children, "Change"));

Dialog.ShowDialog();

});

#endregion

#region Методы

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using Microsoft.EntityFrameworkCore;

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels

{

public class EmployeeChangeViewModel : ViewModelBase

{

#region Свойства

private readonly AuthenticationService AuthService;

public Employee Employee { get; set; }

public List<Group> Groups { get; set; }

public List<Role> Roles { get; set; }

public string Password { get; set; }

public string NewPassword { get; set; }

public string ConfirmPassword { get; set; }

public bool PropertyEnable { get => App.RoleId == 1; }

#endregion

#region Конструктор

public EmployeeChangeViewModel(KinderGartenDbContext db, AuthenticationService authService, MessageService messageService)

{

Db = db;

AuthService = authService;

MessageService = messageService;

Groups = Db.Groups.ToList();

Roles = Db.Roles.ToList();

Employee = Db.Employees.Find(App.UserId);

Messenger.Default.Register<string>(this, message =>

{

switch (message)

{

case "UpdateGroups":

Groups = Db.Groups.ToList();

break;

case "UpdateRoles":

Roles = Db.Roles.ToList();

break;

}

});

Messenger.Default.Register<NotificationMessage<Employee>>(this, message =>

{

if (message.Sender is ShellViewModel || message.Sender is EmployeesViewModel)

{

Employee = message.Content;

}

});

}

#endregion

#region Команды

// Команда изменить основные данные

public ICommand ChangeMainCommand => new RelayCommand(async () =>

{

try

{

var user = await Db.Employees.FirstOrDefaultAsync(x => x.Id == Employee.Id);

user = Employee;

await Db.SaveChangesAsync();

UpdateUser();

MessageService.Message("Success", "Общие данные сохранены!");

Messenger.Default.Send("UpdateEmployees");

}

});

// Команда изменить документ педагога

public ICommand ChangeDocumentCommand => new RelayCommand(async () =>

{

try

{

var document = await Db.Documents.FirstOrDefaultAsync(x => x.Id == Employee.Document.Id);

document = Employee.Document;

await Db.SaveChangesAsync();

UpdateUser();

MessageService.Message("Success", "Документ сохранен!");

Messenger.Default.Send("UpdateEmployees");

}

catch

{

MessageService.Message("Error", "Заполните обязательные поля!");

}

});

// Команда изменить пароль

public ICommand ChangePasswordCommand => new RelayCommand(async () =>

{

if (NewPassword == ConfirmPassword)

{

if (await AuthService.UpdatePassword(Employee.User.Id, Password, NewPassword))

{

UpdateUser();

MessageService.Message("Success", "Пароль сохранен!");

Messenger.Default.Send("UpdateEmployees");

}

else

MessageService.Message("Error", "Неверно введен старый пароль!");

}

else

MessageService.Message("Error", "Пароли не совпадают");

});

public ICommand PhotoChangeCommand => new RelayCommand(() =>

{

OpenFileDialog dlg = new OpenFileDialog();

ImageConverter imageConverter = new ImageConverter();

dlg.InitialDirectory = "";

dlg.Filter = "Image files (\*.jpg,\*.png,\*.bmp)|\*.jpg;\*.png;\*.bmp";

if (dlg.ShowDialog() == true)

Employee.Person.Image = (byte[])imageConverter.ConvertTo(System.Drawing.Image.FromFile(dlg.FileName), typeof(byte[]));

});

#endregion

#region Методы

/// <summary>

/// Сообщение об изменении данных текущего пользователя

/// </summary>

void UpdateUser()

{

Messenger.Default.Send(new NotificationMessage<Employee>(this, Db.Employees

.Include(x => x.Role)

.Include(x => x.Document)

.Include(x => x.User)

.FirstOrDefault(x => x.Id == Employee.Id), "Update"));

Password = String.Empty;

NewPassword = String.Empty;

ConfirmPassword = String.Empty;

}

#endregion

}

}

using GalaSoft.MvvmLight.Command;

using GalaSoft.MvvmLight.Messaging;

using KinderGartenWpf.Models;

using KinderGartenWpf.Models.Objects;

using KinderGartenWpf.Services;

using KinderGartenWpf.ViewModels.Base;

using KinderGartenWpf.Views;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Input;

namespace KinderGartenWpf.ViewModels

{

public class EmployeesViewModel : ViewModelBase

{

#region Свойства

public bool GridEmptyVisibility { get => Employees.Count() == 0; }

public bool RemoveBtnVisibility { get => SelectedItem != null && App.RoleId == 1; }

public bool AddBtnVisibility { get => App.RoleId == 1; }

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

public string Search { get; set; }

public Employee SelectedItem { get; set; }

#endregion

#region Конструктор

public EmployeesViewModel(KinderGartenDbContext db, NavigationService navService, MessageService messageService)

{

Db = db;

NavService = navService;

MessageService = messageService;

Update();

MessageService.Message("Info", "Чтобы посмотреть подробную информацию, дважды щелкните по педагогу");

Messenger.Default.Register<NotificationMessage<int>>(this, message =>

{

if (message.Sender is ShellViewModel)

{

UserRole = message.Content;

}

});

}

#endregion

#region Команды

//Команда поиска

public ICommand SearchCommand => new RelayCommand(() => Update());

//Команда добавить

public ICommand AddCommand => new RelayCommand(() =>

{

var Dialog = new EmployeeAddView();

Messenger.Default.Send(new NotificationMessage<Employee>(this, null, "Add"));

Dialog.ShowDialog();

if (Dialog.DialogResult == true)

Update();

});

//Команда для просмотра подробной информации

public ICommand GridDoubleCommand => new RelayCommand(() =>

{

NavService.Navigate("Cabinet");

Messenger.Default.Send(new NotificationMessage<Employee>(this, SelectedItem, "Detail"));

});

// Команда изменить

public ICommand RemoveCommand => new RelayCommand(async () =>

{

Db.Employees.Remove(SelectedItem);

await Db.SaveChangesAsync();

MessageService.Message("Success", "Сотрудник удален!");

Update();

Messenger.Default.Send("UpdateEmployees");

});

#endregion

#region Методы

/// <summary>

/// Обновляет таблицу педагогов

/// </summary>

void Update()

{

if (string.IsNullOrWhiteSpace(Search))

Employees = Db.Employees.Include(x => x.Role).Include(x => x.Person).Include(x => x.Document).Include(x => x.User).ToList();

else

{

Employees = Db.Employees.Include(x => x.Role).Include(x => x.Person).Include(x => x.Document).Include(x => x.User)

.Where(x => (x.Person.Firstname + x.Person.Lastname + x.Person.Middlename + x.Person.Phone).Contains(Search)).ToList();

}

}

#endregion

}

}

# ПРИЛОЖЕНИЕ В

Список файлов на диске:

* KinderGarten (Информационная система детского сада);
* ВКР.docx;
* Доклад.docx;