МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра Информационные системы и технологии

Специальность Информационные системы и технологии

Направление специальности 1-40 05 01 Информационные системы и технологии

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТ:**

по дисциплине «Объектно-ориентированное программирование»

Тема Программное средство «Компьютерный клуб»

Исполнитель

студент 3 курса 2 группы Лемешевский В.О.

(Ф.И.О.)

Руководитель работы ассистент Мущук А.Н.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Мущук А.Н.

(подпись)

Минск 2024

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc185277890)

[1.1. Анализ прототипов 4](#_Toc185277891)

[1.2. Требования к проекту 6](#_Toc185277892)

[1.3. Выводы по разделу 6](#_Toc185277893)

[2. Анализ требований к программному средству и разработка функциональных требований 7](#_Toc185277894)

[2.1. Описание средств разработки 7](#_Toc185277895)

[2.2. Спецификация функциональных требований к программному средству 9](#_Toc185277896)

[2.3. Спецификация функциональных требований 9](#_Toc185277897)

[2.4. Выводы по разделу: 10](#_Toc185277898)

[3. Проектирование программного средства 11](#_Toc185277899)

[3.1. Общая структура 11](#_Toc185277900)

[3.2. Взаимоотношения между классами 12](#_Toc185277901)

[3.3. Модель базы данных 12](#_Toc185277902)

[3.5. Проектирование последовательностей проекта 13](#_Toc185277903)

[4. Реализация программного средства 15](#_Toc185277904)

[4.1. Основные классы программного средства 15](#_Toc185277905)

[4.2. Описание классов и методов программного средства 15](#_Toc185277906)

[4.2.1. Выполнение входа 15](#_Toc185277907)

[4.2.2. Управление пользователями 16](#_Toc185277908)

[4.2.3. Управление тарифами 16](#_Toc185277909)

[4.2.4. Управление компьютерами 17](#_Toc185277910)

[5. Тестирование, проверка работоспособности и анализ полученных результатов 19](#_Toc185277911)

[5.1. Тестирование авторизации 19](#_Toc185277912)

[5.2. Тестирование создания тарифа 20](#_Toc185277913)

[5.3. Тестирование удаления тарифа 21](#_Toc185277914)

[5.4. Тестирование управления пользователями 22](#_Toc185277915)

[5.5. Тестирование бронирования компьютера и выбора тарифа 24](#_Toc185277916)

[5.6 Выводы по разделу: 25](#_Toc185277917)

[6. Руководство по установке и использованию 26](#_Toc185277918)

[6.1. Выводы по разделу: 34](#_Toc185277919)

[Заключение 35](#_Toc185277920)

[Список литературы 36](#_Toc185277921)

[ПРИЛОЖЕНИЕ А 37](#_Toc185277922)

[ПРИЛОЖЕНИЕ Б 38](#_Toc185277923)

[ПРИЛОЖЕНИЕ В 39](#_Toc185277924)

[ПРИЛОЖЕНИЕ Г 40](#_Toc185277925)

## ВВЕДЕНИЕ

В современном мире компьютерные клубы являются популярным местом среди любителей видеоигр, профессиональных геймеров, студентов и профессионалов в области информационных технологий. Управление такими заведениями требует эффективной системы контроля доступа, учета времени использования оборудования, а также мониторинга работы компьютеров и активности пользователей.

В рамках данного курсового проекта предлагается разработка программного средства для управления компьютерным клубом с использованием языка программирования C# и технологии Windows Presentation Foundation (WPF). Данное программное обеспечение будет предоставлять администратору клуба инструменты для эффективного контроля и управления всеми аспектами его работы и удобный интерфейс с функционалом для пользователей.

Целью данного проекта является создание интуитивно понятного и функционального приложения, способного упростить рутинные задачи администрирования компьютерного клуба и повысить эффективность и удобство для посетителей.

В ходе разработки будут рассмотрены основные аспекты проектирования и реализации программного средства, включая архитектуру приложения, пользовательский интерфейс а также интеграцию с базой данных для хранения информации о пользователях, времени использования и других важных параметров.

В результате успешной реализации данного проекта ожидается создание удобного и мощного инструмента для бронирования и управления в компьютерном клубе, способного повысить качество обслуживания клиентов и оптимизировать рабочие процессы администрации.

**1**.**Аналитический обзор литературы и формирование требований**

## Анализ прототипов

Были проанализированы цели поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

* + 1. **Smartshell**

Smartshell [1] — cовременное программное обеспечение для управления компьютерным клубом. Программный комплекс включает всё необходимое для эффективного управления клубом: удобный дашборд с виджетами, магазин и склад, мониторинг оборудования, отчёты по сменам, подробные логи событий, клубные аккаунты, акции, скидки, промокоды, кассы, уведомления в Telegram и другое. Интерфейс представлен на рисунке 1.1.



Рисунок 1.1 – Интерфейс Smartshell

Явным недостатком является избыточность информации. Например, информация о комплектующих каждого отдельного компьютера не может быть полезна пользователю, и лишь усложняет взаимодействие с программой.

* + 1. AstrumSoft

Программа "AstrumSoft" [2] предназначена для автоматизации работы компьютерных клубов, интернет кафе. Учет оплаченного времени использования компьютеров, игровых приставок или любых других устройств, продажи дополнительных услуг (кофе, бутерброды). Компьютеры за которые не заплачено или вышло время блокируются. Увеличивает доход от работы клуба, следя за администраторами (предотвращает воровство).

Серьёзным недостатком данного программного обеспечения является устаревший, визуально перегруженный интерфейс. Кроме того доступ к функционалу осуществляется интуитивно непонятными способами. Это видно на рисунке 1.2.



Рисунок 1.2 – Интерфейс AstrumSoft

Кроме того, программное средство обеспечивает подробные отчеты, защищенные от изменений, полную поддержку Windows 11. Компьютеры клиентов находятся под надежным контролем.

* + 1. Lightshell

LightShell [3] представляет собой инновационное программное обеспечение, разработанное с учетом потребностей компьютерных клубов. Интерфейс Lightshell использует крайне вычурные цвета, что не является его плюсом. Это продемонстрированно на рисунке 1.3.

В основе LightShell лежит идея управления категориями клиентов. Это позволяет вам легко классифицировать ваших клиентов и предлагать им наиболее подходящие услуги. LightShell также предлагает уникальные возможности для управления пакетами услуг. Вы можете настраивать как сгораемые, так и не сгораемые пакеты услуг, чтобы удовлетворить потребности каждого клиента.

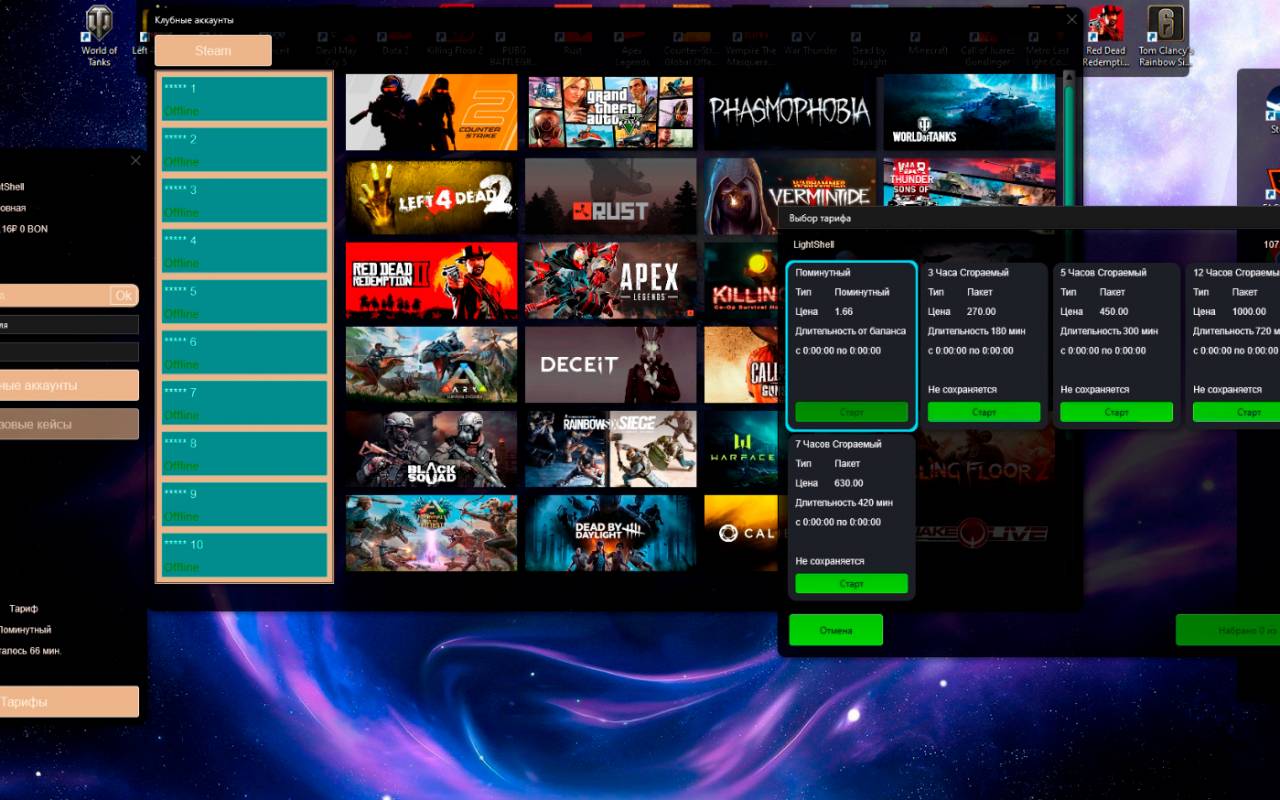


Рисунок 1.3 – Интерфейс Lightshell

Основным недостатком LightShell является отсутствие гибкого контроля над каждым компьютером в зале.

## Требования к проекту

Обзор вышеперечисленных известных аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и позволяет сформулировать список требований, предъявляемых к программному средству, разрабатываемому в данном курсовом проекте. Программное средство должно обеспечивать возможность выполнения перечисленных ниже функций:

* возможность создавать тарифы и компьютеры для ускорения обслуживания;
* возможность регулировать бронь каждого компьютера;
* предоставление удобного и красивого интерфейса для пользователей;

## 1.3. Выводы по разделу

В разделе были определены ключевой функционал для программного средства, самыми важными из которых являются:

* возможность создавать тарифы и компьютеры для ускорения обслуживания;
* возможность регулировать бронь каждого компьютера;
* предоставление удобного и красивого интерфейса для пользователей;

Так же программное средство должно целевую платформу Windows 10 и иметь дизайн в светлых тонах.

Таким образом, в данном разделе сформулированы конкретные требования к разрабатываемому программному средству, основанные на анализе существующих решений. Основное внимание уделено созданию продукта, который будет сочетать в себе функциональность, удобство.

## 2. Анализ требований к программному средству и разработка функциональных требований

# **Описание средств разработки**

При разработке приложения были использованы:

* интегрированная среда разработки Microsoft Visual Studio 2022;
* программная платформа .NET Core 8.0;
* язык программирования C#;
* расширяемый язык разметки XAML;
* технология WPF;
* технология ADO .NET;
* база данных MS SQL Server.

1. **Microsoft Visual Studio 2022**

Microsoft Visual Studio 2022 — это интегрированная среда разработки для написания, отладки и сборки кода, а также последующей публикации приложений. Данный продукт позволяет разрабатывать не только консольные, но и десктопные приложения, с использованием таких технологий, как WinForms или WPF.

1. **Программная платформа .NET Core 8.0**

Платформа .NET Core — это созданная Microsoft в 2016 году кроссплатформенная, с открытым исходным кодом технология, предназначенная для создания и выполнения современных приложений. Она поддерживает разработку различных типов приложений, включая веб-приложения, облачные сервисы, десктопные приложения и микросервисы, работая на операционных системах Windows, macOS и Linux.

Основой .NET Core является общеязыковая среда выполнения Common Language Runtime (CLR), которая обеспечивает выполнение кода, написанного на различных поддерживаемых языках программирования, таких как C#, F#, и Visual Basic. .NET Core предлагает модульную архитектуру, где разработчики могут выбирать и использовать только необходимые пакеты, что позволяет создавать более легкие и производительные приложения.

1. **Язык программирования C#**

В качестве языка программирования используется C# – основной язык разработки в .NET Framework. Язык объектно-ориентированный, имеет строгую статическую типизацию, поддерживает перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения. Используется как основной язык в технологии WPF.

1. **Технология WPF**

Для предоставления пользовательского интерфейса и разграничения дизайна и бизнес-логики используется технология Microsoft WPF – аналог WinForms, система для построения клиентских приложений Windows с возможностями взаимодействия с пользователем и графическая подсистема в составе .NET Framework, использующая язык разметки XAML.

1. **Расширяемый язык разметки XAML**

WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление. XAML представляет собой язык декларативного описания интерфейса, основанный на XML.

1. **Технология ADO .NET**

Для взаимодействия с базой данных в приложении WPF используется технология ADO.NET — это базовый уровень доступа к данным, который входит в состав .NET Framework и предоставляет функционал для работы с различными реляционными базами данных, такими как SQL Server, MySQL и Oracle. В отличие от Entity Framework, который предлагает высокий уровень абстракции, ADO.NET работает на более низком уровне, предоставляя разработчикам полный контроль над процессом взаимодействия с базой данных.

ADO.NET позволяет напрямую взаимодействовать с базой данных с использованием SQL-запросов и поддерживает следующие компоненты:

* Создание подключения к базе данных через классы SqlConnection.
* Выполнение SQL-запросов с помощью класса SqlCommand.
* Чтение данных (DataReader)
* Работа с наборами данных

1. **MS SQL Server**

Для организации баз данных MS SQL Server использует реляционную модель, которая предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта. Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент (например, внешняя программа) отправляет запрос 10 на языке SQL должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения. Основной используемый язык запросов — Transact-SQL — реализован на структурированном языке запросов (SQL) с расширениями.

# **Спецификация функциональных требований к программному средству**

Программное средство должно предоставлять следующие функциональные возможности:

Для пользователя:

* авторизация;
* регистрация;
* просмотр списка тарифов;
* просмотр списка компьютеров;
* бронирование компьютера на определенный день и время;
* выбор тарифов;
* просмотр справочной информации;

Для администратора:

* все функции пользователя;
* добавление новых клиентов;
* редактирование информации о клиентах;
* удаление клиентов;
* добавление новых компьютеров и тарифов;
* добавление новой брони;
* удаление старых броней;

# **Спецификация функциональных требований**

Для функциональности ПС необходимо создание базы данных для хранения информации приложения. Подробно база данных описано в следующем разделе.

В программном средстве необходимо реализовать авторизацию и регистрацию пользователей для доступа ко всем возможностям функциям приложения. Для авторизации входными параметрами являются логин и пароль пользователя, которые содержатся в базе данных. Введенные данные, успешно прошедшие валидацию, сверяются с данными в базе данных.

Пользователь может выбирать тариф, компьютер, время прибытия и забронировать его. Все сведения должны автоматически обновляться в базе данных.

Администратор обладает теми же возможностями, что и пользователь, а также обладает возможностями добавления, просмотра, редактирования и удаления информации о клиентах, компьютерах и тарифах.

## 2.4. Выводы по разделу:

В разделе были определены инструменты которые будут использоваться в программном средстве. К таким относятся:

* программная платформа .NET Core 8.0;
* язык программирования C#;
* расширяемый язык разметки XAML;
* технология WPF;
* технология ADO .NET;
* база данных MS SQL Server.

Таким образом, программное средство построено на современных и эффективных технологиях разработки, которые обеспечивают высокую производительность, стабильность и масштабируемость. Это позволит эффективно решать задачи как для пользователей, так и для администраторов, включая бронирование, управление клиентами, тарифами и компьютерами.

## 3. Проектирование программного средства

# **Общая структура**

Программное средство использует архитектуру MVVM. Программное средство имеет следующую структуру, представленную на рисунке 3.1.



Рисунок 3.1 – Структура проекта

Описание структуры основных папок и файлов проекта представлено в таблице 3.1.

Таблица 3.1 – Описание структуры папок и файлов проекта

|  |  |
| --- | --- |
| Имя файла | Содержание |
| App.config | Файл с параметрами проекта. |
| App.xaml | Файл с общими ресурсами проекта |
| Папка Resources | Папка, содержащая все исходники ресурсов приложения. |
| Папка Helper | Папка, содержащая классы, необходимые в работе других классов. |
| Папка Model | Папка, содержащая классы, которые являются сущностями базы данных. |
| Папка View | Папка, содержащая файлы разметок и классы соответствующих страниц. |
| Папка ViewModel | Папка, содержащая классы являющимися контекстами для страниц. |

# **Взаимоотношения между классами**

Для визуализации взаимосвязей между классами используется диаграмма UML – графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

Для представления внутренней структуры программы в виде классов и связей между ними используется диаграмма классов. Приложение спроектировано таким образом, что каждый класс выполняет свои функции и практически не зависит от других. Диаграмма классов представлена в Приложении А.

# **Модель базы данных**

Для реализации поставленной задачи была создана база данных ComputerClub. Для ее создания использовалась система управления реляционными базами данных MSSQL Server. База данных состоит из таблиц, представленных на рисунке 3.3.

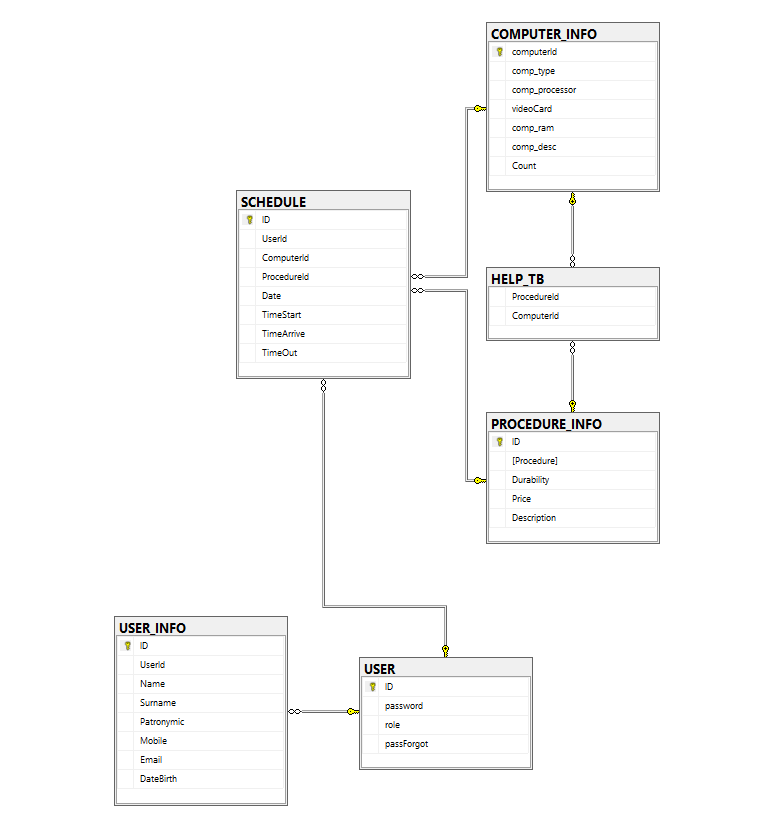


Рисунок 3.3 – Логическая модель базы данных ComputerClub

В базе данных находится 6 таблиц: COMPUTER\_INFO, SCHEDULE, HELP\_TB, PROCEDURE\_INFO, USER и USER\_INFO. Таблицы связаны между собой следующим образом:

* Таблица COMPUTER\_INFO содержит информацию о компьютерах, включая тип компьютера, процессор, видеокарту, оперативную память, описание и количество доступных экземпляров.
* Таблица PROCEDURE\_INFO определяет тарифы с их длительностью, стоимостью и описанием.
* Таблица USER хранит учетные записи пользователей, включая логин (ID), хэшированный пароль, роль пользователя (например, администратор или обычный пользователь) и статус восстановления пароля (passForgot).
* Таблица USER\_INFO содержит дополнительные данные пользователей, такие как имя, фамилия, отчество, мобильный телефон, электронная почта и дата рождения. Эта таблица связана с таблицей USER через внешний ключ UserId.
* Таблица HELP\_TB является сводной таблицей, которая связывает таблицы PROCEDURE\_INFO и COMPUTER\_INFO через внешние ключи ProcedureId и ComputerId.
* Таблица SCHEDULE содержит расписание использования компьютеров пользователями. Она включает время начала, время прибытия, время окончания, дату бронирования и внешние ключи для связи с пользователями (UserId), компьютерами (ComputerId) и тарифами (ProcedureId).

**3.4. Проектирование функционала проекта**

Для общего представления функционального назначения системы используется диаграмма использования, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. На диаграмме использования применяются два типа основных сущностей: варианты использования и группы пользователей.

Разные группы пользователей в диаграмме называются актёрами, и обозначают любые сущности, использующие систему. Любая функция системы называется вариантом использования. Каждый вариант использования обозначает набор действий, который может быть использован актёром для взаимодействии с системой, и определяет набор действий, выполняемых этой системой.

Диаграмма использования представлена в приложении Б.

# **3.5. Проектирование последовательностей проекта**

В целях визуализации взаимодействия объектов системы между собой во времени в едином сценарии использования используется ещё одна UML-диаграмма – диаграмма последовательностей. Данная диаграмма иллюстрирует, как различные части системы взаимодействуют друг с другом для выполнения функции, а также порядок, в котором происходит взаимодействие при выполнении конкретного случая использования.

Для отображения течения времени используется линия жизни объекта, которая изображается с помощью штриховой линии, которая проводится вертикально вниз. С помощью линии жизни показывается период, в течение которого объект существует в системе. Сами объекты изображаются в виде прямоугольников, а сообщения, которыми они обмениваются – в виде линий со стрелками. Диаграмма последовательностей представлена в приложении В.

**3.6.** Выводы по разделу:

В разделе была выбрана архитектура проекта, которая задаёт общий план проектирование продукта. Также для программного средства были спроектированы сущности базы данных, на основе которых получены их аналоги в языке программирования C#.

# **4. Реализация программного средства**

# **Основные классы программного средства**

Для выполнения технических задач программного средства должны быть реализованы следующие функции и соответствующие им классы и методы:

* выполнение входа;
* управление пользователями;
* управление тарифами;
* управление компьютерами;
* бронирование компьютера и выбор тарифа;

Далее подробно рассмотрены каждые из необходимых для выполнения технических задач функции, а также созданные для их выполнения классы и методы и их функционал и реализация.

# **Описание классов и методов программного средства**

## Выполнение входа

Выполнение входа осуществляется в отдельном окне – LoginWindow. В своём контексте данное окно ссылается на отдельную модель представления LoginViewModel.

Модель LoginViewModel представляет собой класс, реализующий паттерн MVVM (Model-View-ViewModel) для приложения, связанного с авторизацией, регистрацией и восстановлением пароля. В данном контексте она используется для обработки действий пользователя и взаимодействия с UI, а также для работы с базой данных. К основным методам относятся:

* ExecuteLogin: метод для обработки логики входа пользователя. Он проверяет введенные логин и пароль, а затем выполняет SQL-запрос, чтобы проверить их в базе данных. Если данные корректны, открывается окно для пользователя или администратора в зависимости от роли.
* ExecuteRegister: метод для открытия окна регистрации. После этого закрывается текущее окно.
* ExecuteClose: метод для закрытия приложения.
* ExecuteForgotPassword: метод для восстановления пароля. Он проверяет, введен ли логин, и если да — обновляет статус пользователя в базе данных, чтобы указать, что пароль был забыт.

Код класса LoginViewModel представлен в приложении Г.

## Управление пользователями

Для добавления новых пользователей используется отдельная страница администратора – ModerView, доступная пользователям роли admin. Для этого необходимо заполнить форму, заполнить такие поля как: логин, пароль, выбрать необходимую роль пользователя. При успешной проверке валидации, страница создает новый экземпляр класса User с полученными ранее параметрами, хэшируя пароль и записывая новую запись в соответствующую таблицу в базе данных.

Для администратора так-же доступна возможность удалить не активную учетную запись пользователя при необходимости введя соответствующий логин пользователя.

Он так-же может изменить роль пользователя и пароль пользователя, если тот отправил запрос о том, что он его забыл. После этого администратор меняет пароль пользователю и система высылает ему сообщение на адрес электронной почты с новым временным паролем.

Всё это так-же осуществляется отдельной моделью представления – ModerViewModel, которая представлена в приложении Г.

## Управление тарифами

Управление тарифами со стороны администратора, так-же как и представлением входа, осуществляется на отдельной странице – ServicePage, или в отдельном модальном окне, если он хочет добавить новый тариф, на странице – AdminView.

Обе бизнес-логики описаны в отдельных классах, на которые они ссылаются, как на отдельную модель представления – AdminViewModel и ServiceViewModel.

Модели ServiceViewModel и AdminViewModel управляют данными о тарифах в системе, обеспечивая взаимодействие с UI через команды и события. Она выполняет запросы к базе данных для поиска, добавления, удаления и обновления информации о тарифах, а также отображает эти данные в DataGrid, валидируя их перед этим. Модель придерживается принципов MVVM, разделяя логику представления (View) и обработки данных (Model), а также использует механизм команд для обработки действий пользователя. К основным методам относятся:

* RegestrProc\_Click: Метод для регистрации нового тарифа в базе данных.
* DeleteLast\_Click: метод для удаления последнего тарифа из базы данных. Запрос удаляет запись с наибольшим ID.
* SearchProc\_Click: метод для поиска тарифа по названию. Он выполняет SQL-запрос с параметром поиска и отображает результаты в DataGrid.
* SearchPrice\_Click: метод для поиска тарифа по цене. Выполняет SQL-запрос для поиска процедур с указанной ценой и отображает результаты в DataGrid.
* Display\_Data: метод для получения и отображения всех данных о тарифах. Он выполняет запрос для получения всех записей из таблицы PROCEDURE\_INFO и выводит их в DataGrid.

Листинг кода классов ServiceViewModel и AdminViewModel представлен в приложении Г.

## Управление компьютерами

Управление компьютерами со стороны администратора, так-же как и представлением управления тарифами, осуществляется на отдельной странице – ComputerPage, или так-же в отдельном модальном окне, если он хочет добавить новый компьютер, на странице – AdminView.

Добавление компьютеров в базу данных тесно связано с управлением аппаратным обеспечением в системе, что выполняется через класс AdminViewModel, где администратор может добавлять, обновлять или удалять записи о компьютерах, а также управлять параметрами оборудования, такими как тип, процессор, видеокарта, количество и другие характеристики. В AdminViewModel используются SQL-запросы для добавления новых компьютеров в таблицу COMPUTER\_INFO, что напрямую связано с функциональностью, описанной в ComputerViewModel, где обновляются или удаляются существующие компьютеры, а также отображаются данные о компьютерах в интерфейсе. Оба класса работают с одной и той же таблицей в базе данных, обеспечивая управление как созданием новых записей, так и их редактированием и удалением. Естественно с соотвутствующей валидацией. К оснонвым методам классов можно отнести:

* SearchProcessor\_Click и SearchType\_Click в ComputerViewModel позволяют фильтровать данные о компьютерах по процессору и типу, что делает работу с данными более гибкой.
* DeleteLast\_Click из ComputerViewModel позволяет удалять записи, что также может быть полезным в административной панели, где необходимо управлять данными о компьютерах.
* InsertComputer\_Click в AdminViewModel и UpdateComputer в ComputerViewModel работают с данными о компьютерах. Первый метод добавляет новый компьютер в базу данных, а второй — обновляет информацию о уже существующем компьютере.

Листинг класса ComputerViewModel представлен в приложении Г.

**4.4.5. Бронирование компьютера и выбор тарифа**

После успешного обновления списка компьютеров и тарифов, покупатель имеет возможность забронировать определенный компьютер, с нужными ему параметрами, и выбрать нужный ему тариф на определенную дату и время.

Так-же как и везде, для реализации данного функционала использовалось отдельное представление – UserWindow, для взаимодействия с пользователем. А весь основной функционал и методы были вынесены в отдельную модель этого представления MVVM – UserViewModel.

По плану, после успешной авторизации пользователь выбирает день на который хочет забронировать определенный тип компьютера. После этого он просматривает занятость каждого компьютера и добавляет свою запись о бронировании на определенное время. После успешной брони, пользователю приходит оповещение от администрации на электронную почту.

К основным методам класса относятся:

* Поиск доступных компьютеров: Метод Bt\_Click и SrchSchedule помогают пользователю выбрать компьютер и дату для бронирования. Они проверяют, доступен ли компьютер в выбранное время.
* Регистрация на использование компьютера: Через метод RegestrPriem\_Click пользователь может забронировать компьютер, если время свободно.
* Отправка уведомлений: Когда бронирование успешно завершено, метод SendAnnounceToUser отправляет уведомление на электронную почту пользователя.
* Отображение расписания: С помощью метода Show в окне отображается информация о занятых и свободных слотах для выбранного компьютера.

Листинг класса UserViewModel представлен в приложении Г.

**4.5.** Выводы по разделу:

В разделе описан процесс успешной реализации программного средства спроектированного в предыдущий разделах. Были описаны классы и методы этих классов которые задавали работу программного средства.

## 5. Тестирование, проверка работоспособности и анализ полученных результатов

## 5.1. Тестирование авторизации

При авторизации программное средство учитывает многие факторы. Одним из таких является проверка корректности заполнения полей, так например поля логина и пароля не могут быть пустыми и не могут содержать информацию более 50 и 15 символов соответственно. Так-же предусмотрена проверка на существование пользователя по введенным данным. Пример попытки ввода некорректных данных представлен на рисунках 5.1 и 5.2.

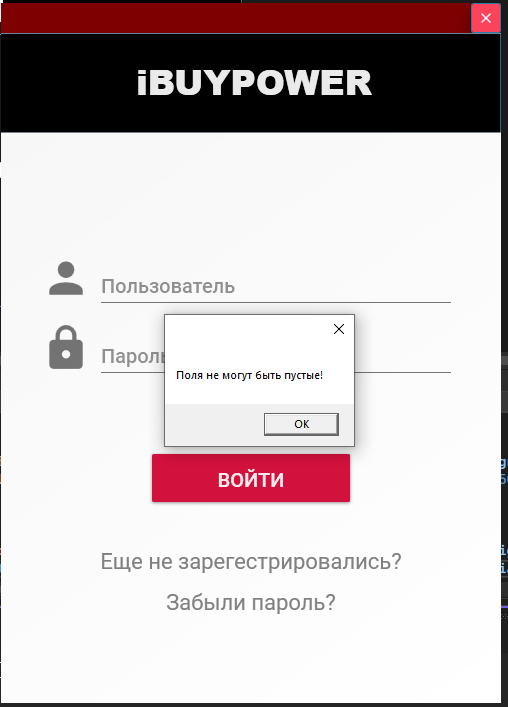
****

Рисунок 5.1 – Обработка пустых полей

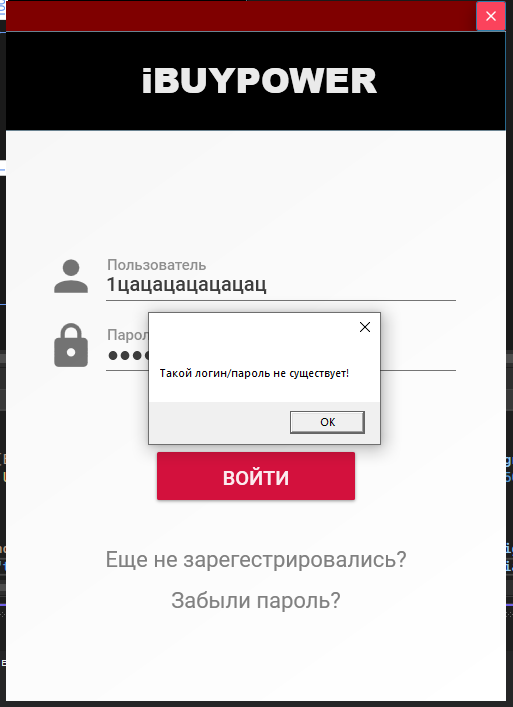
****

Рисунок 5.2 – Обработка неправильно введённых данных

Таким образом программное средство подсказывает пользователю, что ему необходимо изменить чтобы успешно войти в систему.

## 5.2. Тестирование создания тарифа

В форме создания тарифа применяются следующие правила валидации:

* все поля обязательны к заполнению;
* ограничение на длину – 30 символов;
* только неотрицательные числа в поле цена.

Пример тестирования системы валидации данной формы приведен на рисунке 5.3.

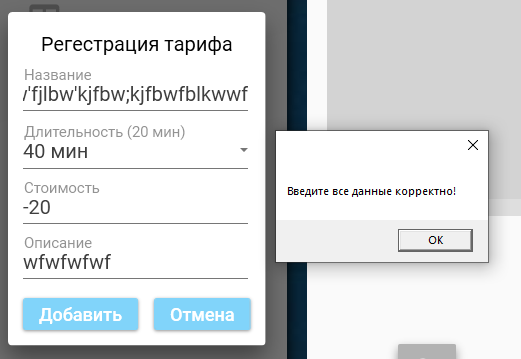


Рисунок 5.3 – Обработка полей создания тарифа

Таким образом, пользователь не сможет указать отрицательную стоимость аренды, так как такое поведение подозрительно и, предположительно, не соответствует ожидаемому результату.

## 5.3. Тестирование удаления тарифа

Администратор не может удалить тарифы с которыми связаны компьютеры, так как это операция над данными не является безопасной. В этом случае программное средство будет выводить сообщение об ошибке, которое представлено на рисунке 5.4.

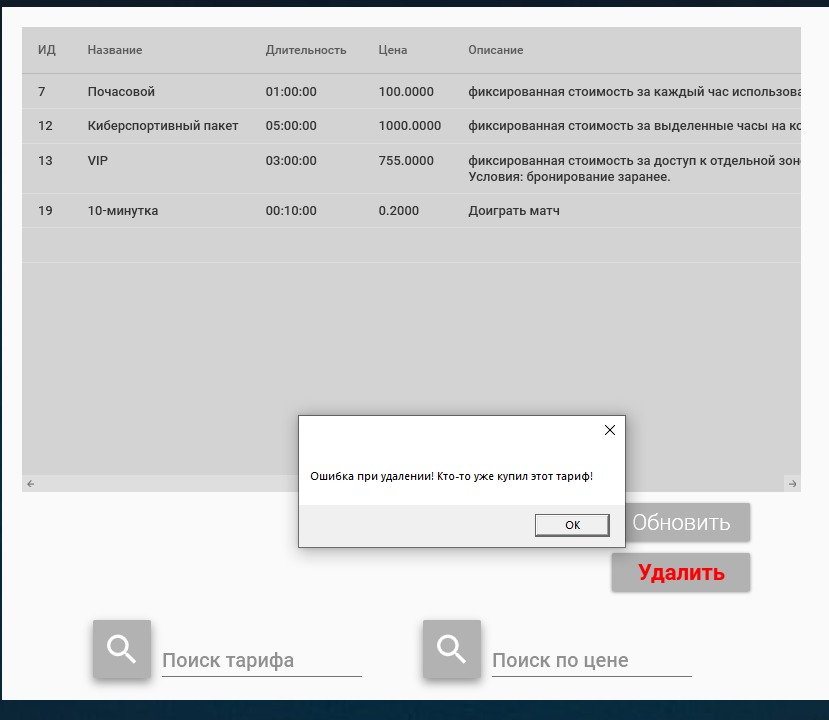


Рисунок 5.4 – Обработка удаления тарифа

Благодаря такого рода проверке, программное средство гарантирует сохранность данных, а также улучшает безопасность.

## 5.4. Тестирование управления пользователями

Администратор должен соблюдать множество правил при заполнении данной формы. К ним относиться:

* заполнить все поля;
* минимальная длина каждого поля 3 символа;
* максимальная длина каждого поля 30 символов;
* роль может быть либо Admin, либо User;
* при изменении пароля должен быть выдан доступ;
* при удалении пользователя проверка на наличие броней.

В случае не соблюдения хотя бы одного правила, программное средство будет выводить сообщение об ошибке, которое представлено на рисунке 5.5.



Рисунок 5.5 – Обработка полей добавления пользователя

Либо же если введены не все данные на рисунке 5.6:

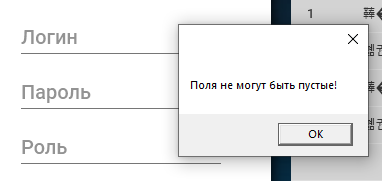


Рисунок 5.6 – Обработка полей добавления пользователя

Проверка на наличие флага возможности изменения пароля администратором на рисунке 5.7:

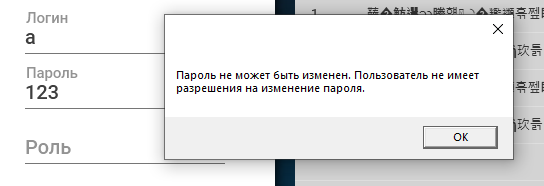


Рисунок 5.7 – Обработка полей изменения пароля

Проверка на наличие активных броней у пользователя при удалении пользователя на рисунке 5.8:

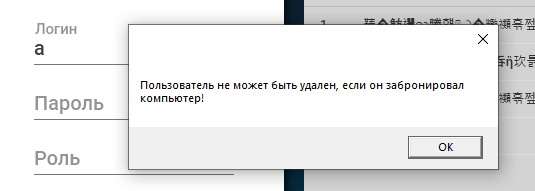


Рисунок 5.8 – Обработка полей удаления пользователя

## 5.5. Тестирование бронирования компьютера и выбора тарифа

Пользователь должен соблюдать множество правил при заполнении данной формы бронирования и выбора даты. К ним относиться:

* заполнить все поля;
* выбор даты не раньше завтрашней и не позже месяца;
* выбор времени прибытия не пересекающимся с другим клиентом;

В случае не соблюдения хотя бы одного правила, программное средство будет выводить сообщение об ошибке, которое представлено на рисунке 5.9.



Рисунок 5.8 – Обработка полей выбора даты и компьютера

Так-же при добавлении брони должны быть заполнены поля:

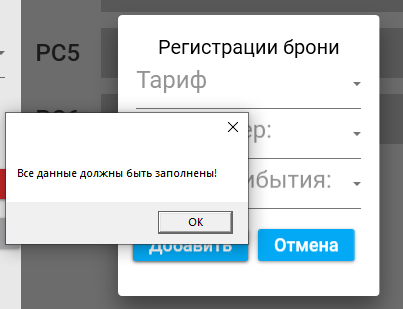


Рисунок 5.9 – Обработка полей добавления брони

И при случае пересечения времени прибытия с другим клиентом:

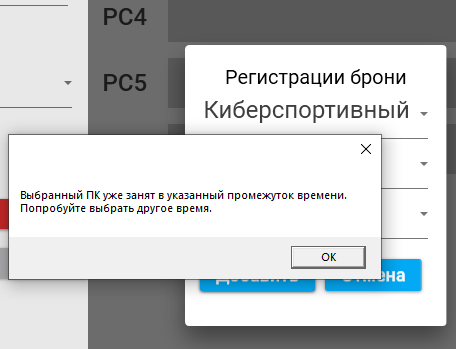


Рисунок 5.10 – Обработка полей добавления брони

## 5.6 Выводы по разделу:

В разделе описан процесс тестирования продукта. Были проверены случаи негативного и положительного тестирования. Были проверены все случаи валидации вводимых пользователем данных.

## Руководство по установке и использованию

При запуске программного средства пользователь попадает на страницу авторизации, содержащая форму входа.

Стартовая страница позволяет зарегистрированному пользователю совершить вход в аккаунт, для этого ему необходимо ввести корректные логин и пароль, после чего пользователь будет перенаправлен на главную страницу приложения. Иначе же, если пользователь не зарегистрирован, то он может перейти по соответствующей ссылке для дальнейшей регистрации. Либо же, выбрать ссылку для запроса смены пароля при необходимости.

Окна авторизации и регистрации представлены на рисунках 6.1 и 6.2.

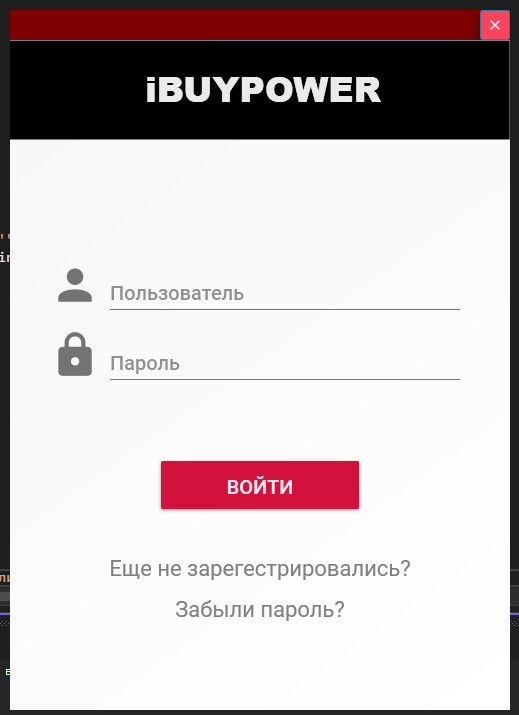


Рисунок 6.1 – Окно входа

Окно регистрации представлено соответсвенно ниже:

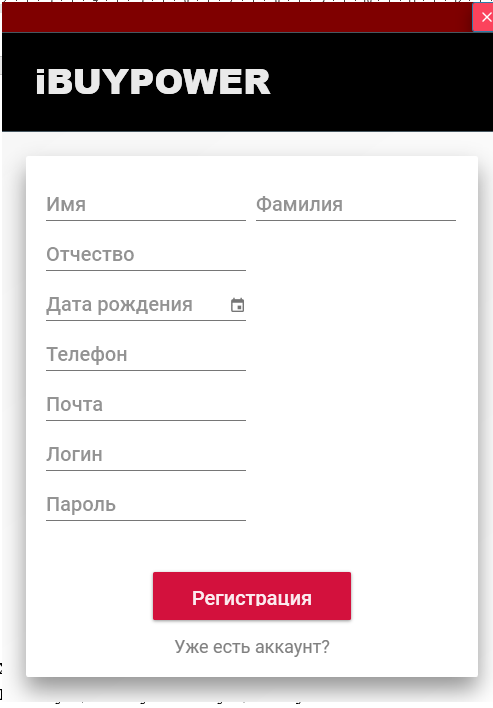


Рисунок 6.2 – Окно регистрации

Находясь на главной странице пользователь получает доступ к всему доступному функционалу в зависимости от роли, пользователи так-же получают доступ к возможности выхода с аккаунта или ознакомления со справочной информацией, а так-же возможности минимизировать окно нажав на соответсвующие кнопки в панели управления наверху. Стоит отметить, что внешний вид приложения слегка отличается в зависимости от роли пользователя. На рисунках 6.3 и 6.4 представлены внешний вид главной страницы для администратора и клиента соответсвенно.

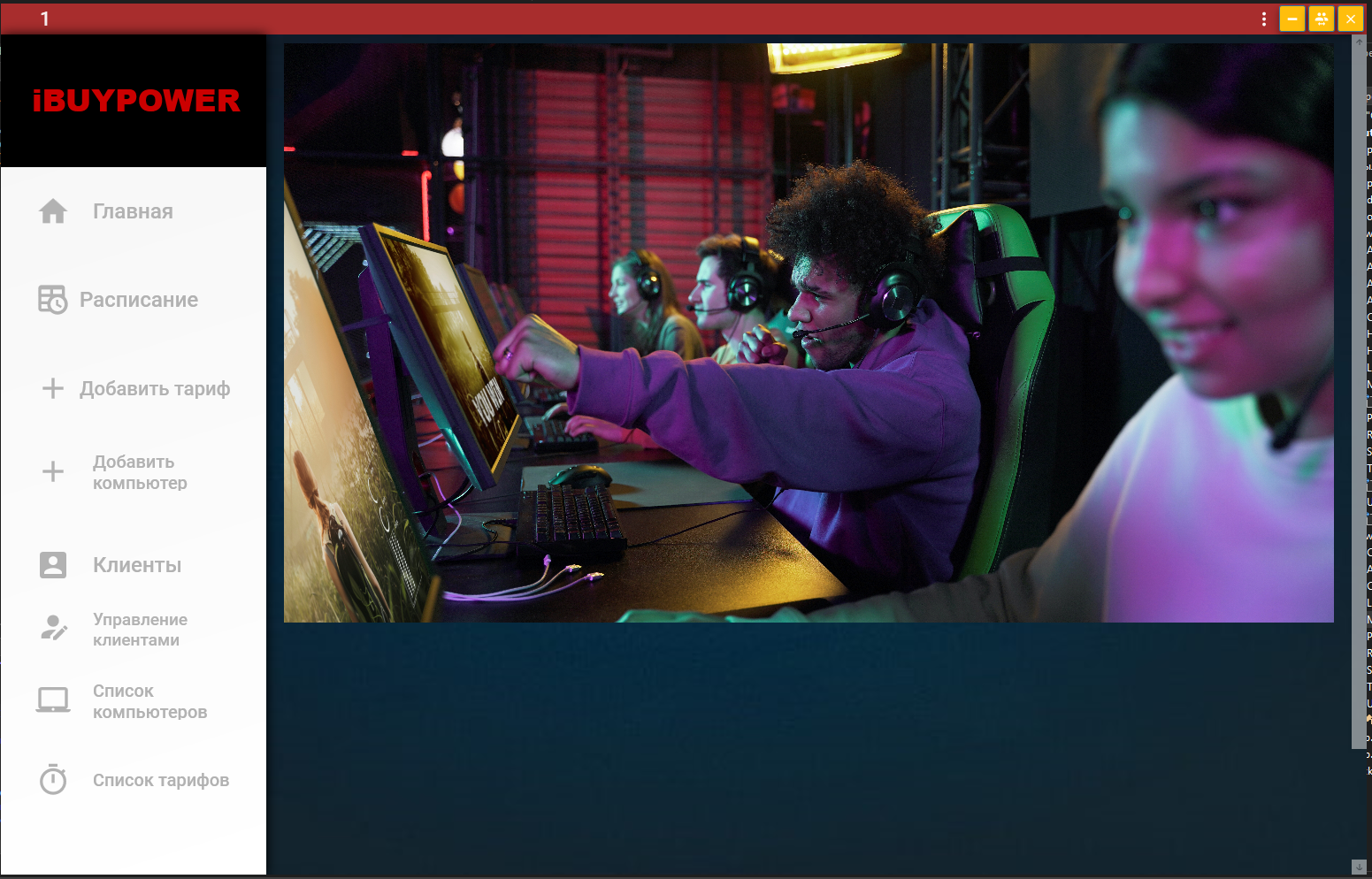


Рисунок 6.3 – Главная страница администратора

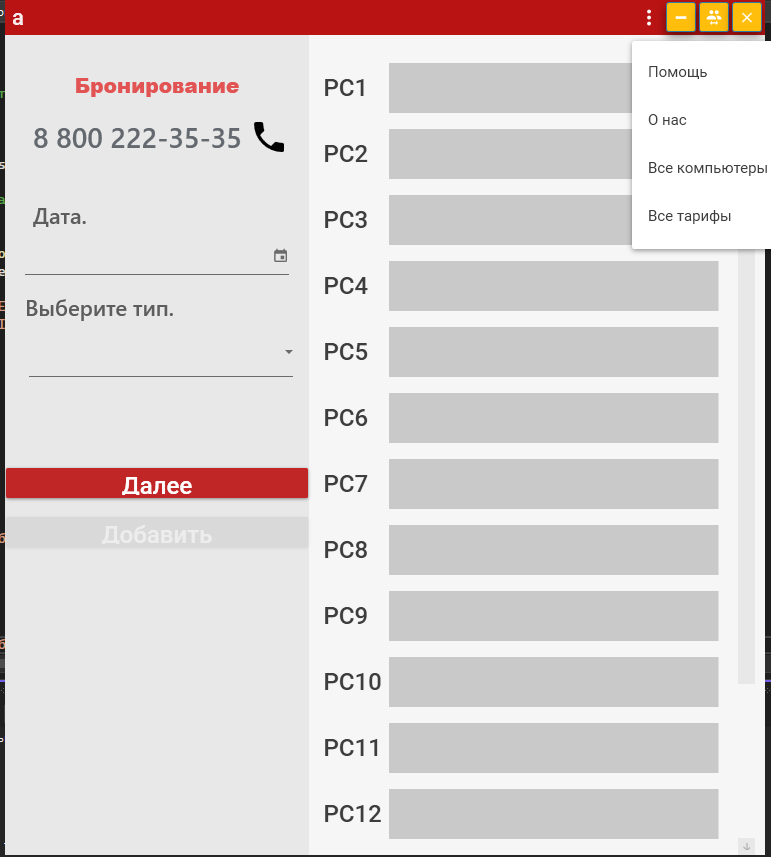


Рисунок 6.4 – Главная страница клиента

На странице клиента отображается интерфейс для бронирования компьютеров в клубе. Слева размещена панель для ввода данных, где пользователь может выбрать дату бронирования через календарь (поле "Дата") и указать тип компьютера из выпадающего списка (поле "Выберите тип"). Ниже находится кнопка "Далее", предназначенная для перехода к следующему этапу процесса бронирования. Справа расположены списки доступных компьютеров, обозначенные как PC1, PC2, ..., PC12, с соответствующими серыми блоками, которые могут отображать состояние компьютеров. В верхней части интерфейса присутствует контактная информация, включая номер телефона и иконку для звонка. Так-же имеется возможность ознакомиться со справочной информацией в выпадающем списке. При выборе даты и компьютера для бронирования пользователю даётся возможность для добавления своей брони с соответсующей формой. Данная форма представлена на рисунке 6.5.

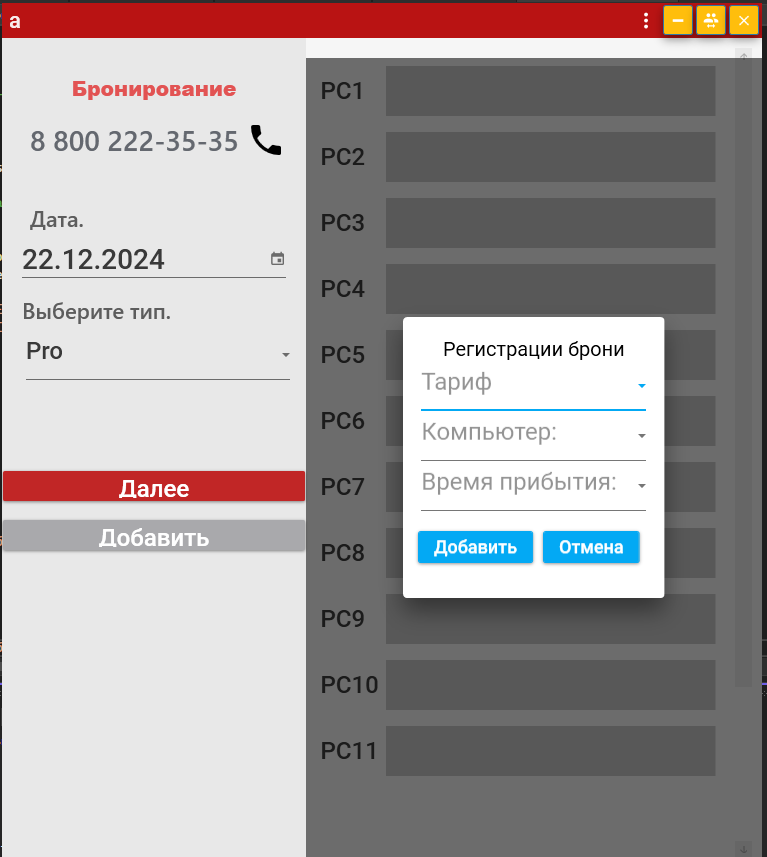


Рисунок 6.5 – Форма для добавления брони клиентом

На странице администратора представлен интерфейс управления компьютерным клубом, разделенный на два основных блока. Слева расположено навигационное меню, включающее разделы: Главная, Расписание, Добавить тариф, Добавить компьютер, Клиенты, а также Список компьютеров и Список тарифов. Эти разделы позволяют администратору управлять тарифами, оборудованием и данными клиентов. Справа находится основное рабочее пространство, где будет отображаться вся рабочая панель администратора в дальнейшем. Страница для управления расписанием администратора представлена на рисунке 6.6.

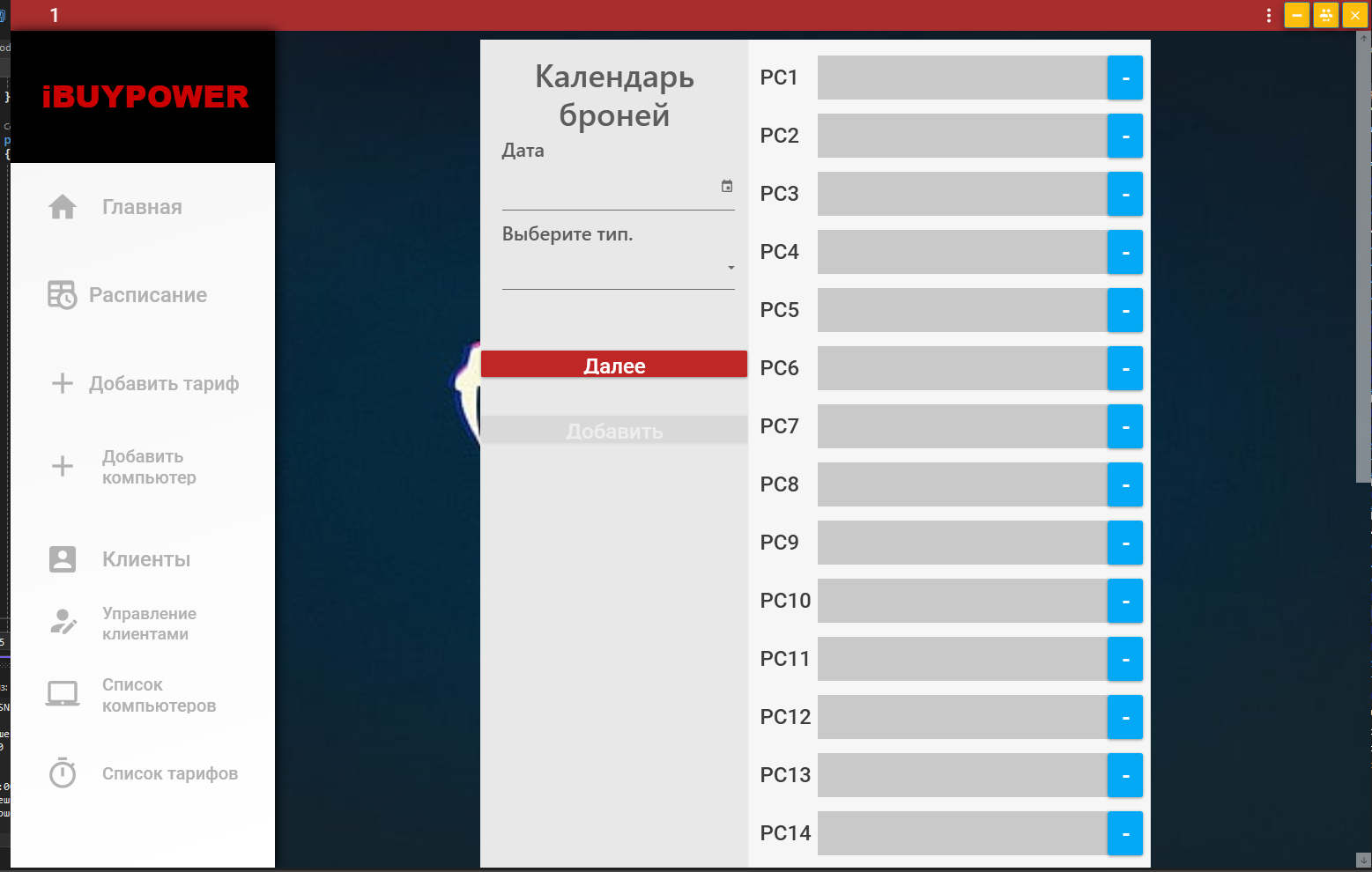


Рисунок 6.6 – Страница расписания

Находясь на странице, администратор имеет всё ту-же возможность управлять бронирование компьютеров с дополнительным функционалом: добавления брони по логину пользователя в соотсветсвующей форме, а так-же удаления всех броней на определенном компьютере в определенный день. Вкладки добавления тарифа и компьютера имеют схожий с добавлением брони принцип модального окна с сооответсующей валидацией. Пример окна добавления компьютера предстален на рисунке 6.7.



Рисунок 6.7 – Окно добавления компьютера

На странице “Клиенты” представлен список клиентов с подробной информацией, включая логин, фамилию, имя, отчество, дату рождения, почту и телефон. Пользователь может искать клиента по логину или фамилии через строки поиска, расположенные ниже таблицы, а также обновить данные, используя кнопку "Обновить". Дизайн интерфейса представлен на рисунке 6.8.

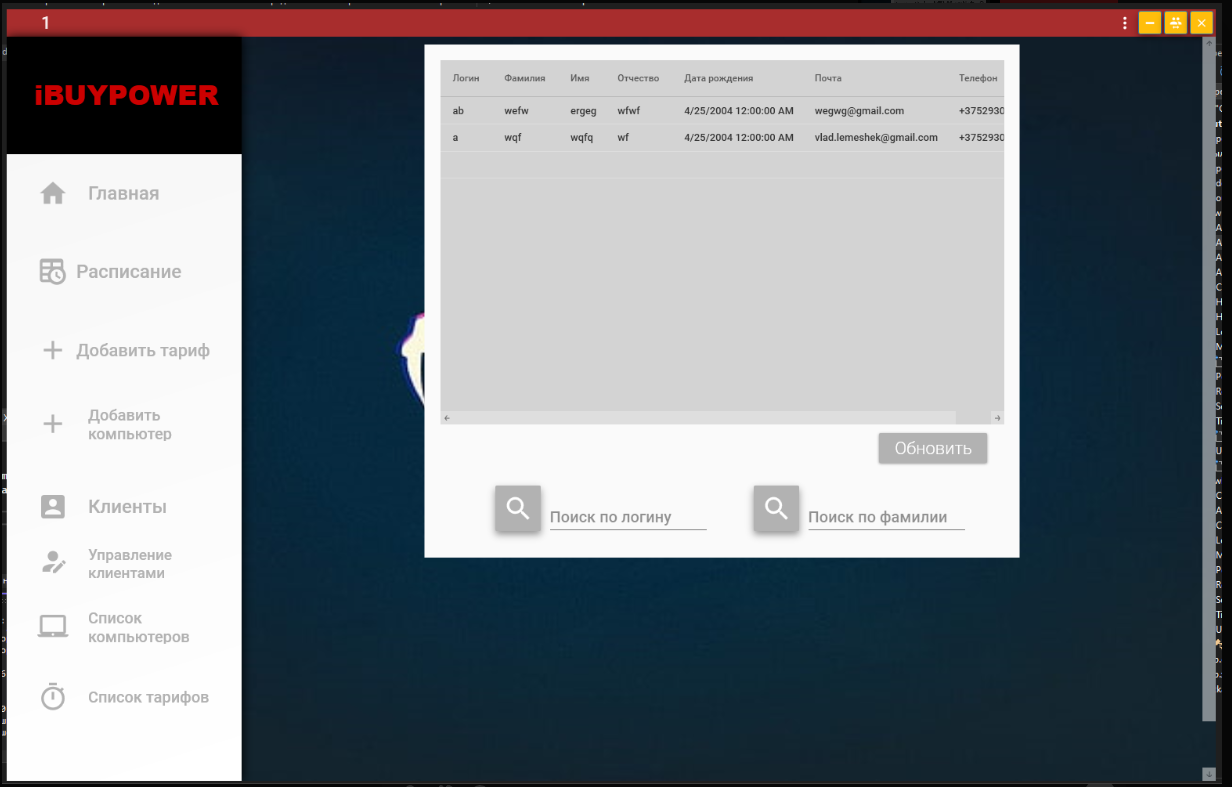


Рисунок 6.8 – Старница клинтов

Страница управления клиентами позволяет администратору управлять учетными записями клиентов. Содержит таблицу с данными, включая ID, пароль, роль (например, "user" или "admin"), а также флаг passForgot (забытый пароль). Справа расположена панель для ввода и редактирования данных (логин, пароль, роль) с кнопками "Подтвердить", "Удалить" и "Изменить", позволяющими добавить нового пользователя, удалить неактивного или изменить пароль существующему при необходимости. И так-же кнопка для обновления существующих данных. Страница представлена на риснуке 6.9.

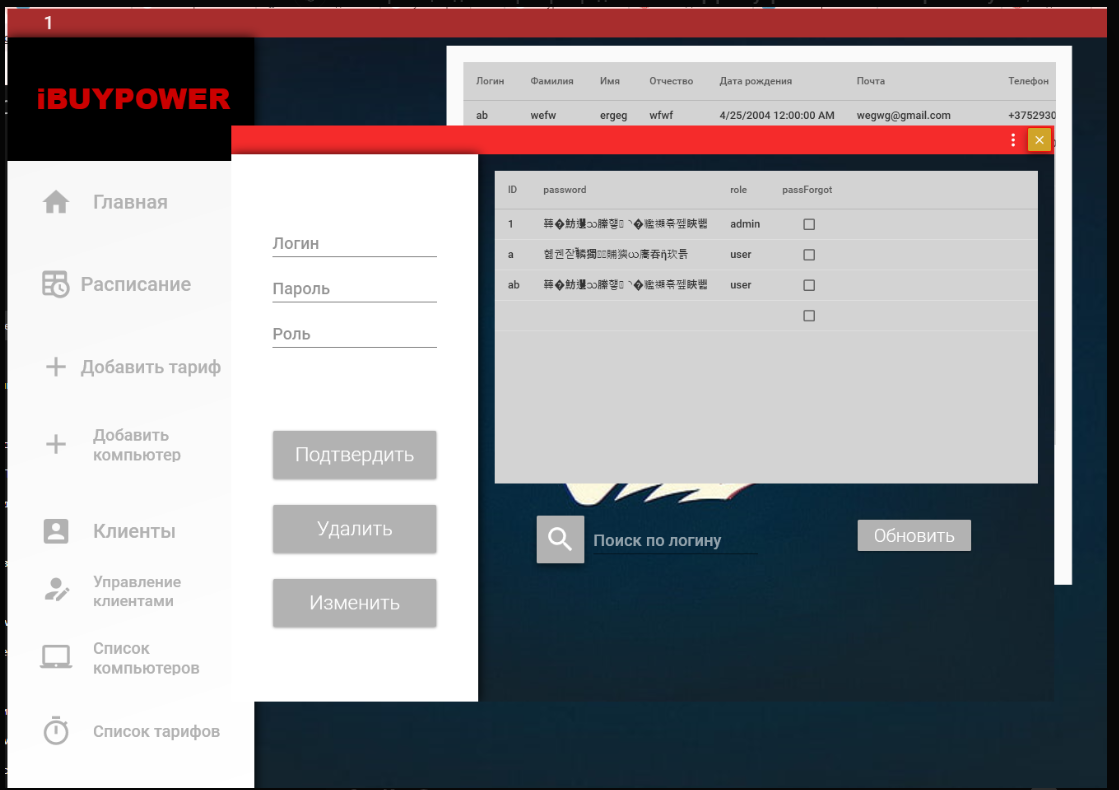


Рисунок 6.9 – Старница управления клинтами

На странице “Список компьютеров” отображается информация о компьютерах в клубе. В таблице указаны ID, тип, процессор, видеокарта, ОЗУ, описание и количество. Справа размещена форма для изменения данных о выбранном компьютере, включая поля для типа, процессора, видеокарты, ОЗУ, описания и количества. Кнопки "Удалить" и "Изменить" позволяют редактировать или удалять послденюю запись, при условии соблюдения валидации и правил. Также присутствует поиск компьютеров по типу и процессору. При нажатии на соовтесвующих кнопках на всех страница всегда будет производиться валидация необходимых полей на всех страницах. Страница представлена на рисунке 6.10.



Рисунок 6.10 – Старница компьютеров

Страница “Список тарифов” содержит таблицу с данными о тарифах, включая ID, название, длительность, цену и описание. Справа расположена панель для редактирования данных тарифа с полями название, цена, длительность и описание. Администратор может обновить информацию, удалить выбранный тариф или выполнить поиск по названию тарифа и цене с помощью соответствующих строк поиска. Страница представлена на рисунке 6.11.



Рисунок 6.11 – Старница тарифов

## Выводы по разделу:

В разделе описан процесс использования программного средства. В нем пользователя инструктирую, как воспользоваться каждой из функции, которые предоставляет программное средство.

## Заключение

В ходе выполнения курсового проекта было разработано программное средство для управления компьютерным клубом на языке C# с использованием технологий ADO .NET, WPF.

При разработке программного средства были выполнены все пункты из указанного списка предполагаемого основного функционала приложения, а именно:

* создана база данных;
* реализована возможность авторизации и регистрации;
* реализована возможность управлением тарифами;
* реализована возможность управлением компьютерами;
* реализована возможность управлением клиентами;
* реализован интерфейс и функционал для бронирования компьютеров клиентами;

Кроме того, программное средство отличается четкой структурой и архитектурой, основанной на паттерне MVVM (Model-View-ViewModel), что позволило разделить логику, данные и представление, обеспечив гибкость и удобство дальнейшей поддержки и масштабирования. Интерфейс разработан с учетом интуитивной понятности для пользователей, что упрощает взаимодействие с системой как для администраторов, так и для клиентов. Реализация функций взаимодействия с базой данных через ADO.NET гарантирует надежное хранение и обработку данных, а использование WPF позволило создать современный и визуально привлекательный интерфейс. Результаты тестирования подтверждают устойчивость работы программы при различных сценариях использования, что свидетельствует о ее надежности и соответствии заявленным требованиям технического задания

# **Список литературы**

1. Онлайн-ресурс “smartshell.gg” [Электронный ресурс] – Режим доступа smartshell.gg – Дата доступа: 02.12.2024
2. Онлайн-ресурс “astrumsoft.com” [Электронный ресурс] – Режим доступа astrumsoft.com – Дата доступа: 02.12.2024
3. Онлайн-ресурс “lightshell.ru” [Электронный ресурс] – Режим доступа lightshell.ru – Дата доступа: 02.12.2024
4. Пацей, Н.В. Курс лекций по языку программирования C# / Н. В. Пацей. – Минск: БГТУ, 2018. – 175 с.
5. Руководство по WPF // [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/ – Дата доступа: 10.11.2024
6. Работа с ADO .NET [Электронный ресурс] – https://metanit.com/sharp/adonet/ – Дата доступа 10.11.2024

## ПРИЛОЖЕНИЕ А

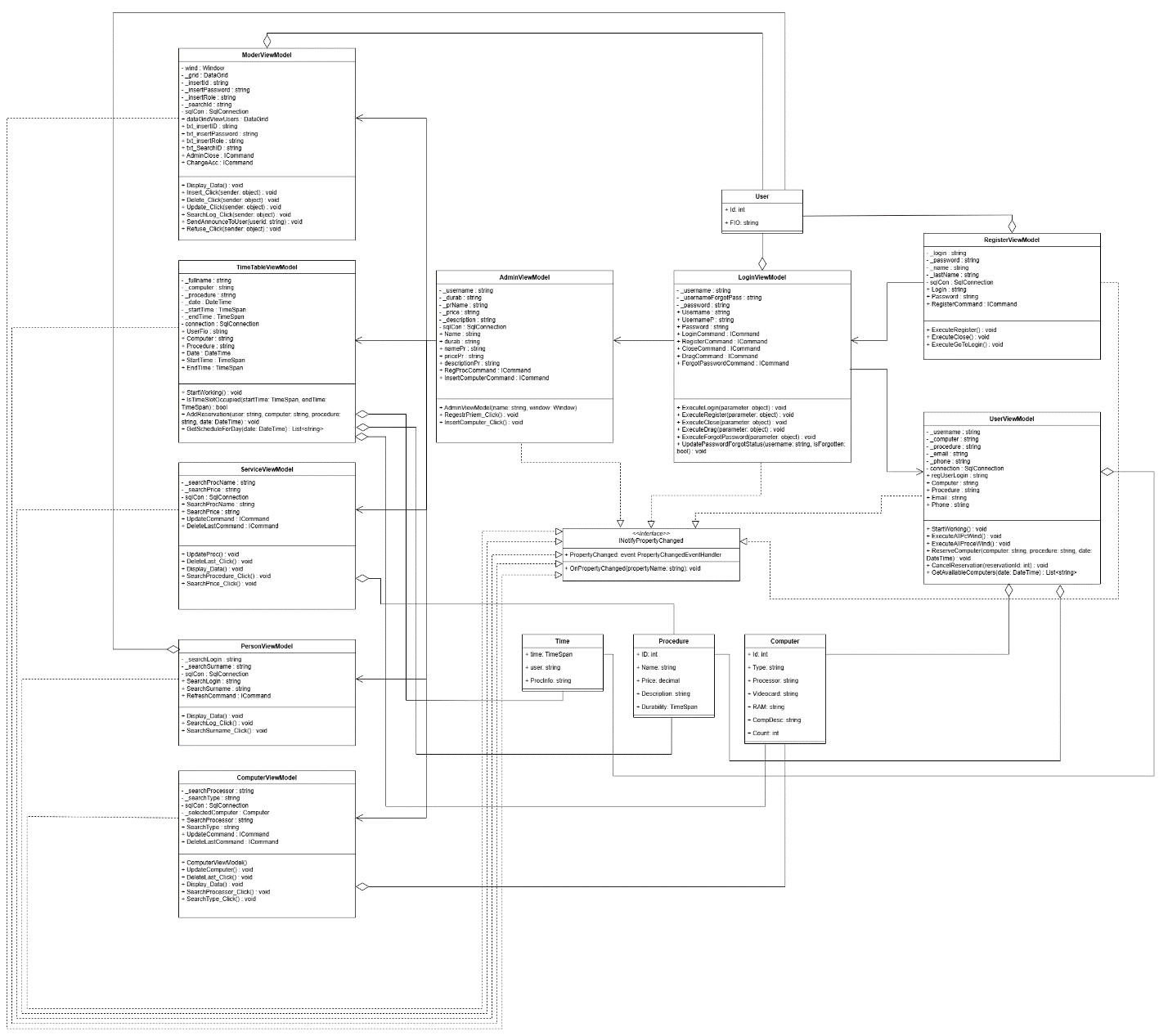


Рисунок А.1 – диаграмма классов

## ПРИЛОЖЕНИЕ Б

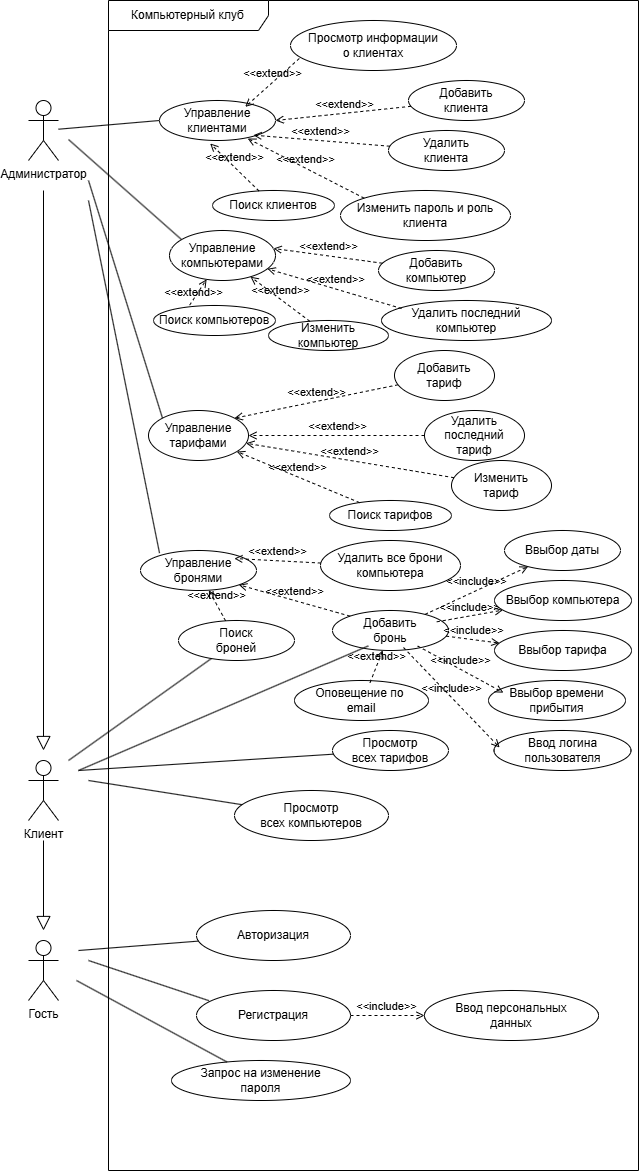


Рисунок Б.1 – диаграмма вариантов использования

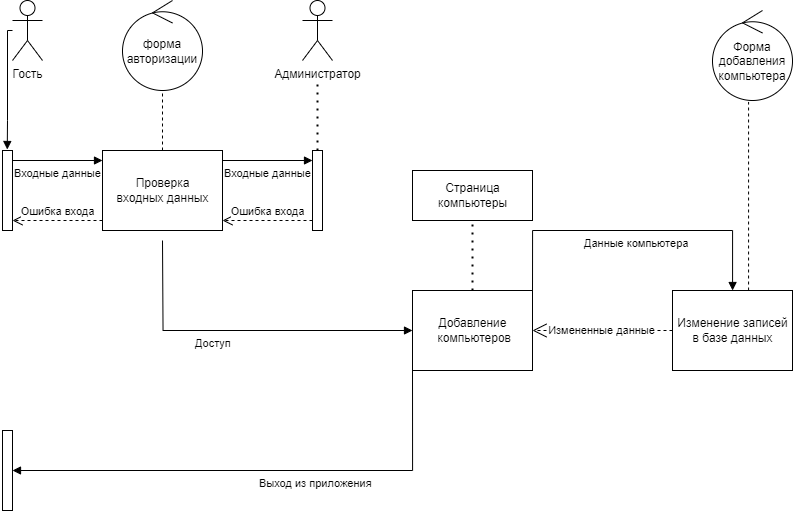
ПРИЛОЖЕНИЕ В

Рисунок В.1 – диаграмма последовательности

## 

## ПРИЛОЖЕНИЕ Г

Листинг 1: Класс LoginViewModel

|  |
| --- |
| public class LoginViewModel  {  private string \_username;  private string \_usernameForgotPass;  private string \_password;  public static string Online\_person { get; private set; }  public string Username  {  get => \_username;  set  {  \_username = value;  OnPropertyChanged();  }  }  public string UsernameP  {  get => \_usernameForgotPass;  set  {  \_usernameForgotPass = value;  OnPropertyChanged();  }  }  public string Password  {  get => \_password;  set  {  \_password = value;  OnPropertyChanged();  }  }  public ICommand LoginCommand { get; }  public ICommand RegisterCommand { get; }  public ICommand CloseCommand { get; }  public ICommand DragCommand { get; }  public ICommand ForgotPasswordCommand { get; }  public event PropertyChangedEventHandler PropertyChanged;  protected void OnPropertyChanged([CallerMemberName] string propertyName = null)  {  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));  }  public LoginViewModel()  {  LoginCommand = new RelayCommand(ExecuteLogin);  RegisterCommand = new RelayCommand(ExecuteRegister);  CloseCommand = new RelayCommand(ExecuteClose);  DragCommand = new RelayCommand(ExecuteDrag);  ForgotPasswordCommand = new RelayCommand(ExecuteForgotPassword);  }  private void ExecuteForgotPassword(object parameter)  {  if (!string.IsNullOrWhiteSpace(UsernameP))  {  UpdatePasswordForgotStatus(UsernameP, true);  }  else  {  MessageBox.Show("Пожалуйста, введите логин.");  }  }  private void ExecuteLogin(object parameter)  {  SqlConnection sqlCon = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\DataBase\stm.mdf';Integrated Security=True;Connect Timeout=30");  try  {  if (sqlCon.State == ConnectionState.Closed)  sqlCon.Open();  if (!string.IsNullOrWhiteSpace(Username) && !string.IsNullOrWhiteSpace(Password))  {  string query = "SELECT [USER].role FROM [USER] WHERE ID=@ID AND password=HASHBYTES('SHA2\_256', @password) ";  SqlCommand sqlCmd = new SqlCommand(query, sqlCon)  {  CommandType = CommandType.Text  };  sqlCmd.Parameters.AddWithValue("@ID", Username);  sqlCmd.Parameters.AddWithValue("@password", Password);  string role = Convert.ToString(sqlCmd.ExecuteScalar());  if (role == "admin")  {  OpenWindow(new AdminWindow(Username));  }  else if (role == "user")  {  OpenWindow(new UserWindow(Username));  }  else  {  MessageBox.Show("Такой логин/пароль не существует!");  }  }  else  {  MessageBox.Show("Поля не могут быть пустые!");  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при входе!");  }  finally  {  sqlCon.Close();  }  }  private void ExecuteRegister(object parameter)  {  RegisterWindow registerWindow = new RegisterWindow();  registerWindow.Show();  Application.Current.MainWindow.Close();  }  private void ExecuteClose(object parameter)  {  Application.Current.Shutdown();  }  private void ExecuteDrag(object parameter)  {  if (parameter is Window window)  {  window.DragMove();  }  }  private void OpenWindow(Window window)  {  Online\_person = Username;  window.Show();  Application.Current.MainWindow.Close();  }  private void UpdatePasswordForgotStatus(string username, bool isForgotten)  {  SqlConnection sqlCon = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\DataBase\stm.mdf';Integrated Security=True;Connect Timeout=30");  try  {  if (sqlCon.State == ConnectionState.Closed)  sqlCon.Open();  string queryCheck = "SELECT [USER].[Id]FROM [USER] WHERE Id = @Username";  SqlCommand sqlCmdCheck = new SqlCommand(queryCheck, sqlCon)  {  CommandType = CommandType.Text  };  sqlCmdCheck.Parameters.AddWithValue("@Username", username);  if (sqlCmdCheck.ExecuteScalar() == null)  {  MessageBox.Show("Такого логина не найдено!");  return;  }  // Запрос для обновления поля IsPasswordForgotten  string query = "UPDATE [USER] SET [passForgot] = @IsForgotten WHERE Id = @Username";  SqlCommand sqlCmd = new SqlCommand(query, sqlCon)  {  CommandType = CommandType.Text  };  // Добавляем параметры в запрос  sqlCmd.Parameters.AddWithValue("@Username", username);  sqlCmd.Parameters.AddWithValue("@IsForgotten", isForgotten ? 1 : 0);// 1 - если забыли пароль, 0 - если не забыли  sqlCmd.ExecuteNonQuery(); // Выполняем запрос  MessageBox.Show("Запрос обновлен успешно.");  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при обновлении статуса сброса пароля!");  }  finally  {  sqlCon.Close();  }  } |

Листинг 2: Класс ModerViewModel

|  |
| --- |
| public class ModerViewModel : INotifyPropertyChanged  {  public Window wind;  private DataGrid \_grid;  public DataGrid dataGridViewUsers { get => \_grid; set { \_grid = value; OnPropertyChanged(); } }  private string \_insertId;  private string \_insertPassword;  private string \_insertRole;  public string txt\_insertID { get => \_insertId; set { \_insertId = value; OnPropertyChanged(); } }  public string txt\_insertPassword { get => \_insertPassword; set { \_insertPassword = value; OnPropertyChanged(); } }  public string txt\_insertRole { get => \_insertRole; set { \_insertRole = value; OnPropertyChanged(); } }  private string \_searchId;  public string txt\_SearchID { get => \_searchId; set { \_searchId = value; OnPropertyChanged(); } }  public SqlConnection sqlCon = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\DataBase\stm.mdf';Integrated Security=True;Connect Timeout=30");//подключение бд  public event PropertyChangedEventHandler PropertyChanged;  protected void OnPropertyChanged([CallerMemberName] string propertyName = null)  {  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));  }  public ICommand AdminClose { get; }  public ICommand ChangeAcc { get; }  public ICommand HelpBtn { get; }  public ICommand AboutBtn { get; }  public ICommand InsertCommand { get; }  public ICommand DeleteCommand { get; }  public ICommand UpdateCommand { get; }  public ICommand SearchCommand { get; }  public ICommand RefreshCommand { get; }  public ModerViewModel(Window window)  {  this.wind = window;  AdminClose = new RelayCommand(AdminClose\_Click);  ChangeAcc = new RelayCommand(ChangeAcc\_Click);  HelpBtn = new RelayCommand(HelpBut\_Click);  AboutBtn = new RelayCommand(AboutBtn\_Click);  InsertCommand = new RelayCommand(Insert\_Click);  DeleteCommand = new RelayCommand(Delete\_CLick);  UpdateCommand = new RelayCommand(Update\_Click);  SearchCommand = new RelayCommand(SearchLog\_Click);  RefreshCommand = new RelayCommand(Refuse\_Click);  }  #region =Menu=  private void AdminClose\_Click(object sender)//закрыть окно  {  wind.Close();  }  private void ChangeAcc\_Click(object sender)//поменять акк  {  LoginWindow loginWindow = new LoginWindow();  loginWindow.Show();  wind.Close();  }  #endregion  #region=PopUp=  private void HelpBut\_Click(object sender)  {  HelpWindow helpWindow = new HelpWindow();  helpWindow.Show();  }  private void AboutBtn\_Click(object sender)  {  AboutWindow aboutWindow = new AboutWindow();  aboutWindow.Show();  }  #endregion  public void SendAnnounceToUser(string userid)  {  using (SqlConnection connection = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\DataBase\stm.mdf';Integrated Security=True;Connect Timeout=30"))  {  connection.Open();  string query = @"SELECT [USER\_INFO].[Email] , [Name] , [Surname]from [USER\_INFO] where [UserId] = @Userid";  using (SqlCommand sqlCmd = new SqlCommand(query, connection))  {  try  {  sqlCmd.CommandType = CommandType.Text;  sqlCmd.Parameters.AddWithValue("@Userid", userid);  var result = sqlCmd.ExecuteReader();  if (result.Read())  {  string name = result.GetString(1);  string surname = result.GetString(2);  string semail = result.GetString(0);  string uemail = "ibuypowerclub@gmail.com";  string upass = "abap wjhp scve smmi";  MailAddress to = new MailAddress(semail);  MailAddress from = new MailAddress(uemail);  MailMessage message = new MailMessage(from, to)  {  Subject = "iBUYPOWER",  IsBodyHtml = false,  Body = $"Уважаемый(ая) {name} {surname}! Ваш пароль был успешно изменен на временный : {txt\_insertPassword}\nПожалуйста не сообщайте никому и измените его в ближайшее время, если вам нужно! С уважением, администрация iBUYPOWER!"  };  SmtpClient smtp = new SmtpClient("smtp.gmail.com", 587)  {  Credentials = new NetworkCredential(uemail, upass),  EnableSsl = true  };  smtp.Send(message);  MessageBox.Show("Сообщение успешно отправлено!");  }  }  catch (FormatException)  {  MessageBox.Show("Неверный формат электронной почты. Почта должна иметь окончания - @gmail/yandex/mail/bk/list и другие");  return;  }  }  }  }  #region =Кнопки работы c DataBase=  private void Insert\_Click(object sender)//подтвердить  {  try  {  if (sqlCon.State == ConnectionState.Closed)//проверка на подключение к бд  {  if (txt\_insertID != "" && txt\_insertPassword != "" && txt\_insertRole != "")//проверка на пустые строки  {  if (txt\_insertRole == "user" || txt\_insertRole == "admin")//проверка на нужную роль  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText = "insert into [USER] (ID, password, role) values ('" + txt\_insertID + "', '" + txt\_insertPassword + "', '" + txt\_insertRole + "')";  cmd.ExecuteNonQuery();  sqlCon.Close();  Display\_Data();  txt\_insertID = "";  txt\_insertPassword = "";  txt\_insertRole = "";  MessageBox.Show("Добавлено!");  }  else  {  MessageBox.Show("Роль может быть только:\nuser\nadmin");  }  }  else  {  MessageBox.Show("Поля не могут быть пустые!");  }  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при добавлении пользователя!");  }  finally  {  sqlCon.Close();  }  }  public void Display\_Data() //показ DataBase [USER]  {  try  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText = "SELECT \* FROM [USER]";  cmd.ExecuteNonQuery();  DataTable dta = new DataTable();  SqlDataAdapter dtaAdp = new SqlDataAdapter(cmd);  dtaAdp.Fill(dta);  dataGridViewUsers.ItemsSource = dta.DefaultView;  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка отображения пользователе!");  }  finally  {  sqlCon.Close();  }  }  private void Refuse\_Click(object sender)//обновить DB  {  Display\_Data();  }  private void Delete\_CLick(object sender)//удалить запись  {  try  {  if (sqlCon.State == ConnectionState.Closed)  {  if (txt\_insertID != "")//проверка на пустые строки  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText =  "DELETE FROM [USER\_INFO] WHERE [UserId] = '" + txt\_insertID + "'; " +  "DELETE FROM [USER] WHERE [ID] = '" + txt\_insertID + "';";  cmd.ExecuteNonQuery();  Display\_Data();  txt\_insertID = "";  txt\_insertPassword = "";  txt\_insertRole = "";  MessageBox.Show("Удалено!");  }  else  {  MessageBox.Show("Поля не могут быть пустые!");  }  }  }  catch (Exception ex)  {  MessageBox.Show("Пользователь не может быть удален, если он забронировал компьютер!");  }  finally  {  sqlCon.Close();  }  }  private void Update\_Click(object sender) // изменить запись  {  try  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  SqlCommand cmdCheck = sqlCon.CreateCommand();  cmdCheck.CommandType = CommandType.Text;  cmdCheck.CommandText = "SELECT passForgot FROM [USER] WHERE ID = @UserID";  cmdCheck.Parameters.AddWithValue("@UserID", txt\_insertID);  var passForgotFlag = cmdCheck.ExecuteScalar();    if (passForgotFlag != null && (bool)passForgotFlag)  {    SqlCommand cmdUpdatePassword = sqlCon.CreateCommand();  cmdUpdatePassword.CommandType = CommandType.Text;  cmdUpdatePassword.CommandText = "UPDATE [USER] SET [Password] = HASHBYTES('SHA2\_256',@NewPassword) , [passForgot] = 0 WHERE [ID] = @UserID";  cmdUpdatePassword.Parameters.AddWithValue("@NewPassword", txt\_insertPassword);  cmdUpdatePassword.Parameters.AddWithValue("@UserID", txt\_insertID);  cmdUpdatePassword.ExecuteNonQuery();  SendAnnounceToUser(txt\_insertID);  txt\_insertID = "";  txt\_insertPassword = "";  txt\_insertRole = "";  MessageBox.Show("Пароль успешно обновлен!");  }  else  {  MessageBox.Show("Пароль не может быть изменен. Пользователь не имеет разрешения на изменение пароля.");  }  if (!string.IsNullOrEmpty(txt\_insertRole))  {    SqlCommand cmdUpdateRole = sqlCon.CreateCommand();  cmdUpdateRole.CommandType = CommandType.Text;  cmdUpdateRole.CommandText = "UPDATE [USER] SET [role] = @NewRole WHERE [ID] = @UserID";  cmdUpdateRole.Parameters.AddWithValue("@NewRole", txt\_insertRole);  cmdUpdateRole.Parameters.AddWithValue("@UserID", txt\_insertID);  cmdUpdateRole.ExecuteNonQuery();  Display\_Data();  txt\_insertID = "";  txt\_insertPassword = "";  txt\_insertRole = "";  MessageBox.Show("Роль успешно обновлена!");  }  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при обновлении пользователя!");  }  finally  {  sqlCon.Close();  }  }  private void SearchLog\_Click(object sender)//поиск по логину  {  try  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText = "SELECT \* FROM [USER] WHERE ID = '" + txt\_SearchID + "'";  cmd.ExecuteNonQuery();  DataTable dt = new DataTable();  SqlDataAdapter da = new SqlDataAdapter(cmd);  da.Fill(dt);  dataGridViewUsers.ItemsSource = dt.DefaultView;  sqlCon.Close();  txt\_SearchID = "";  }  }  catch (Exception ex)  {  MessageBox.Show("Отсутствуют данные!");  }  finally  {  sqlCon.Close();  }  }  #endregion  } |

Листинг 3: Класс ServiceViewModel

|  |
| --- |
| public class ServiceViewModel : INotifyPropertyChanged  {  public DataGrid \_dataGridViewUsers1;  private string \_searchProcName;  private string \_searchPrice;  private Procedure \_selectedProc = new Procedure();  public Procedure SelectedProc  {  get => \_selectedProc;  set  {  \_selectedProc = value;  OnPropertyChanged(nameof(SelectedProc));  }  }  public DataGrid dataGridViewUsers1  {  get => \_dataGridViewUsers1;  set  {  \_dataGridViewUsers1 = value;  OnPropertyChanged(nameof(SearchProcName));  }  }  public string SearchProcName  {  get => \_searchProcName;  set  {  \_searchProcName = value;  OnPropertyChanged(nameof(SearchProcName));  }  }  public string SearchPrice  {  get => \_searchPrice;  set  {  \_searchPrice = value;  OnPropertyChanged(nameof(SearchPrice));  }  }  public SqlConnection sqlCon = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\DataBase\stm.mdf';Integrated Security=True;Connect Timeout=30");//подключение бд  public event PropertyChangedEventHandler PropertyChanged;  protected void OnPropertyChanged([CallerMemberName] string propertyName = null)  {  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));  }  public ICommand RefreshCommand { get; set; }  public ICommand SearchProcedureCommand { get; set; }  public ICommand SearchPriceCommand { get; set; }  public ICommand DeleteLastCommand { get; set; }  public ICommand UpdateCommand { get; }  public ServiceViewModel()  {  RefreshCommand = new RelayCommand(Refuse\_Click);  SearchProcedureCommand = new RelayCommand(SearchProcedure\_Click);  SearchPriceCommand = new RelayCommand(SearchPrice\_Click);  DeleteLastCommand = new RelayCommand(DeleteLast\_Click);  UpdateCommand = new RelayCommand(UpdateProc);  }  private void Refuse\_Click(object param)//обновить  {  Display\_Data();  }  private void UpdateProc(object param)  {  try  {  if (sqlCon.State == ConnectionState.Closed)  sqlCon.Open();  string query = "UPDATE [PROCEDURE\_INFO] SET " +  "[Procedure] = @proc, " +  "Price = @price, " +  "Durability = @durab, " +  "Description = @desc " +  "WHERE ID = @pid";  if (SelectedProc == null ||  string.IsNullOrWhiteSpace(SelectedProc.ID.ToString()) ||  string.IsNullOrWhiteSpace(SelectedProc.Name) ||  string.IsNullOrWhiteSpace(SelectedProc.Durability.ToString()) ||  string.IsNullOrWhiteSpace(SelectedProc.Price.ToString()) ||  string.IsNullOrWhiteSpace(SelectedProc.Description))  {  MessageBox.Show("Все поля должны быть заполнены и не содержать только пробелы!");  return;  }  SqlCommand cmd = new SqlCommand(query, sqlCon);  cmd.Parameters.AddWithValue("@pid", SelectedProc.ID);  cmd.Parameters.AddWithValue("@proc", SelectedProc.Name);  cmd.Parameters.AddWithValue("@durab", SelectedProc.Durability);  cmd.Parameters.AddWithValue("@price", SelectedProc.Price);  cmd.Parameters.AddWithValue("@desc", SelectedProc.Description);  SelectedProc = null;  cmd.ExecuteNonQuery();  MessageBox.Show("Данные успешно обновлены!");  Display\_Data();  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка изменения в БД! {ex.Message}");  }  finally  {  sqlCon.Close();  }  }  private void DeleteLast\_Click(object param)  {  try  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  }  string query = @"  DELETE FROM [PROCEDURE\_INFO]  WHERE [Procedure] = (  SELECT TOP 1 [Procedure]  FROM [PROCEDURE\_INFO]  ORDER BY [ID] DESC  )";  SqlCommand cmd = new SqlCommand(query, sqlCon);  int rowsAffected = cmd.ExecuteNonQuery();  if (rowsAffected > 0)  {  MessageBox.Show("Последняя запись успешно удалена!");  }  else  {  MessageBox.Show("Записи для удаления не найдены.");  }  Display\_Data();  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка при удалении! Кто-то уже купил этот тариф!");  }  finally  {  sqlCon.Close();  }  }  public void Display\_Data()  {  try  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText = "SELECT [ID] as 'ИД',[Procedure] as 'Название', [Durability] as 'Длительность', [Price] as 'Цена', [Description] as 'Описание' FROM [PROCEDURE\_INFO]";  cmd.ExecuteNonQuery();  DataTable dta = new DataTable();  SqlDataAdapter dtaAdp = new SqlDataAdapter(cmd);  dtaAdp.Fill(dta);  dataGridViewUsers1.ItemsSource = dta.DefaultView;  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка отоброжения тарифов!");  }  finally  {  sqlCon.Close();  }  }  private void SearchProcedure\_Click(object param)//поиск по названию процедуры  {  try  {  if (SearchProcName != "")  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText = "SELECT [ID] as 'ИД', [Procedure] as 'Название', [Durability] as 'Длительность', [Price] as 'Цена', [Description] as 'Описание' FROM [PROCEDURE\_INFO] WHERE [Procedure] LIKE @proc";  cmd.Parameters.AddWithValue("@proc", SearchProcName);  cmd.ExecuteNonQuery();  DataTable dt = new DataTable();  SqlDataAdapter da = new SqlDataAdapter(cmd);  da.Fill(dt);  dataGridViewUsers1.ItemsSource = dt.DefaultView;  SearchProcName = "";  }  }  else  {  MessageBox.Show("Для поиска нужно ввести слово!");  }  }  catch (Exception ex)  {  MessageBox.Show("Неверный формат");  }  finally  {  sqlCon.Close();  }  }  private void SearchPrice\_Click(object param)//поиск тарифа по цене  {  try  {  if (SearchPrice != "")  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText = "SELECT [ID] as 'ИД',[Procedure] as 'Название', [Durability] as 'Длительность', [Price] as 'Цена', [Description] as 'Описание' FROM [PROCEDURE\_INFO] WHERE [Price] = '" + SearchPrice + "'";  cmd.ExecuteNonQuery();  DataTable dt = new DataTable();  SqlDataAdapter da = new SqlDataAdapter(cmd);  da.Fill(dt);  dataGridViewUsers1.ItemsSource = dt.DefaultView;  SearchPrice = "";  }  }  else  {  MessageBox.Show("Для поиска нужно ввести слово!");  }  }  catch (Exception ex)  {  MessageBox.Show("Неверный формат");  }  finally  {  sqlCon.Close();  }  }  } |

Листинг 4: Класс AdminViewModel

|  |
| --- |
| public class AdminViewModel : INotifyPropertyChanged  {  private string \_username;  private string \_durab;  private string \_prName;  private string \_price;  private string \_description;  public string Name  {  get => \_username;  set  {  \_username = value;  OnPropertyChanged();  }  }  public string durab  {  get => \_durab;  set  {  \_durab = value;  OnPropertyChanged();  }  }  public string namePr  {  get => \_prName;  set  {  \_prName = value;  OnPropertyChanged();  }  }  public string pricePr  {  get => \_price;  set  {  \_price = value;  OnPropertyChanged();  }  }  public string descriptionPr  {  get => \_description;  set  {  \_description = value;  OnPropertyChanged();  }  }  private string \_type;  private string \_proc;  private string \_vc;  private string \_ram;  private string \_desc;  private string \_count;  public string txt\_count { get => \_count; set { \_count = value; OnPropertyChanged(); } }  public string txt\_regType { get => \_type; set { \_type = value; OnPropertyChanged(); } }  public string txt\_proc { get => \_proc; set { \_proc = value; OnPropertyChanged(); } }  public string txt\_videocard { get => \_vc; set { \_vc = value; OnPropertyChanged(); } }  public string txt\_ram { get => \_ram; set { \_ram = value; OnPropertyChanged(); } }  public string txt\_desc { get => \_desc; set { \_desc = value; OnPropertyChanged(); } }  public Frame AdFrameNavigation { get; set; }  public Frame AdFrameNavigation2 { get; set; }  public event PropertyChangedEventHandler PropertyChanged;  protected void OnPropertyChanged([CallerMemberName] string propertyName = null)  {  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));  }  public SqlConnection sqlCon = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\DataBase\stm.mdf';Integrated Security=True;Connect Timeout=30");  public Window wind { get; set; }  public ICommand AdminClose {get; }  public ICommand ChangeAcc { get; }  public ICommand MinimizeWin { get; }  public ICommand HelpBtn { get; }  public ICommand AboutBtn { get; }  public ICommand TimetableBtn { get; }  public ICommand ClientBtn { get; }  public ICommand ClientPanelBtn { get; }  public ICommand ComputerBtn { get; }  public ICommand HomePageBtn { get; }  public ICommand RegProcCommand { get; }  public ICommand ServiceCommand { get; }  public ICommand InsertComputerCommand { get; }  public AdminViewModel(string name,Window window)  {  Name = name;  this.wind = window;  AdminClose = new RelayCommand(AdminClose\_Click);  ChangeAcc = new RelayCommand(ChangeAcc\_Click);  MinimizeWin = new RelayCommand(MinimizedWin\_CLick);  HelpBtn = new RelayCommand(HelpBut\_Click);  AboutBtn = new RelayCommand(AboutBtn\_Click);  TimetableBtn = new RelayCommand(Timetable\_CLick);  ClientBtn = new RelayCommand(Client\_Click);  ClientPanelBtn = new RelayCommand(ClientsPanel\_Click);  ComputerBtn = new RelayCommand(Computer\_Click);  HomePageBtn = new RelayCommand(HomePage\_Click);  RegProcCommand = new RelayCommand(RegestrPriem\_Click);  ServiceCommand = new RelayCommand(Service\_Click);  InsertComputerCommand = new RelayCommand(InsertComputer\_Click);  }  #region =Меню=  private void AdminClose\_Click(object sender)//закрыть окно  {  wind.Close();  }  private void ChangeAcc\_Click(object sender)//поменять акк  {  LoginWindow loginWindow = new LoginWindow();  loginWindow.Show();  wind.Close();  }  private void MinimizedWin\_CLick(object sender)//кнопка свернуть окно  {  wind.WindowState = WindowState.Minimized;  }  #endregion  #region=PopUp=  private void HelpBut\_Click(object sender)//кнопка помощь в pop up  {  HelpWindow helpWindow = new HelpWindow();  helpWindow.ShowDialog();  }  private void AboutBtn\_Click(object sender)  {  AboutWindow aboutWindow = new AboutWindow();  aboutWindow.ShowDialog();  }  #endregion  #region=MainButtons=  private void Timetable\_CLick(object sender)//кнопка вызова расписания  {  AdFrameNavigation2.Visibility = Visibility.Visible;  AdFrameNavigation.Content = new TimetablePage();  AdFrameNavigation2.Content = new PersonPage();  }  private void Client\_Click(object sender)//кнопка перехода на клиенты страницу  {  AdFrameNavigation.Content = new PersonPage();  AdFrameNavigation2.Visibility = Visibility.Collapsed;  }  private void Service\_Click(object sender)//кнопка перехода на страницу с услугами  {  AdFrameNavigation.Content = new ServicePage();  AdFrameNavigation2.Visibility = Visibility.Collapsed;  }  private void ClientsPanel\_Click(object sender)  {  ModerWindow moderWindow = new ModerWindow();  moderWindow.ShowDialog();  }  private void Computer\_Click(object sender)  {  AdFrameNavigation.Content = new ComputerPage();  AdFrameNavigation2.Visibility = Visibility.Collapsed;  }  private void HomePage\_Click(object sender)//кнопка перехода на главную страницу  {  AdFrameNavigation.Content = new HomePage();  AdFrameNavigation2.Visibility = Visibility.Collapsed;  }  #endregion  TimeSpan Conver(string durab)  {  TimeSpan timedurab = new TimeSpan(0, 0, 0);  switch (durab)  {  case "10 мин":  timedurab = new TimeSpan(0, 10, 0);  break;  case "20 мин":  timedurab = new TimeSpan(0, 20, 0);  break;  case "30 мин":  timedurab = new TimeSpan(0, 30, 0);  break;  case "40 мин":  timedurab = new TimeSpan(0, 40, 0);  break;  case "50 мин":  timedurab = new TimeSpan(0, 50, 0);  break;  case "60 мин":  timedurab = new TimeSpan(1, 0, 0);  break;  case "90 мин":  timedurab = new TimeSpan(1, 30, 0);  break;  case "120 мин":  timedurab = new TimeSpan(2, 0, 0);  break;  case "180 мин":  timedurab = new TimeSpan(3, 0, 0);  break;  case "240 мин":  timedurab = new TimeSpan(4, 0, 0);  break;  case "300 мин":  timedurab = new TimeSpan(5, 0, 0);  break;  case "360 мин":  timedurab = new TimeSpan(6, 0, 0);  break;  default:  timedurab = new TimeSpan(0, 30, 0);  break;  }  return timedurab;  }  public bool IsTimeSlotOccupied()  {  bool isOccupied = false;  try  {  if (sqlCon.State == ConnectionState.Closed)  sqlCon.Open();  string query = @"SELECT COUNT(\*)  FROM [PROCEDURE\_INFO]  WHERE [Procedure] = @ProcName "  ;  SqlCommand command = new SqlCommand(query, sqlCon);  command.Parameters.AddWithValue("@ProcName", namePr);  int count = Convert.ToInt32(command.ExecuteScalar());  isOccupied = count > 0;  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка проверки: {"Ошибка при работе с БД!"}");  }  finally  {  sqlCon.Close();  }  return isOccupied;  }  private void RegestrPriem\_Click(object sender)//кнопка регестрации процедуры  {  try  {  if (IsTimeSlotOccupied())  {  MessageBox.Show("Выбранный тариф уже занят. Выберите другое название.");  return;  }  if(string.IsNullOrEmpty(namePr) || string.IsNullOrEmpty(pricePr) || string.IsNullOrEmpty(descriptionPr) || string.IsNullOrEmpty(durab) )  {  MessageBox.Show("Введите все данные корректно!");  return;  }  if (int.Parse(pricePr) < 0)  {  MessageBox.Show("Введите все данные корректно!");  return;  }  if(namePr.Length > 30)  {  MessageBox.Show("Длина не может быть больше 30!");  return;  }  if (sqlCon.State == ConnectionState.Closed)  sqlCon.Open();  String quer = "INSERT INTO [PROCEDURE\_INFO] ([Durability], [Procedure], [Price], [Description]) values (@durab, @proced, @price, @descrep)";  SqlCommand cmd = new SqlCommand(quer, sqlCon);  cmd.CommandType = CommandType.Text;  cmd.Parameters.AddWithValue("@durab", Conver(durab));  cmd.Parameters.AddWithValue("@proced", namePr);  cmd.Parameters.AddWithValue("@price", pricePr);  cmd.Parameters.AddWithValue("@descrep", descriptionPr);  cmd.ExecuteNonQuery();  durab = "";  namePr = "";  pricePr = "";  descriptionPr = "";  MessageBox.Show("Занесено!");  }  catch (Exception ex)  {  MessageBox.Show("Данные введены не корректно!");  }  finally  {  sqlCon.Close();  }  }  #region =Computer=  private void InsertComputer\_Click(object sender)//добавление компа  {  try  {  int count;  if (sqlCon.State == ConnectionState.Closed)  {  if (txt\_regType != "" && txt\_proc != "" && txt\_ram != "" && txt\_videocard != "" && txt\_count != "" && Int32.TryParse(txt\_count, out count))//проверка на пустые строки and int  {  if (count > 15 || count < 0)  {  MessageBox.Show("Кол-во не может быть >15 и <0");  txt\_count = "";  return;  }  sqlCon.Open();  String quer2 = "INSERT INTO [COMPUTER\_INFO]values (@type, @proc, @videocard, @ram, @desc,@count)";  SqlCommand cmd2 = new SqlCommand(quer2, sqlCon);  cmd2.CommandType = CommandType.Text;  cmd2.Parameters.AddWithValue("@type", txt\_regType);  cmd2.Parameters.AddWithValue("@proc", txt\_proc);  cmd2.Parameters.AddWithValue("@videocard", txt\_videocard);  cmd2.Parameters.AddWithValue("@ram", txt\_ram);  cmd2.Parameters.AddWithValue("@desc", txt\_desc);  cmd2.Parameters.AddWithValue("@count", txt\_count);  cmd2.ExecuteNonQuery();  MessageBox.Show("Congrulation!");  sqlCon.Close();  txt\_regType = "";  txt\_proc = "";  txt\_ram = "";  txt\_videocard = "";  txt\_desc = "";  txt\_count = "";  }  else  {  MessageBox.Show("Все поля доджны быть заполнены!");  }  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при работе с БД!");  }  finally  {  sqlCon.Close();  }  }  #endregion  } |

Листинг 5: Класс ComputerViewModel

|  |
| --- |
| public class ComputerViewModel : INotifyPropertyChanged  {  public DataGrid \_dataGridViewUsers1;private string \_searchProcessor;  private string \_searchType;private Computer \_selectedComputer = new Computer();  public Computer SelectedComputer  {  get => \_selectedComputer;  set  {  \_selectedComputer = value;  OnPropertyChanged(nameof(SelectedComputer));  }  }  public DataGrid dataGridViewUsers1  {  get => \_dataGridViewUsers1;  set  {  \_dataGridViewUsers1 = value;  OnPropertyChanged(nameof(SearchProcessor));  }  }public string SearchProcessor  {  get => \_searchProcessor;  set  {  \_searchProcessor = value;  OnPropertyChanged(nameof(SearchProcessor));  }  }  public string SearchType  {  get => \_searchType;  set  {  \_searchType = value;  OnPropertyChanged(nameof(SearchType));  }  }public SqlConnection sqlCon = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\DataBase\stm.mdf';Integrated Security=True;Connect Timeout=30");//подключение бдpublic event PropertyChangedEventHandler PropertyChanged;  protected void OnPropertyChanged([CallerMemberName] string propertyName = null)  {  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));  }public ICommand RefreshCommand { get; set; }  public ICommand SearchProcessorCommand { get; set; }  public ICommand SearchTypeCommand { get; set; }  public ICommand DeleteLastCommand { get; set; }  public ICommand UpdateCommand { get; }  public ComputerViewModel()  {  RefreshCommand = new RelayCommand(Refuse\_Click);  SearchProcessorCommand = new RelayCommand(SearchProcessor\_Click);  SearchTypeCommand = new RelayCommand(SearchType\_Click);  DeleteLastCommand = new RelayCommand(DeleteLast\_Click);  UpdateCommand = new RelayCommand(UpdateComputer);  }private void Refuse\_Click(object param)//обновить  {  Display\_Data();  }private void UpdateComputer(object param)  {  try  {  if (sqlCon.State == ConnectionState.Closed)  sqlCon.Open();  if (string.IsNullOrEmpty(SelectedComputer.Type) ||  SelectedComputer.Id == 0||  SelectedComputer.Id == null ||  string.IsNullOrEmpty(SelectedComputer.Processor) ||  string.IsNullOrEmpty(SelectedComputer.Videocard) ||  string.IsNullOrEmpty(SelectedComputer.RAM) ||  string.IsNullOrEmpty(SelectedComputer.CompDesc) ||  SelectedComputer.Count <= 0 ||  SelectedComputer.Count > 15)  {  MessageBox.Show("Все поля должны быть заполнены, а количество должно быть больше нуля и меньше 15!");  return;  }  string query = "UPDATE [COMPUTER\_INFO] SET " +  "comp\_type = @Type, " +  "comp\_processor = @Processor, " +  "comp\_videocard = @Videocard, " +  "comp\_ram = @RAM, " +  "comp\_desc = @Description, " +  "Count = @Count " +  "WHERE computerId = @cid";  SqlCommand cmd = new SqlCommand(query, sqlCon);  cmd.Parameters.AddWithValue("@cid", SelectedComputer.Id);  cmd.Parameters.AddWithValue("@Type", SelectedComputer.Type);  cmd.Parameters.AddWithValue("@Processor", SelectedComputer.Processor);  cmd.Parameters.AddWithValue("@Videocard", SelectedComputer.Videocard);  cmd.Parameters.AddWithValue("@RAM", SelectedComputer.RAM);  cmd.Parameters.AddWithValue("@Description", SelectedComputer.CompDesc);  cmd.Parameters.AddWithValue("@Count", SelectedComputer.Count);  SelectedComputer = null;cmd.ExecuteNonQuery();  MessageBox.Show("Данные успешно обновлены!");  Display\_Data();  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка изменения в БД!");  }  finally  {  sqlCon.Close();  }  }private void DeleteLast\_Click(object param)  {  try  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  }  string query = @"  DELETE FROM [COMPUTER\_INFO]  WHERE [computerId] = (  SELECT TOP 1 [computerId]  FROM [COMPUTER\_INFO]  ORDER BY [computerId] DESC  )";  SqlCommand cmd = new SqlCommand(query, sqlCon);  int rowsAffected = cmd.ExecuteNonQuery();  if (rowsAffected > 0)  {  MessageBox.Show("Последняя запись успешно удалена!");  }  else  {  MessageBox.Show("Записи для удаления не найдены.");  }  Display\_Data();  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка при удалении! Кто-то уже забронировал этот компьютер!");  }  finally  {  sqlCon.Close();  }  }  public void Display\_Data()  {  try  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText = "SELECT [computerId] as 'ИД',[comp\_type] as 'Тип', [comp\_processor] as 'Процессор', [comp\_videocard] as 'Видеокарта', [comp\_ram] as 'ОЗУ', [comp\_desc] as 'Описание' , [Count] as 'Колличество' FROM [COMPUTER\_INFO]";  cmd.ExecuteNonQuery();  DataTable dta = new DataTable();  SqlDataAdapter dtaAdp = new SqlDataAdapter(cmd);  dtaAdp.Fill(dta);  dataGridViewUsers1.ItemsSource = dta.DefaultView;  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка отоброжения тарифов!");  }  finally  {  sqlCon.Close();  }  }private void SearchProcessor\_Click(object param)//поиск по названию процедуры  {  try  {  if (SearchProcessor != "")  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText = "SELECT [computerId] as 'ИД',[comp\_type] as 'Тип', [comp\_processor] as 'Процессор', [comp\_videocard] as 'Видеокарта', [comp\_ram] as 'ОЗУ', [comp\_desc] as 'Описание' , [Count] as 'Колличество' FROM [COMPUTER\_INFO] WHERE [comp\_processor] LIKE @proc";  cmd.Parameters.AddWithValue("@proc", SearchProcessor);  cmd.ExecuteNonQuery();  DataTable dt = new DataTable();  SqlDataAdapter da = new SqlDataAdapter(cmd);  da.Fill(dt);  dataGridViewUsers1.ItemsSource = dt.DefaultView;  SearchProcessor = "";  }  }  else  {  MessageBox.Show("Для поиска нужно ввести слово!");  }  }  catch (Exception ex)  {  MessageBox.Show("Неверный формат");  }  finally  {  sqlCon.Close();  }  }  private void SearchType\_Click(object param)  {  try  {  if (SearchType != "")  {  if (sqlCon.State == ConnectionState.Closed)  {  sqlCon.Open();  SqlCommand cmd = sqlCon.CreateCommand();  cmd.CommandType = CommandType.Text;  cmd.CommandText = "SELECT [computerId] as 'ИД',[comp\_type] as 'Тип', [comp\_processor] as 'Процессор', [comp\_videocard] as 'Видеокарта', [comp\_ram] as 'ОЗУ', [comp\_desc] as 'Описание', [Count] as 'Колличество' FROM [COMPUTER\_INFO] WHERE [comp\_type] ='" + SearchType + "'";  cmd.ExecuteNonQuery();  DataTable dt = new DataTable();  SqlDataAdapter da = new SqlDataAdapter(cmd);  da.Fill(dt);  dataGridViewUsers1.ItemsSource = dt.DefaultView;  SearchType = "";  }  }  else  {  MessageBox.Show("Для поиска нужно ввести слово!");  }  }  catch (Exception ex)  {  MessageBox.Show("Неверный формат");  }  finally  {  sqlCon.Close();  }  }  } |

Листинг 5: Класс UserViewModel

|  |
| --- |
| public class UserViewModel : INotifyPropertyChanged  {  private readonly Window \_window;  private string \_username;  private string \_computer;  private string \_date;  private string \_user;  private string \_procedure;  public string regUserLogin { get=> \_username; set { \_username = value; OnPropertyChanged(); } }  public string Computer  {  get => \_computer;  set  {  \_computer = value;  OnPropertyChanged();  }  }  public string Procedure  {  get => \_procedure;  set  {  \_procedure = value;  OnPropertyChanged();  }  }  public ICommand DragAdminWinCommand { get; }  public ICommand AdminCloseCommand { get; }  public ICommand ChangeAccCommand { get; }  public ICommand MinimizedWinCommand { get; }  public ICommand HelpButCommand { get; }  public ICommand AboutBtnCommand { get; }  publicICommand AllPcCommand { get; }  public ICommand AllProceCommand { get; }  public List<Computer> computers { get;set; }  public List<User> users { get;set; }  public List<Procedure> procedures { get;set; }  public SqlConnection connection = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\DataBase\stm.mdf';Integrated Security=True;Connect Timeout=30");  public event PropertyChangedEventHandler PropertyChanged;  protected void OnPropertyChanged([CallerMemberName] string propertyName = null)  {  PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));  }  public UserViewModel(string regUserLogin,Window window/\*List<Computer> \_computers,List<User> \_users,List<Procedure> \_procedures\*/)  {    this.regUserLogin = regUserLogin;  this.\_window = window;  //this.computers = \_computers;  //this.users = \_users;  //this.procedures = \_procedures;  computers = new List<Computer>();  users = new List<User>();  procedures = new List<Procedure>();  DragAdminWinCommand = new RelayCommand(ExecuteDragAdminWin);  AdminCloseCommand = new RelayCommand(ExecuteAdminClose);  ChangeAccCommand = new RelayCommand(ExecuteChangeAcc);  MinimizedWinCommand = new RelayCommand(ExecuteMinimizedWin);  HelpButCommand = new RelayCommand(ExecuteHelpBut);  AboutBtnCommand = new RelayCommand(ExecuteAboutBtn);  AllPcCommand = new RelayCommand(ExecuteAllPcWind);  AllProceCommand = new RelayCommand(ExecuteAllProceWind);  }  public void StartWorking()  {  #region bd подключение компьютера  //подключить  try  {  if (connection.State == ConnectionState.Closed)  connection.Open();  String query = "SELECT [COMPUTER\_INFO].[computerId],[COMPUTER\_INFO].[comp\_type] FROM [COMPUTER\_INFO]";  SqlCommand command = new SqlCommand(query, connection);  command.CommandType = CommandType.Text;  SqlDataReader reader = command.ExecuteReader();  while (reader.Read())  {  int Id = reader.GetInt32(0);  string type = reader.GetString(1);  Computer computer = new Computer(Id, type);  computers.Add(computer);  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при работе с БД!");  }  finally  {  connection.Close();  }  #endregion  #region bd подключенеи юзера  try  {  if (connection.State == ConnectionState.Closed)  connection.Open();  String query = "SELECT [USER\_INFO].[ID],[USER\_INFO].[Name],[USER\_INFO].[Surname], [USER\_INFO].[Patronymic] FROM [USER\_INFO]";  SqlCommand command = new SqlCommand(query, connection);  command.CommandType = CommandType.Text;  SqlDataReader reader = command.ExecuteReader();  while (reader.Read())  {  int Id = reader.GetInt32(0);  string Name = reader.GetString(1);  string Surname = reader.GetString(2);  string Patronymic = reader.GetString(3);  User user = new User(Id, Surname + " " + Name + " " + Patronymic);//фио  users.Add(user);  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при работе с БД!");  }  finally  {  connection.Close();  }  #endregion//доделать поиск юзеров  #region bd подключение процедур  try  {  if (connection.State == ConnectionState.Closed)  connection.Open();  String query = "SELECT [PROCEDURE\_INFO].[Procedure], [PROCEDURE\_INFO].[ID]FROM [PROCEDURE\_INFO]";  SqlCommand command = new SqlCommand(query, connection);  command.CommandType = CommandType.Text;  SqlDataReader reader = command.ExecuteReader();  while (reader.Read())  {  string Name = reader.GetString(0);  int ID = reader.GetInt32(1);  Procedure procedure = new Procedure(Name, ID);  procedures.Add(procedure);  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при работе с БД!");  }  finally  {  connection.Close();  }  #endregion  }  #region =Меню=  private void ExecuteDragAdminWin(object parameters)//для перемещения окна  {  \_window.DragMove();  }  private void ExecuteAdminClose(object parameters)//закрыть окно  {  \_window.Close();  }  private void ExecuteChangeAcc(object parameters)//поменять акк  {  LoginWindow loginWindow = new LoginWindow();  loginWindow.Show();  \_window.Close();  }  private void ExecuteAllPcWind(object parameters)//поменять акк  {  AllComputers allPc = new AllComputers();  allPc.ShowDialog();  }  private void ExecuteAllProceWind(object parameters)//поменять акк  {  AllProces allProce = new AllProces();  allProce.ShowDialog();  }  private void ExecuteMinimizedWin(object parameters)  {  \_window.WindowState = WindowState.Minimized;  }  #endregion  #region=PopUp=  private void ExecuteHelpBut(object parameters)//кнопка помощь в pop up  {  HelpWindow helpWindow = new HelpWindow();  helpWindow.ShowDialog();  }  private void ExecuteAboutBtn(object parameters)  {  AboutWindow aboutWindow = new AboutWindow();  aboutWindow.ShowDialog();  }  #endregion  #region Convert  public int IdProc(string prsSrch) //срабатывает регистрации тарифов, находит по строке нужны ай ди из списков  {  int idProc = 0;  if (procedures != null && procedures.Count != 0)  {  for (int i = 0; i < procedures.Count; i++)  {  if (procedures[i].Name.ToString() == prsSrch)  idProc = procedures[i].ID;  }  }  return idProc;  }  public int UserId()  {  int userId = 0;  if (connection.State == ConnectionState.Closed)  connection.Open();  string query = "SELECT [ID] FROM [USER\_INFO] WHERE [UserId] = @Login";  SqlCommand command = new SqlCommand(query, connection);  command.Parameters.AddWithValue("@Login", regUserLogin);  object result = command.ExecuteScalar();  if (result != null)  {  userId = Convert.ToInt32(result);  }  else  {  MessageBox.Show("Пользователь с указанным логином не найден.");  }  return userId;  }  public int ComputerId()  {  int dc = 0;  for (int i = 0; i < computers.Count; i++)  {  if (computers[i].Type == Computer)  dc = computers[i].Id;  }  return dc;  }  #endregion  public TimeSpan TimeArriveProc(string time)  {  TimeSpan timeStartProc = new TimeSpan(0, 0, 0);  if (TimeSpan.TryParse(time, out TimeSpan parsedTime))  {  if (parsedTime >= new TimeSpan(10, 0, 0) && parsedTime <= new TimeSpan(24, 0, 0) && parsedTime.Minutes % 30 == 0)  {  timeStartProc = parsedTime;  }  }  return timeStartProc;  }  public TimeSpan TimeOutProc(string time)  {  TimeSpan timeOutProc = new TimeSpan(0, 0, 0);  if (TimeSpan.TryParse(time, out TimeSpan parsedTime))  {  if (parsedTime >= new TimeSpan(10, 0, 0) && parsedTime <= new TimeSpan(24, 0, 0) && parsedTime.Minutes % 30 == 0)  {  timeOutProc = parsedTime;  }  }  return timeOutProc;  }  public TimeSpan TimeStartProc(string time)  {  TimeSpan timeStartProc = new TimeSpan(0, 0, 0);  switch (time)  {  case "PC1":  timeStartProc = new TimeSpan(10, 0, 0);  break;  case "PC2":  timeStartProc = new TimeSpan(11, 0, 0);  break;  case "PC3":  timeStartProc = new TimeSpan(12, 0, 0);  break;  case "PC4":  timeStartProc = new TimeSpan(13, 0, 0);  break;  case "PC5":  timeStartProc = new TimeSpan(14, 0, 0);  break;  case "PC6":  timeStartProc = new TimeSpan(15, 0, 0);  break;  case "PC7":  timeStartProc = new TimeSpan(16, 0, 0);  break;  case "PC8":  timeStartProc = new TimeSpan(17, 0, 0);  break;  case "PC9":  timeStartProc = new TimeSpan(18, 0, 0);  break;  case "PC10":  timeStartProc = new TimeSpan(19, 0, 0);  break;  case "PC11":  timeStartProc = new TimeSpan(20, 0, 0);  break;  case "PC12":  timeStartProc = new TimeSpan(21, 0, 0);  break;  case "PC13":  timeStartProc = new TimeSpan(22, 0, 0);  break;  case "PC14":  timeStartProc = new TimeSpan(23, 0, 0);  break;  case "PC15":  timeStartProc = new TimeSpan(24, 0, 0);  break;  }  return timeStartProc;  }  public TimeSpan GetProcedureDuration(int procedureId)  {  TimeSpan duration = TimeSpan.Zero;  try  {  if (connection.State == ConnectionState.Closed)  connection.Open();  string query = @"SELECT [Durability] FROM [PROCEDURE\_INFO] WHERE [ID] = @ProcedureId";  SqlCommand sqlCmd = new SqlCommand(query, connection);  sqlCmd.Parameters.AddWithValue("@ProcedureId", procedureId);  var result = sqlCmd.ExecuteScalar();  if (result != null && TimeSpan.TryParse(result.ToString(), out duration))  {  return duration;  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при получении длительности процедуры: " + "Ошибка при работе с БД!");  }  finally  {  connection.Close();  }  return duration;  }  public bool IsTimeSlotOccupied(int computerId, DateTime date, TimeSpan timeArrive, TimeSpan timeOut,TimeSpan timeStart)  {  bool isOccupied = false;  try  {  // Проверяем, что время выхода не раньше времени прибытия  if (timeOut <= timeArrive)  {  MessageBox.Show("Время выхода не может быть раньше или равно времени прибытия!");  return true;  }  if (connection.State == ConnectionState.Closed)  connection.Open();  // Мы ищем все записи, у которых время прибытия и выхода пересекаются с новым интервалом  string query = @"SELECT TimeArrive, TimeOut  FROM [SCHEDULE]  WHERE [ComputerId] = @ComputerId  AND [Date] = @Date  AND [TimeStart] = @TimeStart";  SqlCommand command = new SqlCommand(query, connection);  command.Parameters.AddWithValue("@ComputerId", computerId);  command.Parameters.AddWithValue("@Date", date);  command.Parameters.AddWithValue("@TimeStart", timeStart);  SqlDataReader reader = command.ExecuteReader();  while (reader.Read())  {  TimeSpan existingArrive = reader.GetTimeSpan(0);// Время прибытия существующей записи  TimeSpan existingOut = reader.GetTimeSpan(1); // Время выхода существующей записи  // Проверка на пересечение интервалов  // Новый интервал [timeArrive, timeOut] не должен пересекаться с уже существующим  if ((timeArrive < existingOut && timeOut > existingArrive))  {  isOccupied = true;  break; // Если хотя бы одно пересечение найдено, выходим из цикла  }  }  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка проверки занятости времени: {"Ошибка при работе с БД!"}");  }  finally  {  connection.Close();  }  return isOccupied;  }  } |