МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Новосибирский национальный исследовательский государственный университет»
(Новосибирский государственный университет, НГУ)
Структурное подразделение Новосибирского государственного университета – Высший колледж информатики Университета (ВКИ НГУ)
КАФЕДРА ИНФОРМАТИКИ

Отчет по курсовому проекту

ПМ 01. Разработка программных модулей

РАЗРАБОТКА ПРИЛОЖЕНИЯ «РЕСТОРАННЫЙ БИЗНЕС» НА ПЛАТФОРМЕ WPF

Руководитель Голкова Н.В.
«27» декабря 2023 г.

Студент 3 курса Денисова К.И.
гр. 107а1 «27» декабря 2023 г.

Новосибирск
2023

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

XAML – язык разметки, который появился вместе с первой версией WPF от Microsoft. Он используется для инициализации объектов в технологиях на платформе .NET.

Windows Presentation Foundation (WPF) – платформа пользовательского интерфейса, которая не зависит от разрешения и использует векторный механизм визуализации.

База данных (БД) - это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе.

Система управления базами данных (СУБД) – комплекс программно-языковых средств, позволяющих создать базы данных и манипулировать данными.

Программное средство (ПС) – компьютерная программа или логически связанная совокупность программ, предназначенная для автоматизации в определённой области профессиональной деятельности.

Десктопное приложение — программа, которая устанавливается на компьютер пользователя и работает под управлением операционной системы. Такие приложения высокопроизводительные, могут работать напрямую с принтерами, сканерами, факсами и прочей техникой.

Операционная система (ОС) — это специальный набор программ, благодаря которому все системы компьютера взаимодействуют как между собой, так и с пользователем.

MVVM – паттерн разработки, позволяющий разделить приложение на три функциональные части: Model — основная логика программы, View — вид или представление, ViewModel — модель представления, которая служит прослойкой между другими двумя частями.

ВВЕДЕНИЕ

В современном мире всё большую популярность набирает досуг вне дома. Ресторанный бизнес – одно из самых стремительно развивающихся направлений в сфере обслуживания. Тяжело представить какой-либо праздник без похода в ресторан с семьей или друзьями. Даже в будние дни многие работники предпочитают провести обеденный перерыв в кафе, наслаждаясь приятной атмосферой. Именно повышенный интерес к изысканным ресторанным блюдам и увеличение количества посетителей вызывает необходимость автоматизации ресторанного бизнеса.

Для удовлетворения потребностей посетителей в ресторане должна быть налажена работа персонала. Но контролировать все процессы в заведении возможно только ежедневно храня и обрабатывая огромный объем информации. Необходимо отслеживать поставки продуктов, а также своевременно пополнять их запасы и хранить данные об их применении. К счастью, в век высоких технологий огромное распространение получили программные средства для бизнеса.

Обеспечить правильное функционирование ресторана можно путем разработки базы данных, в которой будет храниться вся информация, и пользовательского интерфейса, позволяющего манипулировать данными. Автоматизация составления заказов, приема поставок и отслеживания расхода продуктов позволит сотрудникам сделать упор на качестве работы, а значит улучшить сервис и увеличить прибыль. Использование программных средств упрощает ведение бизнеса, освобождая время для разработки стратегий развития.

Целью проекта является разработка программного средства для автоматизации ресторанного бизнеса.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- Произвести анализ предметной области;
- Изучить характеристики программных сред и средств;
- Спроектировать базу данных;
- Спроектировать и реализовать архитектуру десктопного приложения с учетом всех требований;
- Провести тестирование программного средства.
- Написать руководство пользователя: создать подробное руководство по использованию разработанного приложения.
- Написать пояснительную записку: документирование процесса разработки, принятых решений и результатов.

Таким образом, разработка программного продукта для ресторана предоставит решение для оптимизации бизнес-процессов и улучшения взаимодействия с поставщиками, что сделает работу ресторана более эффективной.

1 ПОСТАНОВКА ЗАДАЧИ ВКР

Цель работы – разработать базу данных и пользовательский интерфейс для ресторана, позволяющие автоматизировать работу кухни, используя СУБД Microsoft SQL Server и технологию WPF. Программное средство должно отвечать бизнес требованиям, пользовательским и системным требованиям, требованиям к пользовательскому интерфейсу.

1.1 Бизнес требования

Разработка программного продукта в современном мире неотделима от четкого определения его целевой аудитории и бизнес-целей. В данном разделе представлен анализ целевой аудитории и бизнес-целей приложения «Ресторанный бизнес».

Данное программное средство предназначено для использования работниками ресторана, занимающими различные должности. Приложение разрабатывается для конкретного ресторана, поэтому доступ к данным и программному средству имеют толь сотрудники заведения. Десктопное приложение не подразумевает возможность регистрации, открывать доступ к программному средству может только пользователь, обладающий правами администратора (в данном случае шеф-повар ресторана).

Бизнес-цели:

- Оптимизация процессов поставки. Подразумевает упрощение составления заказов и приема продуктов от поставщиков.
- Повышение эффективности, а именно ускорение процесса документирования, позволяющее сконцентрироваться на качестве оказания услуг.
- Безопасность. Шеф-повар может отслеживать качество работы сотрудников и проверять точное количество продуктов на складе.

Ожидается, что разработка и внедрение данного приложения приведут к заметному росту эффективности работы ресторана, улучшению сервиса и достижению поставленных бизнес-целей.

1.2 Пользовательские требования

Рассмотрим пользовательские требования к системе «Ресторанный бизнес», которые направлены на создание удобного и эффективного инструмента для сотрудников. Все работник должны иметь возможность взаимодействия с системой:

1. Авторизация:
    - Должна быть предусмотрена возможность входа в систему с использованием зарегистрированных учетных данных.
2. Просмотр продуктов на складе:
    - Пользователь должен видеть информацию о продуктах в наличии, а именно количество и дату поставки.
3. Выбор и удаление продукта из системы:
    - Возможность взятие продукта со склада.
    - Удаление продукта по истечении срока годности.
    - Система должна предоставлять подтверждение успешного изменения данных.

Су-шеф ресторана должен обладать расширенные возможности взаимодействия с программой, а именно:

4. Заказ продуктов:
    - Просмотр списка поставщиков и поставляемых ими продуктов.
    - Составление и отправление заявки на поставку продуктов.
    - Подтверждение системой успешно выполненного заказа.

Самым широким кругом возможностей использования приложения должен обладать шеф-повар, исполняющий обязанности администратора программного средства:

5. Прием поставок:
    - Просмотр информации о поставке и списка доставленных продуктов.
    - Добавление продуктов на склад.
    - Системное подтверждение окончания приема продуктов.
6. Просмотр данных сотрудников ресторана:
    - Подразумевает доступ к личным данным сотрудников.
    - Возможность удаления сотрудника, то есть ограничения его возможностей входа в приложение.
7. Добавление новых пользователей:
    - Ввод в систему данных о новом сотруднике.
    - Предоставление логина и пароля для доступа к приложению.

Следовательно, все вышеописанные требования направлены на обеспечение удобства пользования системой. Этот раздел требований представляет собой фундаментальный набор функций, которые будут реализованы в системе «Ресторанный бизнес».

1.3 Системные требования

Системные требования представляют собой неотъемлемую часть процесса разработки программного продукта, определяя минимальные и рекомендуемые характеристики для его корректного функционирования.

Ключевые параметры:

- Платформа. Операционная система, на которой должно быть установлено приложение, должна быть Windows 10 (64-bit) или более поздняя версия.

- Процессор. Для нормального функционирования необходим процессор Intel Core i3 с тактовой частотой 2.0 ГГц.
- Графический адаптер: Поддерживается интегрированный графический процессор.

Таким образом, обеспечение оптимальной работы программного продукта напрямую зависит от соответствия технических характеристик вашего устройства указанным системным требованиям.

1.4 Требования к графическому пользовательскому интерфейсу

Пользовательский интерфейс должен быть визуально лаконичным и удобным в использовании для всех категорий пользователей, а так же отвечать следующим требованиям:

1. Внешний вид пользовательского интерфейса:
    - Использование фирменных цветов ресторана, для которого разрабатывается приложение, а именно оттенки бордового, золотого и белого.
    - Использование логотипа ресторана. На рисунке 1 представлен логотип Ресторана «Морозово»:

Рисунок 1 – Логотип Ресторана «Морозово»

    - Чётко читаемые шрифты.
    - Изображения высокого качества.

2. Доступ к функциональности системы:
    - Лёгкая и понятная навигация по разделам приложения.
    - Удобные формы для ввода данных.
    - Выделение активных элементов для повышения внимания пользователя.

Графический пользовательский интерфейс разрабатывается с учетом эстетики салона красоты и должен обеспечивать удобство взаимодействия для пользователей.

2 АНАЛИЗ ТРЕБОВАНИЙ И ОПРЕДЕЛЕНИЕ СПЕЦИФИКАЦИЙ

2.1 Описание предметной области задачи ВКР

Ресторанный бизнес отличается от всех остальных видов бизнеса. Это предприятие, которое объединяет в себе искусство и традиции, механизмы деятельности и опыт маркетологов, философию обслуживания и концепцию формирования потенциальной аудитории.

С каждым годом ресторанный бизнес стремительно развивается. Идет серьезная конкурентная борьба за посетителей. Именно этот фактор заставляет директоров продумывать не только основную стратегию и стиль деятельности ресторана, но и детали, придающие заведению уникальность и неповторимость.

Для успешного функционирования ресторанного учреждения важно качество блюд, меню, уровень обслуживания, цена, атмосфера и менеджмент. Кроме того, должна быть налажена связь с поставщиками продуктов питания, а также система хранения продуктов в самом заведении. В ресторане ведется учет всех продуктов, хранящихся на складе. Чтобы контролировать работу персонала, в журнале должна фиксироваться информация о том, какие продукты берут сотрудники и в каком количестве. Также необходимо своевременно заказывать продукты у поставщиков, во избежание нехватки необходимых ингредиентов. Для этого шеф-повар ресторана или исполняющий его обязанности составляет заявку, в которой указывает необходимые продукты.

После поставки сверяются все привезенные продукты, после чего они вносятся в список продуктов на складе. Качество еды напрямую зависит от свежести ингредиентов, из которых она приготовлена. Поэтому в заведении должны внимательно следить за сроками годности продуктов, списывая и перерабатывая испортившиеся.

Кроме того, через кухню происходит заказ продукции для бара ресторана. Работа официантов и барменов также отслеживается по журналу, в котором указываются взятые продукты.

Во многих ресторанах подбором персонала занимается шеф-повар, формируя профессиональную и слаженную команду поваров. Именно высокая квалификация работников и качество блюд привлекает гостей в ресторан, а автоматизация процессов в заведении позволяет наладить качественную работу ресторана.

База данных, разрабатываемая для ресторанного бизнеса, должна отражать специфику данной предметной области и обеспечивать удобное и надежное хранение информации. Таким образом, предметом разработки является приложение для манипулирования данными, представляющее собой электронный журнал.

2.2 Классы и характеристики пользователей

Разрабатываемый пользовательский интерфейс для базы данных ресторана должен быть доступен всем работникам заведения, не зависимо от должности. Можно выделить несколько классов пользователей:

1. Работники с базовым уровнем доступа, а именно повара, заготовщики, официанты и бармены. Данные должности не подразумевают поддержание связи с поставщиками и контроль продуктов на складе.

2. Су-шеф, обладающий большим кругом обязанностей. Он контролирует составление заявок поставщикам.

3. Шеф-повар. Кроме всех вышеперечисленных обязанностей, он отвечает за поставки продуктов, контролирование продуктов на складе, а так же за подбор персонала.

Таким образом, классы пользователей выделяются по занимаемой должности и кругу должностных обязанностей.

2.3 Функциональные требования

Функциональные требования для разных классов пользователей отличаются. Но ряд возможностей приложения доступны для всех пользователей, а именно:

- Возможность свернуть приложение;
- Выход из приложения;
- Авторизация по логину и паролю;
- Просмотр списка продуктов на складе;
- Просмотр продуктов по категориям;
- Взятие определенного продукта.

Войти в приложение могут лишь сотрудники, внесенные в базу данных ресторана.

Помимо вышеперечисленных возможностей, су-шеф обладает правами:

- Удалять продукт из списка продуктов на складе;
- Просматривать предложенные для заказа продукты поставщиков;
- Выбирать необходимые продукты у поставщика и формировать заявку.

К исключительным правам шеф-повара относятся:

- Принятие поставки;
- Добавление продуктов на склад;
- Просмотр всех сотрудников;
- Удаление сотрудника в случае увольнения;
- Добавление сотрудника при приеме на работу.

Наиболее полными правами пользования приложением обладает шеф-повар, так как он несет ответственность за работу всей кухни. Приложение рассчитано на ограниченный круг людей, поэтому доступ к данным пользователь получает только после приема на работу.

На рисунке 2 представлена диаграмма вариантов использования приложения разными классами пользователей.

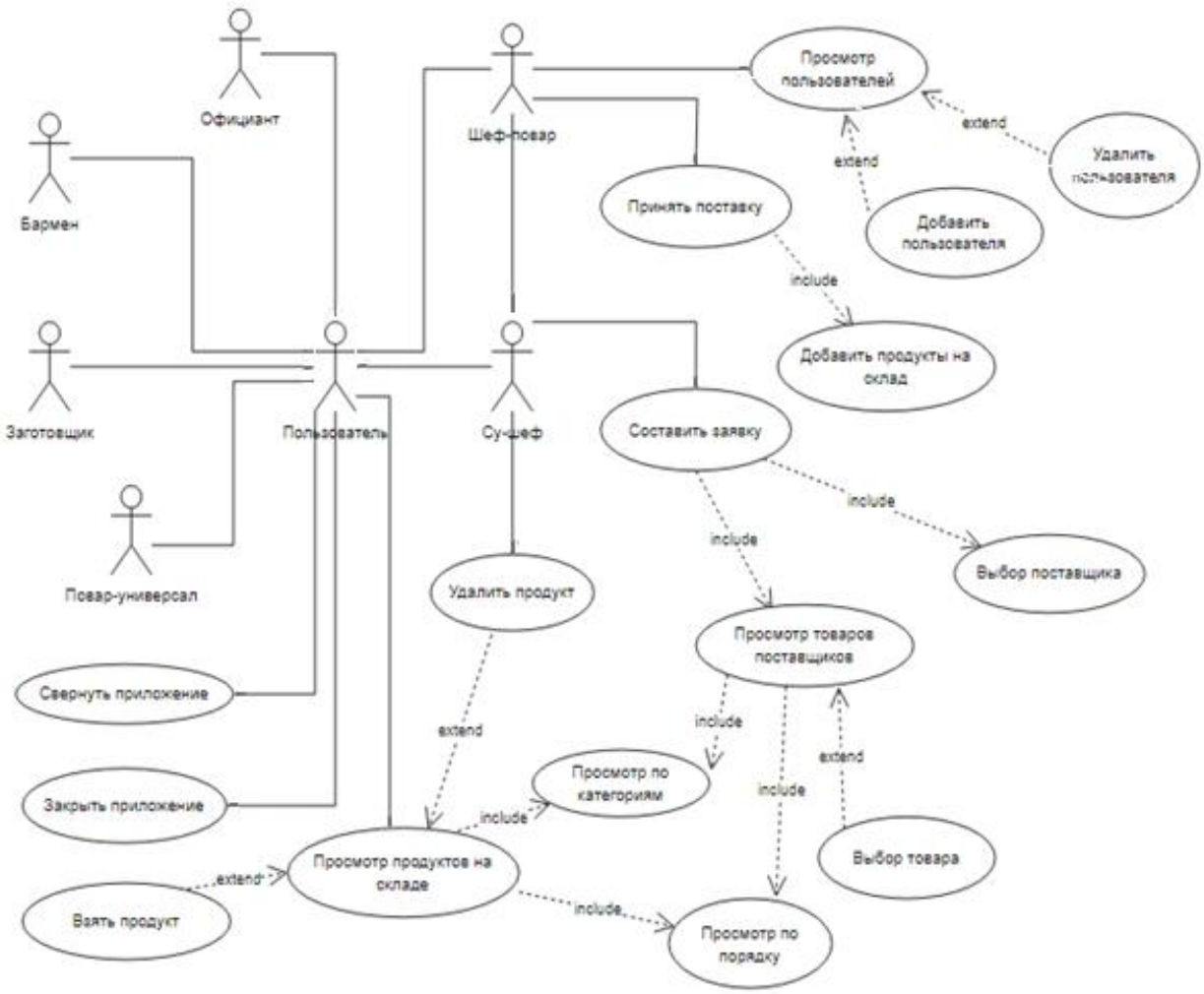


Рисунок 2 – UML-диаграмма прецедентов

Таким образом, был представлен комплекс ключевых возможностей, направленных на обеспечение удовлетворения потребностей пользователей.

2.4 Нефункциональные требования

Приложение «Ресторанный бизнес» должно соответствовать следующим нефункциональным требованиям:

- Производительность. Система должна предоставлять отклик на запросы пользователей в течение не более 2 секунд.

- Переносимость. Приложение должно корректно функционировать на последних версиях операционной системы Windows, включая Windows 10.
- Надежность. Не должно быть сбоев в работе программы.
- Безопасность. Пользовательские данные должны храниться в базе данных, система должна иметь механизм авторизации пользователей.
- Удобство использования. Интерфейс должен быть интуитивно понятным и лёгким в использовании.

Таким образом, нефункциональные требования направлены на обеспечение эффективности, устойчивости, безопасности и удобства использования приложения.

3 ВЫБОР ПРОГРАММНЫХ СРЕД И СРЕДСТВ РАЗРАБОТКИ

Основная цель любого инструмента разработки программного обеспечения - создание программного обеспечения путем помощи в написании кода без ошибок или с помощью простого в использовании пользовательского интерфейса. Выбор инструментов программирования и языков программирования - это комбинация множества различных факторов. Выбранные среда и средства разработки должны отвечать поставленным требованиям.

3.1 Сравнительный анализ имеющихся возможностей по выбору средств разработки

В мире программирования пользовательский интерфейс играет ключевую роль, определяя визуальное взаимодействие пользователя с приложением. Для разработки пользовательского интерфейса в Windows-приложениях используются различные технологии, две из которых на сегодняшний день являются наиболее популярными — это Windows Presentation Foundation (WPF) и Windows Forms.

В таблице 1 представлена сравнительная характеристика технологий WPF и Windows Forms.

Таблица 1 – Сравнительная характеристика технологий разработки.

Характеристика	WPF	Windows Forms
Архитектура технологии	Поддерживает многопоточность, разделяя логику и UI-действия. Следует принципу разделения кода и дизайна, где логика приложения и его дизайн разделены на различные	Основана на модели приложения с одним потоком выполнения. Логика приложения и его дизайн объединены в одном классе формы. Использует

	классы. Основана на векторной графике и использует язык разметки XAML.	компоненты Windows, такие как кнопки и текстовые поля, для создания интерфейса пользователя.
Производительность	Обе технологии предоставляют разработчикам широкий набор инструментов для создания интерфейса, таких как элементы управления, стили, визуализация данных	
Преимущества в производительности	Позволяет создавать более сложные и красивые пользовательские интерфейсы. Позволяет частичное обновление интерфейса и улучшает производительность во время работы.	Более легковесная архитектуру, поэтому быстрее выполняется при создании простых интерфейсов. Более низкое потребление ресурсов системы.
Возможности	Гибкость и масштабируемость, векторная графика, привязка данных упрощает разработку, расширяемость позволяет создавать пользовательские элементы и использовать их повторно.	Простота использования, работает более эффективно и требует меньше вычислительных ресурсов.
Ограничения	Требуется больше вычислительных ресурсов, имеет более сложный и гибкий подход к созданию интерфейса.	Ограниченный дизайн, не предлагает встроенной поддержки для мультимедийных элементов и композиции.

При выборе между WPF и Windows Forms, важно учитывать требования и цели проекта, а также опыт разработчиков. Обе технологии имеют свои

преимущества и ограничения, и выбор должен быть основан на конкретных потребностях и условиях проекта.

Для хранения данных в базе данных и манипулирования ими необходимо использовать СУБД. SQL Server и Visual FoxPro две базы данных в настоящее время поддерживаются Microsoft. SQL Server был разработан Microsoft как база для распределенных клиентов в клиент-серверной архитектуре. Visual FoxPro была куплена Microsoft и первоначально разрабатывалась для операционной системы DOS, преимущественно для монопольного использования.

В таблице 2 представлена сравнительная характеристика баз данных SQL Server и Visual FoxPro.

Таблица 2 – Сравнительная характеристика баз данных.

Характеристика	SQL Server	Visual FoxPro
Применение	Для любых целей, продолжает расширяться по мере наполнения информацией, без заметного уменьшения быстродействия операций.	Для небольших сетей со средним количеством пользователей около 50. Представляет собой файл-сервер приложений.
Безопасность	Максимальная безопасность. Данные защищены от несанкционированного доступа за счет интеграции сетевой безопасности с сервером безопасности.	Пользователи имеют доступ к данным. Прикладная программа, используя пользователей, может добавить больше защиты, но если пользователь имеет возможность обхода.
Обслуживание	Возможны изменения в структуре данных а так же резервное копирование во время работы сервера, без остановки.	Резервное копирование и изменение структуры можно производить, только когда система (клиенты) базы отключены.

Использование в приложениях	Является приложением базы данных при работе на . Net. Приложение может расширяться и адаптироваться по мере изменения бизнес-климата.	Достаточно много рабочих приложений. Однако эксперты предсказывают, что долгосрочная поддержка Microsoft может быть ограничена.
Минусы	Моноплатформенность	Обрабатываемые данные читаются по сети клиентом, потом обрабатываются им. Узкое место систем, заложен потолок производительности. При сбое хотя бы одного клиента рушатся индексы и заголовки таблиц.

Анализируя сравнительную характеристику двух баз данных, можно сделать вывод, что SQL Server является более надежной и предпочтительной для использования при разработке программного средства.

3.2 Характеристика выбранных программных сред и средств

Для разработки программного средства «Ресторанный бизнес» были выбраны следующие средства разработки: язык программирования C#, среда для разработки Visual Studio, технология WPF и СУБД Microsoft sql server.

C# - это язык программирования от компании Microsoft, разработанный к началу 2000-х годов. Изначально он планировался для создания программ под Windows, а в итоге стал универсальным. Он часто применяется для создания приложений на платформе .NET, включая веб-приложения, службы и настольные программы. C# объединяет простоту языка с высокой производительностью и мощными возможностями.

Достоиства:

- Независимость от аппаратного функционала. Программу не нужно адаптировать под многочисленные платформы и операционные системы. Виртуальная машина .NET Framework сама выполняет эту задачу.
- Управление памятью. Автоматическое очищение памяти для поддержания стабильной работы.
- Безопасность типов. C# – язык со строгой типизацией, что позволяет выявлять ошибки на этапе компиляции.
- Многозадачность. C# поддерживает асинхронное программирование, что особенно полезно для работы с вводом/выводом и сетевыми операциями.

Недостатки:

- Невысокая скорость. При открытии программы на C#код сначала адаптируется под конкретное аппаратное обеспечение, а уже потом исполняется. Таким образом, скорость загрузки становится значительно ниже.
- Ограниченные возможности на низком уровне. В случае необходимости работы с низкоуровневыми задачами, такими как управление памятью, C# может оказаться менее гибким.
- Безопасность. Код, написанный на C#, очень просто декомпилировать.

Visual Studio – это мощная интегрированная среда разработки (IDE), которая используется для создания широкого спектра приложений. Разработанная компанией Microsoft, она предлагает множество удобных и инновационных функций, которые помогают программистам улучшить свою продуктивность и достичь успеха в своей работе.

Преимущества:

- Расширенные функциональные возможности. Включает в себя интегрированные средства отладки, автодополнение, интеграцию с системами управления версиями.

- Широкий выбор поддерживаемых языков программирования. C++, C#, VB.NET и F#. Это позволяет разработчикам выбирать тот язык, с которым они наиболее знакомы и владеют наилучшим образом.
- Инструменты отладки. Мощные инструменты отладки позволяют быстро выявлять и устранять ошибки в коде.
- Официальная поддержка Microsoft. Гарантирует качество и стабильность работы среды разработки. Официальная поддержка Microsoft включает в себя обновления и исправления.

Недостатки:

- Массивность. Понижает производительность компьютера, может привести к сбоям.
- Ограничения по платформам. Некоторые функции и интеграции могут быть ориентированы в основном на экосистему Windows.

Windows Presentation Foundation (WPF) - платформа пользовательского интерфейса, которая не зависит от разрешения и использует векторный механизм визуализации, способный использовать все преимущества современного графического оборудования. Предоставляет комплексный набор функций разработки приложений, которые включают в себя язык XAML, элементы управления, привязку к данным, макет, двумерную и трехмерную графику, анимацию, стили, шаблоны, документы, мультимедиа, текст и типографические функции.

Достоинства:

- Декларативный язык разметки XAML. Использование XAML делает разработку интерфейса более удобной и читаемой.
- Гибкость и мощные стили. WPF предоставляет мощные средства для создания стилей и шаблонов, обеспечивая гибкость при оформлении элементов интерфейса.
- Анимации и эффекты. Возможность создавать анимации и применять различные визуальные эффекты для улучшения пользовательского опыта.

- Данные и привязки. Поддержка привязки данных упрощает взаимодействие с данными и их отображение.

Недостатки:

- Засорение памяти экземплярами ResourceDictionary. Чем больше подключаемые словари, чем больше экземпляров — тем больше уходит времени на инициализацию содержащего их представления и тем больше памяти расходуется впустую.
- Утечка памяти. Даже в среде с автоматической сборкой мусора можно легко получить утечки памяти. Наиболее частая причина утечек — подписка на события без последующего удаления обработчика.
- Ограниченная поддержка платформ. WPF ориентирована в основном на Windows, что может быть недостатком, если ваши приложения должны работать на других операционных системах.

Microsoft sql server - система управления базами данных (СУБД), разработанная компанией Майкрософт. Доступна в нескольких редакциях. Может работать на ПК, ноутбуке, сервере, на виртуальной машине или в облаке. Предназначена для хранения и обработки данных. При взаимодействии с ней пользователи могут отправлять запросы и получать ответы.

Преимущества:

- Масштабирование системы. Взаимодействовать с ней можно как на простых ноутбуках, так и на ПК с мощным процессором, который способен обрабатывать большой объем запросов.
- Размер страниц – до 8 Кб. Данные извлекаются быстро, а сложную информацию удобнее хранить. Система обрабатывает транзакции в интерактивном режиме, есть динамическая блокировка.
- Автоматизация рутинных административных задач. Например, управление блокировками и памятью, редактура размеров файлов.

В программе продуманы настройки, можно создавать профили пользователей.

Минусы:

- Зависимость от ОС. Система работает только с Windows.

4 АЛГОРИТМ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

4.1 Этапы реализации ПС (ПМ)

Создание программной системы включает в себя несколько этапов проектирования, а именно анализ требований, проектирование интерфейса, разработка базы данных, разработка приложения, интеграция всех компонентов и Тестирование и отладка. Рассмотрим каждый этап более подробно:

1. Анализ требований:
    - Изучение функциональных и нефункциональных требований.
    - Разработка архитектуры системы.
    - Проектирование пользовательского интерфейса.
    - Создание диаграмм и спецификаций.
2. Разработка приложения:
    - Написание и отладка серверной части программы.
    - Создание базы данных и ее оптимизация.
3. Разработка интерфейса:
    - Проектирование и реализация пользовательского интерфейса.
    - Внедрение дизайна.
    - Тестирование интерфейса на различных устройствах.
4. Интеграция и тестирование:
    - Совмещение различных компонентов системы.
    - Тестирование целостности системы.
    - Поиск и устранение ошибок.
5. Оптимизация и доработка:
    - Анализ работы системы в эксплуатации.
    - Внесение улучшений и оптимизаций.
    - Доработка функционала в соответствии с обратной связью пользователей.

Таким образом, необходимо пройти все вышеперечисленные этапы, чтобы разработать программное средство, соответствующее поставленным требованиям и удовлетворяющее потребности пользователей.

4.2 Пользовательский интерфейс ПС (ПМ)

4.2.1 Взаимодействие пользователей с ПС (ПМ)

Взаимодействие пользователей с приложением «Ресторанный бизнес» определяются прецедентами:

1. Прецедент «Авторизация в системе»:
    - Начало прецедента: Пользователь запускает приложение.
    - Системный ответ: Отображается окно авторизации.
    - Действие пользователя: Вводит логин и пароль.
    - Системный ответ: Проверка введенный данных. При успешной авторизации – переход на главную страницу, функционал которой определяется должностью, занимаемой пользователем. Если авторизация не выполнилась – выводит сообщение об ошибке.
2. Прецедент «Просмотр профиля пользователя»:
    - Начало прецедента: Пользователь нажимает кнопку «Профиль».
    - Системный ответ: Открывает страницу «Профиль».
    - Действие пользователя: Просматривает свои данные.
3. Прецедент «Выбор продукта со склада»:
    - Начало прецедента: Пользователь нажимает кнопку «Склад продуктов».
    - Системный ответ: Отображается страницу «Склад продуктов».
    - Действие пользователя: Выбирает категорию продуктов.
    - Системный ответ: Сортирует данные и выводит список продуктов.
    - Действие пользователя: Выбирает продукт из списка.
    - Системный ответ: Выделяет выбранный продукт.

- Действие пользователя: Вводит количество продуктов и нажимает кнопку «Взять» или «Удалить».
- Системный ответ: Проверка введенных данных. При вводе корректных данных, если выбрана кнопка «Взять» - информация о пользователе и продукте записывается в журнал, иначе - удаляет продукт из списка. Выводи сообщение об успешной операции.

4. Прецедент «Составление заявки»:
    - Начало прецедента: Пользователь нажимает кнопку «Составить заявку».
    - Системный ответ: Открывает страницу «Составить заявку».
    - Действие пользователя: Выбирает поставщика.
    - Системный ответ: Сортирует и выводит продукты выбранного поставщика. Обновляет список категорий продуктов от выбранного поставщика
    - Действие пользователя: Выбирает категорию продуктов.
    - Системный ответ: Сортирует и выводит продукты из выбранной категории.
    - Действие пользователя: Выбирает продукт из списка.
    - Системный ответ: Выделяет выбранный продукт.
    - Действие пользователя: Вводит количество продуктов и нажимает на кнопку «Добавить в заявку».
    - Системный ответ: Добавляет в список заказанных продуктов
    - Действие пользователя: Выбирает продукт из списка заказанных продуктов и нажимает на кнопку «Удалить из заявки».
    - Системный ответ: Выделяет выбранный продукт и удаляет его из списка заказанных продуктов.
    - Действие пользователя: Нажимает на кнопку «Заказать».
    - Системный ответ: Сохраняет данные о заявке, её составе и пользователе. Выводи сообщение об успешной операции.

5. Прецедент «Прием поставки»:
    - Начало прецедента: Пользователь нажимает кнопку «Поставки».
    - Системный ответ: Открывает страницу «Поставки».
    - Действие пользователя: Просматривает информацию о поставках и выбирает одну из них.
    - Системный ответ: Выделяет выбранную поставку
    - Действие пользователя: Нажимает на кнопку «Принять».
    - Системный ответ: Изменяет состояние поставки как принятая, добавляет продукты из поставки на склад. Выводи сообщение об успешной операции.
6. Прецедент «Просмотр пользователей»:
    - Начало прецедента: Пользователь нажимает кнопку «Сотрудники».
    - Системный ответ: Открывает страницу «Сотрудники».
    - Действие пользователя: Просматривает информацию о сотрудниках.
7. Прецедент «Удаление пользователя»:
    - Начало прецедента: Пользователь находится в окне «Сотрудники». Выбирает сотрудника.
    - Системный ответ: Выделяет выбранного сотрудника.
    - Действие пользователя: Нажимает на кнопку «Удалить».
    - Системный ответ: Выделяет выбранного сотрудника.
    - Действие пользователя: Нажимает на кнопку «Удалить».
    - Системный ответ: Удаляет информацию о пользователе. Ограничивает возможности пользователя взаимодействовать с программным средством. Выводи сообщение об успешной операции.
8. Прецедент «Добавление пользователя»:
    - Начало прецедента: Пользователь нажимает кнопку «Добавить сотрудника».

- Системный ответ: Открывает страницу «Добавить сотрудника».
- Действие пользователя: Вводит данные о новом пользователе и нажимает кнопку «Добавить».
- Системный ответ: Проверяет корректность данных. Если данные введены верно, добавляет пользователя, открывает доступ к приложению по введенному логину и паролю. Выводи сообщение об успешной операции.

9. Прецедент «Выход из приложения»:
    - Начало прецедента: Пользователь нажимает кнопку «Выйти».
    - Системный ответ: Завершение текущей сессии пользователя. Открытие окна авторизации.
    - Действие пользователя: При необходимости может повторно авторизоваться.

Приведенные прецеденты представляют основные варианты взаимодействия пользователя с приложением «Ресторанный бизнес».

4.2.2 Проектирование структуры экранов ПС (ПМ) и схемы навигации

Структура экранов приложения «Ресторанный бизнес»:

1. Авторизация:
    - Функциональность: ввод логина и пароля для авторизации в системе.
    - Навигация: После успешной авторизации – переход на главный экран. Переход на одну из трех страниц, предоставляющих разные возможности (в зависимости от должности).
2. Главный экран:
    - Функциональность: отображение логотипа ресторана, возможность выбора раздела и выхода из аккаунта.
    - Навигация: Можно перейти к разделам профиль, склад продуктов, составить заявку, поставки, сотрудники, добавить сотрудника, а так же вернуться к окну авторизации.

3. Раздел «Профиль»:
    - Функциональность: вывод данный о пользователе, а именно ФИО, номера телефона и должности.
    - Навигация: переход к другим разделам или переход к окну авторизации.
4. Раздел «Склад продуктов»:
    - Функциональность: отображение списка продуктов на складе, возможность сортировки продуктов по категориям, выбора продукта, можно взять его или удалить со склада.
    - Навигация: переход к другим разделам или переход к окну авторизации.
5. Раздел «Составить заявку»:
    - Функциональность: отображение списка продуктов поставщиков, возможность сортировки продуктов по поставщикам и категориям. Можно выбрать продукт и добавить его в состав заявки, так же удалить позицию из заявки и отправить заявку.
    - Навигация: переход к другим разделам или переход к окну авторизации.
6. Раздел «Поставки»:
    - Функциональность: отображение списка поставок, можно выбрать поставку и принять её, автоматически добавляя продукты на склад.
    - Навигация: переход к другим разделам или переход к окну авторизации.
7. Раздел «Сотрудники»:
    - Функциональность: отображение списка сотрудников, возможность сортировки по должностям, можно удалить пользователя.
    - Навигация: переход к другим разделам или переход к окну авторизации.

8. Раздел «Добавить сотрудника»:
    - Функциональность: ввод данных сотрудника и добавление нового пользователя.
    - Навигация: переход к другим разделам или переход к окну авторизации.

Навигационная схема построена с учетом удобства пользователя, обеспечивая легкий доступ к основным функциям приложения. Продемонстрируем навигацию между экранами с помощью схемы функционирования экранов, представленной на рисунке 3.

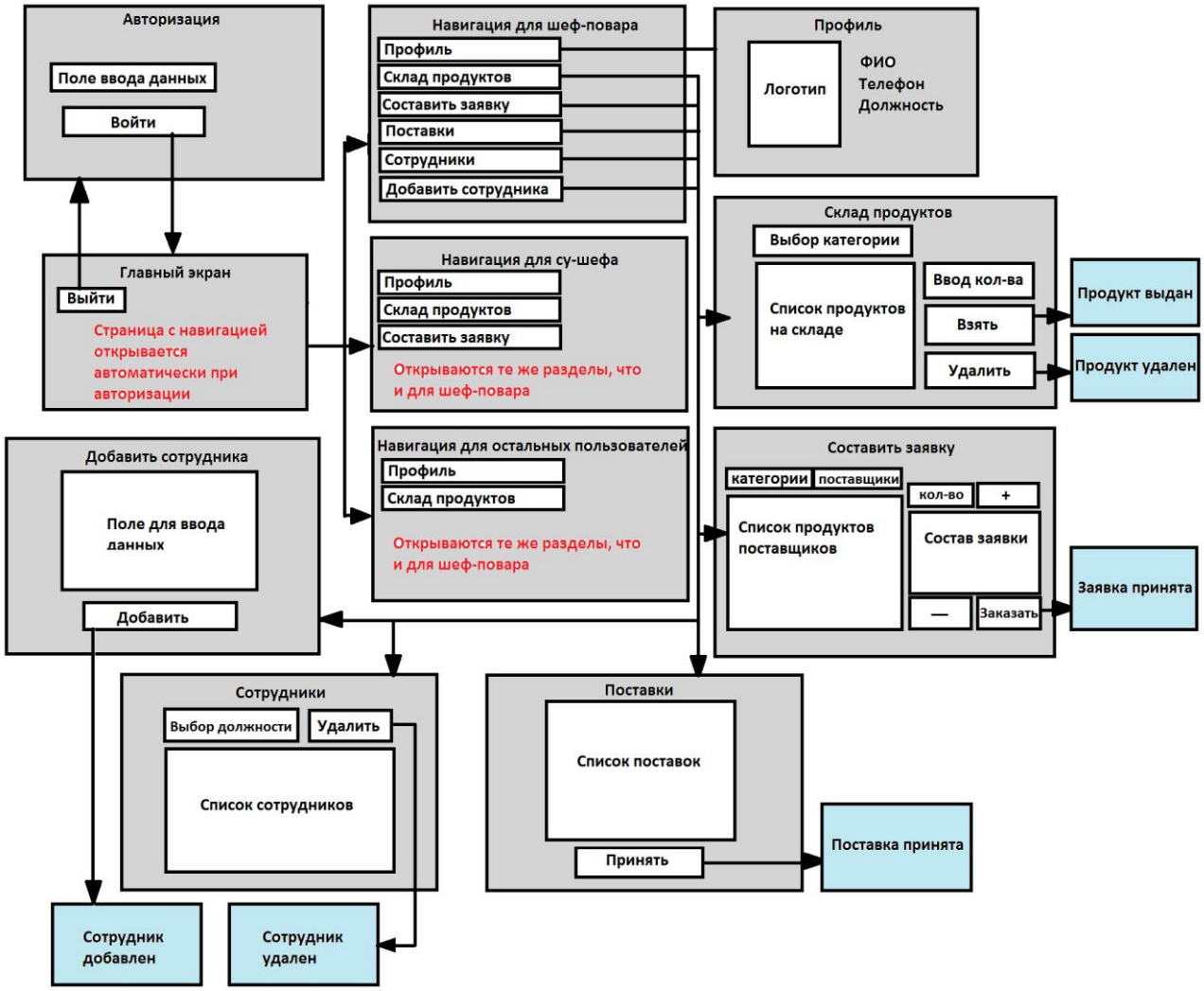


Рисунок 3 – Схема функционирования экранов

Таким образом, навигационная схема разработанного приложения «Салон красоты» предоставляет пользователям удобный и интуитивно понятный

интерфейс для взаимодействия с основными функциональными блоками системы. Архитектурное решение использовать страницы обеспечивает возможность разработки приложения, используя паттерн MVVM.

4.3 Входные, выходные и промежуточные данные

В данном разделе представлена подробная структура входных, выходных и промежуточных данных разрабатываемого приложения «Ресторанный бизнес».

Входные данные включают:

- Пользовательский ввод. Входные данные от пользователя, вводимые через графический интерфейс пользователя (GUI). Включают данные о пользователе при авторизации, продуктах, поставках и т.д.
- Данные из базы данных. Система получает входные данные из базы данных, такие как информация о продуктах, пользователях, поставщиках.

Промежуточные данные:

- Данные обработки. Внутренние данные, используемые системой для обработки информации. Например, списки продуктов, поставок, сотрудников.

Выходные данные:

- Графический интерфейс. Информация, представленная на GUI, включая экраны авторизации, составления заявок, приема поставок и другие сведения, необходимые для взаимодействия с пользователем.
- Данные для базы данных. Изменения и обновления, которые система вносит в базу данных в результате действий пользователя, такие как создание новой заявки или сотрудника.

Структура входных, выходных и промежуточных данных тесно связана с функциональностью программного средства.

4.4 Разработка базы данных, реализуемой в рамках ПС (ПМ)

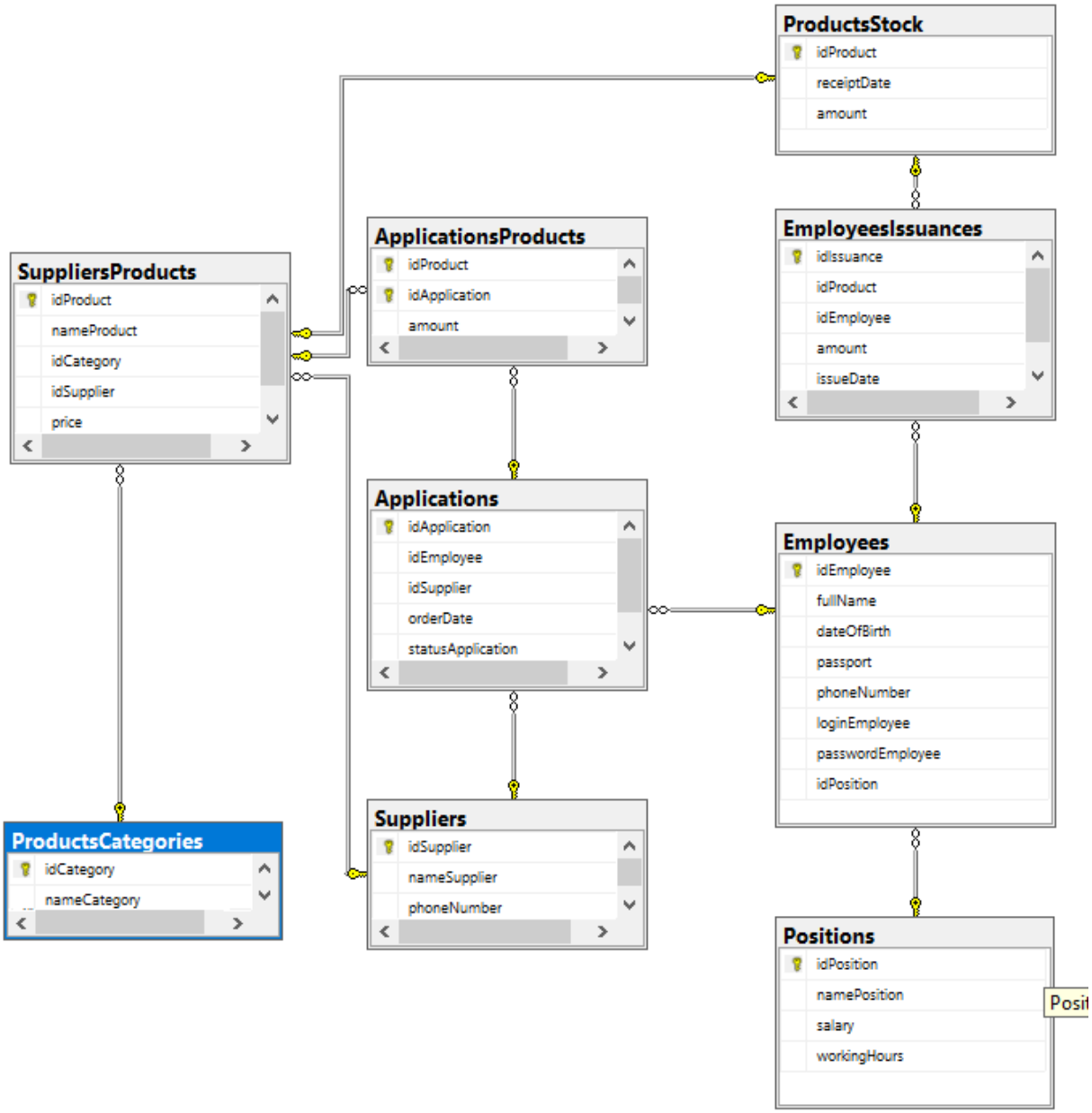
В рамках проекта была разработана база данных (БД), предназначенная для хранения информации о сотрудниках ресторана, поставщиках, продуктах, заявках и журнале выдачи продуктов.

БД состоит из следующих таблиц:

- Positions – должности. В ней хранятся уникальные номера должностей, название, зар.плата и количество рабочих часов в месяце.
- Employees – сотрудники ресторана. Содержит информацию об уникальном идентификаторе, ФИО, дате рождения, паспортных данных, номере телефона, логина, пароля и уникального номера должности.
- Suppliers – поставщики. В ней информация: уникальный идентификатор, название организации и номер телефона.
- ProductsCategories – категории продуктов. Хранит данные о номере категории и её названии.
- SuppliersProducts – продукты поставщиков. Хранится информация об идентификаторе продукта, названии, номере категории продукта, идентификаторе поставщика и цене.
- ProductsStock – склад продуктов. В ней содержится информация о номере продукта, дате приема и количестве на складе.
- EmployeesIssuances – журнал выдачи продуктов со склада. В ней указываются номера колонки в журнале, продукта, сотрудника, взявшего продукт, а так же количество и дата.
- Applications – заявки. Хранит данные о номере заявки, уникальных идентификаторах сотрудника поставщика, о дате заказа и статусе заявки. Если статус равен 0, то поставка ещё не принята, если 1 – то продукты уже добавлены на склад.

- ApplicationsProducts – состав заявки. Содержит информацию о номерах заявки и продукта и о количестве заказанного продукта.

Каждая таблица содержит уникальный идентификатор - первичный ключ, так же в разработанной БД существуют внешний связи между таблицами – внешние ключи. Эти связи обеспечивают целостность данных и позволяют эффективно организовывать информацию. На рисунке 4 представлена диаграмма базы данных.



В данной схеме:

(PK) обозначает первичный ключ,

(FK) обозначает внешний ключ.

Эта схема демонстрирует, как таблицы базы данных для программного средства «Ресторанный бизнес» связаны между собой посредством внешних ключей, обеспечивая целостность данных в базе данных.

4.5 Архитектура ПС (ПМ)

Архитектура приложения играет ключевую роль в обеспечении его гибкости, масштабируемости и удобства использования. Она служит основой для разработки и понимания структуры программы, определяя взаимосвязь между ее компонентами.

Приложение «Ресторанный бизнес» разработано, используя паттерн MVVM. Это значит, что проект состоит из трех компонентов: Model, View и ViewModel. При этом модель и внешний вид приложение не связаны напрямую, а связь между ними осуществляется через ViewModel.

Классы, содержащиеся в Model:

- Employee. В этом классе определяется сущность пользователя. Экземпляр данного класса хранит информацию о пользователе, а именно: уникальный идентификатор, ФИО и остальные данные, хранящиеся в БД в таблице Employees. Аналогично данному классу определены ещё три класса сущностей:
- Product – продукт.
- ProductsApplication – заявка.
- SuppliersProduct – продукт посавщика.

Данные классы имеют только публичные поля и конструктор.

ViewModel подключается к пользовательскому интерфейсу при помощи DataContext. Классы ViewModel:

1. Базовые классы. Паттерн MVVM подразумевает создание классов, наследующих свойства от интерфейса INotifyPropertyChanged, который используется для выявления изменения свойств объектов View, и ICommand, необходимого для привязки команд.

Для удобства разработки были созданы:

- Абстрактный класс ViewModelBase (наследник от INotifyPropertyChanged).
- Класс RelayCommand (наследник ICommand).

Все остальные классы ViewModel наследуются уже от класса ViewModelBase. Так как в приложении «Ресторанный бизнес» используется база данных, для удобства разработки был создан класс для взаимодействия с базой данных:

- DataBase. Методы этого класса позволяют создавать соединение с базой данных, а так же выполнять запросы. Для работы с базой данных Microsoft SQL Server был установлен пакет для управления пазами данных System.Data.SqlClient от Microsoft.

2. Классы ViewModel для взаимодействия с окнами. Проект состоит из двух окон и страниц, которые привязываются к окнам при помощи фреймов. Данные классы наследуются от класса ViewModelBase и отвечают за связь окон приложения и сущностей. Классы:
    - LoginWindowVM – класс для LoginWindow – окна авторизации. В данном классе производится проверка введенных пользователем логина и пароля, а так же создание экземпляра класса Employee – текущего пользователя.
    - ControlWindowVM – отвечает за управление окном ControlWindow. Оно открывается после успешной авторизации пользователя. В ControlWindowVM передается информация о текущем пользователе. При загрузке окна к фрейму привязывается одна из трех страниц (в зависимости от должности сотрудника), отвечающих за функционал приложения.

3. Классы ViewModel для взаимодействия со страницами.

В проекте разработаны три страницы, содержащие кнопки для перехода на другие страницы, содержащие главный функционал приложения. Для этих трех страниц (ControlPageChef, ControlPageSuchef, ControlPageUsers) используется один класс из ViewModel:

- ControlPagesVM. Данный класс определяет текущую страницу, обрабатывая нажатия на соответствующие кнопки, и привязывает эту страницу к фрейму на одной из страниц управления.

Остальные шесть страниц имеют каждый свой класс ViewModel. Создание персонального класса для каждой страницы обусловлено удобством разработки и тестирования программного средства. Для удобства классы повторяют названия страниц, для которых написаны, но в конце добавлено сокращение VM (ViewModel). Классы управления страницами:

- ProfileVM – для соответствующей страницы Profile. Используя класс EmployeeVM (описан ниже), обеспецивает вывод данных о текущем пользователе.
- ProductsStockVM – для страницы ProductsStock. Используется для получения из базы данных информации о продуктах, хранящихся на складе, а так же добавления новых данных в таблицу EmployeesIssuances, когда пользователь берет какой-то продукт со склада.
- CreateApplicationVM. Используется для страницы CreateApplication. Обеспечивает вывод данных о продуктах, предлагаемых поставщиками, а так же создание заявок. В классе создается список выбранных продуктов, которые записываются в базу данных после отправки заявки.
- ProductsSuppliesVM. Класс, отвечающий за прием поставок. На страницу выводится информация из базы данных о ещё не

принятых поставках, а при обработке нажатия на кнопку продукты из заявки добавляются на склад.

- UsersListVM. Данный класс обеспечивает вывод данных пользователей из базы данных, их сортировку, а так же удаление пользователей.
- AddUserVM. Позволяет добавлять в базу данных информацию о новом пользователе, после чего он может пройти авторизацию по сохраненным логину и паролю.

Для выбора параметра для сортировки данных из базы данных используются элементы combobox (выпадающий список). Для обработки события изменения выбранного значения используются триггеры. Все данные из базы данных выводятся в dataGrid, это позволяет выбирать определенную строку и обращаться к её значениям.

4. Классы ViewModel для сущностей.

Для каждого класса из Model был разработан класс, чтобы обращаться к полям экземпляров сущностей. Такой подход заметно упрощает код и уменьшает его количество. Название классов ViewModel для сущностей так же соответствует названиям классов Model:

- EmployeeVM. Класс для создания связи с полями экземпляров класса Employee.
- ProductsApplicationVM
- ProductVM
- SuppliersProductVM

Все свойства данных классов соответствуют полям моделей.

Таким образом, отделение логики программного средства от данных и пользовательского интерфейса позволяет сокращать жесткие зависимости между различными видами кода. Это упрощает изменение отдельных блоков кода без непредвиденного побочного воздействия на другие блоки.

5 ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ

5.1 План тестирования

Тестирование является неотъемлемой частью разработки программных продуктов, особенно в контексте создания современных и универсальных приложений. Важным этапом в этом процессе является тестирование интерфейса, взаимодействия пользователя, базы данных, производительности и безопасности.

План тестирования приложения «Ресторанный бизнес» включает следующее:

1. Тестирование интерфейса:
   - Проверка отображения на различных разрешениях экрана:
   - Проверка корректного отображения элементов интерфейса на различных устройствах и экранах.
2. Взаимодействие пользователя:
   - Тестирование работы интерфейса при вводе данных пользователем.
   - Проверка реакции интерфейса на действия пользователя (клики, ввод текста).
3. Тестирование базы данных:
   - Проверка корректности хранения данных:
   - Проверка, что данные в базе хранятся верно и без искажений.
   - Проверка правильности работы механизмов вставки и обновления данных.
4. Тестирование производительности базы данных:
   - Оценка времени выполнения запросов к базе данных.
   - Проверка эффективности индексов и структуры таблиц.
5. Тестирование безопасности:
   - Проверка уровня доступа. Тестирование наличия несанкционированного доступа к данным.

• Проверка корректности применения механизмов аутентификации и авторизации.

6. Тестирование на уязвимости:

• Поиск и проверка на уязвимости в системе.

Таким образом, тестирование в различных аспектах обеспечивает надежность и функциональность разрабатываемого приложения, поднимая его к высоким стандартам качества и удовлетворяя потребности пользователей.

5.2 Результаты тестирования

Результаты тестирования подтверждают, что разработанное приложение успешно прошло все испытания. Интерфейс обеспечивает корректное отображение на различных разрешениях, а взаимодействие пользователя с ним происходит без ошибок. База данных хранит информацию верно, обеспечивая эффективное выполнение запросов.

В области безопасности приложение демонстрирует надежную работу механизмов аутентификации и авторизации, а выявленные уязвимости были оперативно устранены.

5.3 Оптимизация ПС

На этапе оптимизации приложения «Ресторанный бизнес» были успешно проведены мероприятия по повышению его эффективности и производительности. Был использован паттерн MVVM, который поддерживает многопоточность, а значит, увеличивает скорость загрузки и работы приложения. Так же были применены словари ресурсов, чтобы избежать избыточности кода. Произведена оптимизация хранения данных, что обеспечивает их корректное и эффективное использование. Механизмы вставки и обновления данных были настроены для максимальной производительности, что способствует оперативному доступу к информации. Результаты оптимизации подтверждают готовность приложения к успешному внедрению.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

После запуска приложения пользователь увидит окно, где предлагается пройти авторизацию для входа в аккаунт. Необходимо ввести свои логин и пароль в поля для ввода, представленные на рисунке 5.

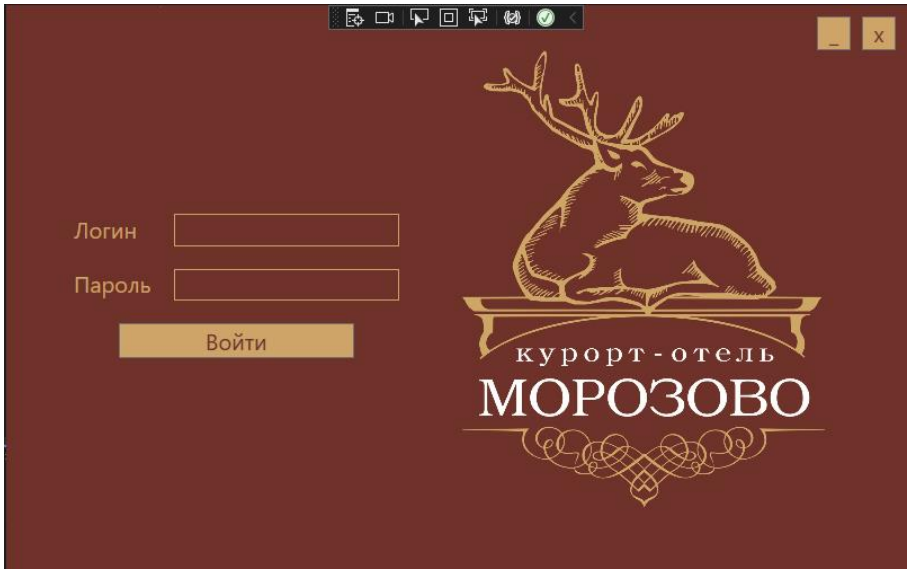


Рисунок 5 – окно авторизации

Если данные были введены неверно, появится сообщение об ошибке как на рисунке 6.

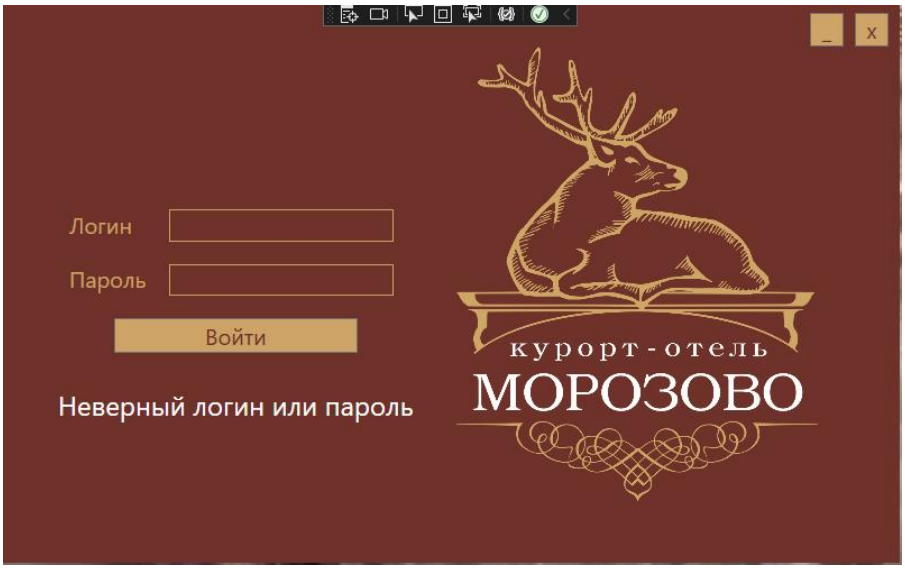


Рисунок 6 – окно авторизации при неверном вводе логина или пароля

После успешной авторизации откроется главное окно приложения. Его внешний вид и количество кнопок зависит от должности сотрудника. На рисунке 7 представлен главный экран для официанта, а так же бармена и поваров.

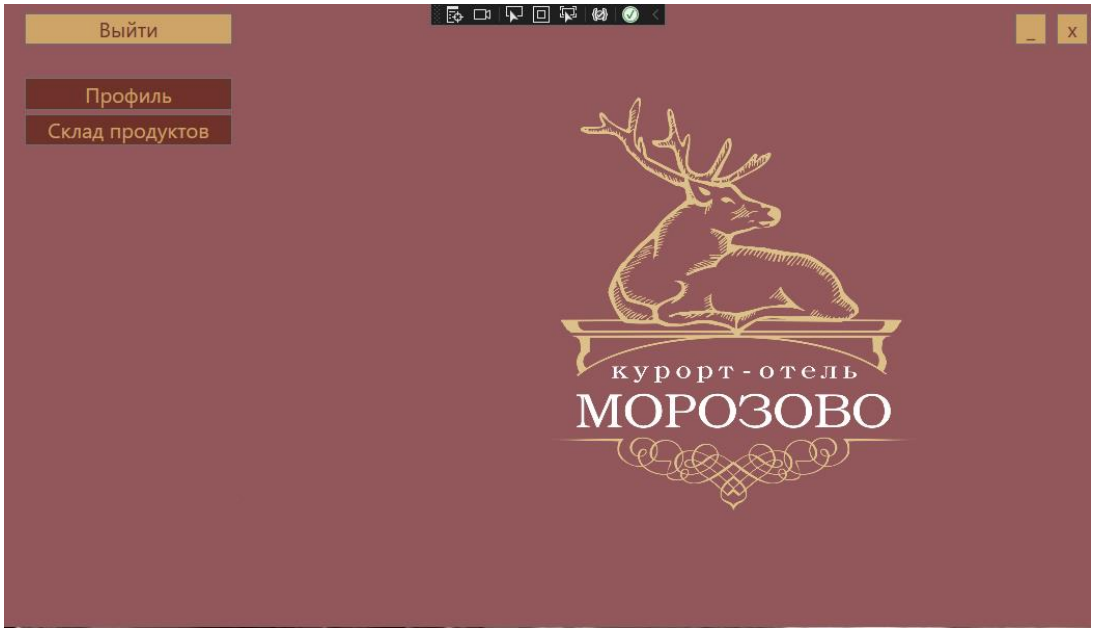


Рисунок 7 – Главный экран для официанта

Су-шеф обладает большим кругом возможностей, главное окно для него представлено на рисунке 8.

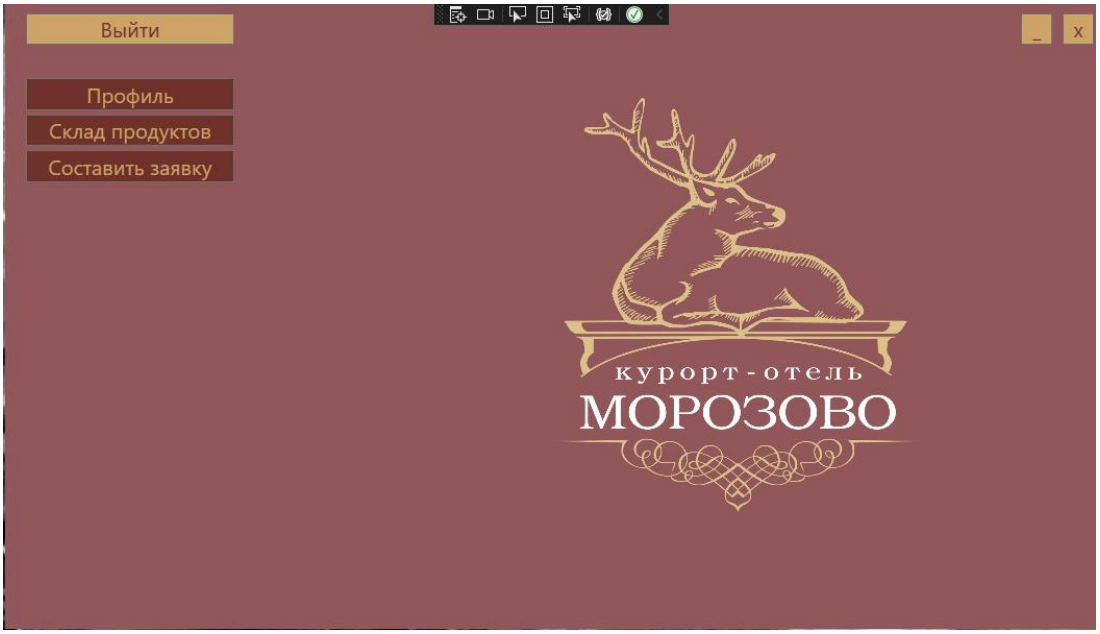


Рисунок 8 - Главный экран для су-шефа

На рисунке 9 изображен главный экран приложения, если войти в него через аккаунт шеф-повара. Он обладает всеми возможностями взаимодействия с приложением.

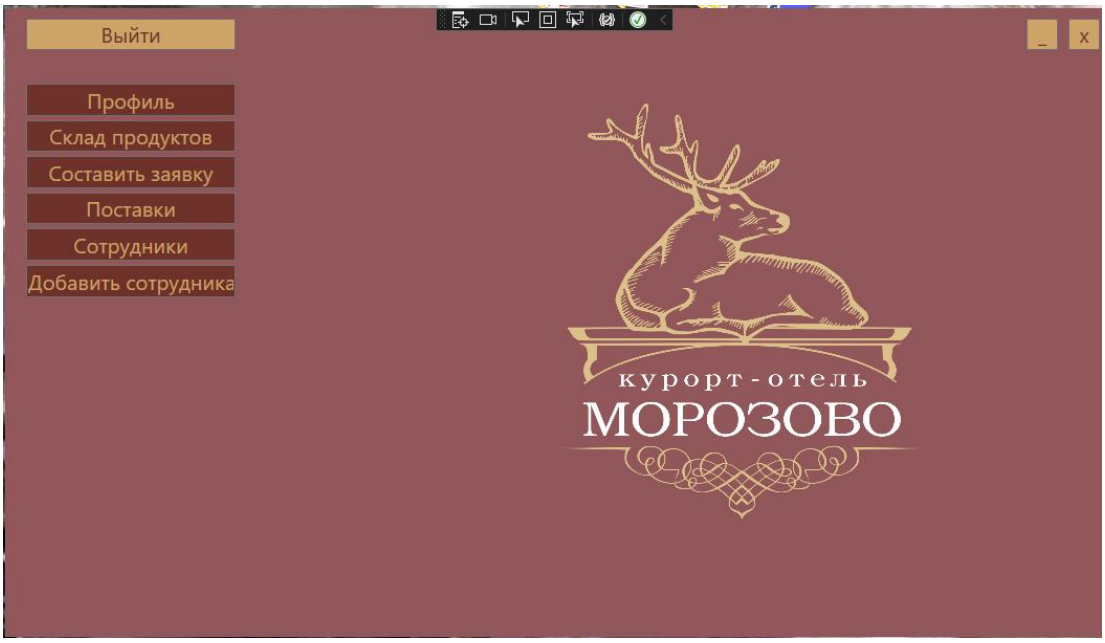


Рисунок 9 - Главный экран для шеф-повара

При нажатии на кнопку «Выйти» в левом верхнем углу пользователь выйдет из своего аккаунта и откроется окно авторизации.

При нажатии на кнопку «Профиль» откроется соответствующая страница (рисунок 10).

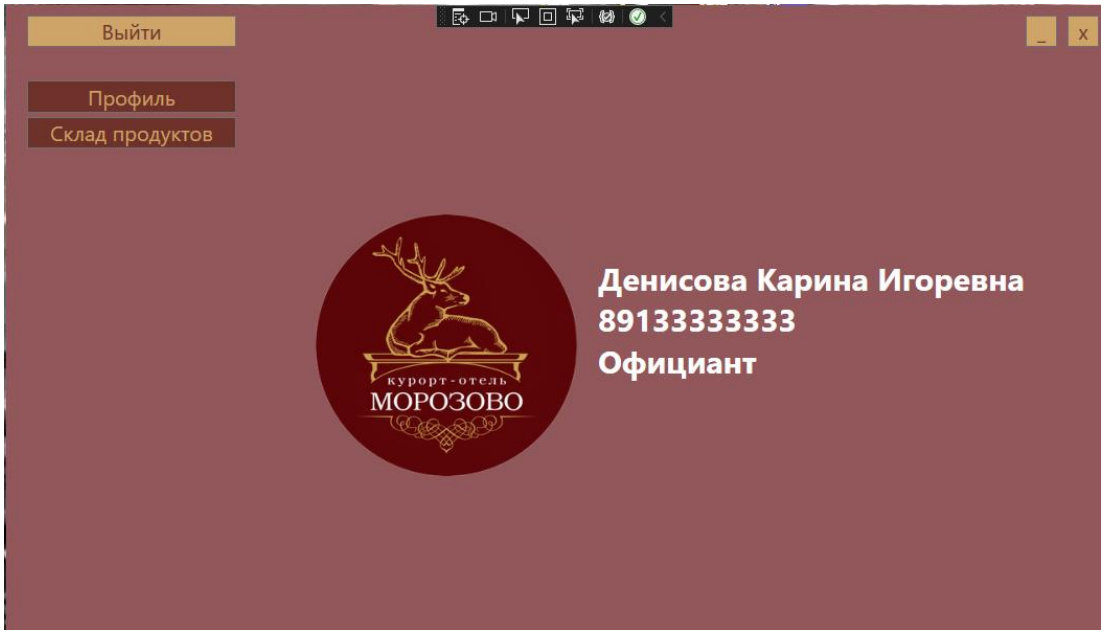


Рисунок 10 – Раздел «Профиль»

В разделе «Склад продуктов», изображенном на рисунке 11, пользователь может просмотреть список продуктов, а так же взять какой-либо продукт или удалить.

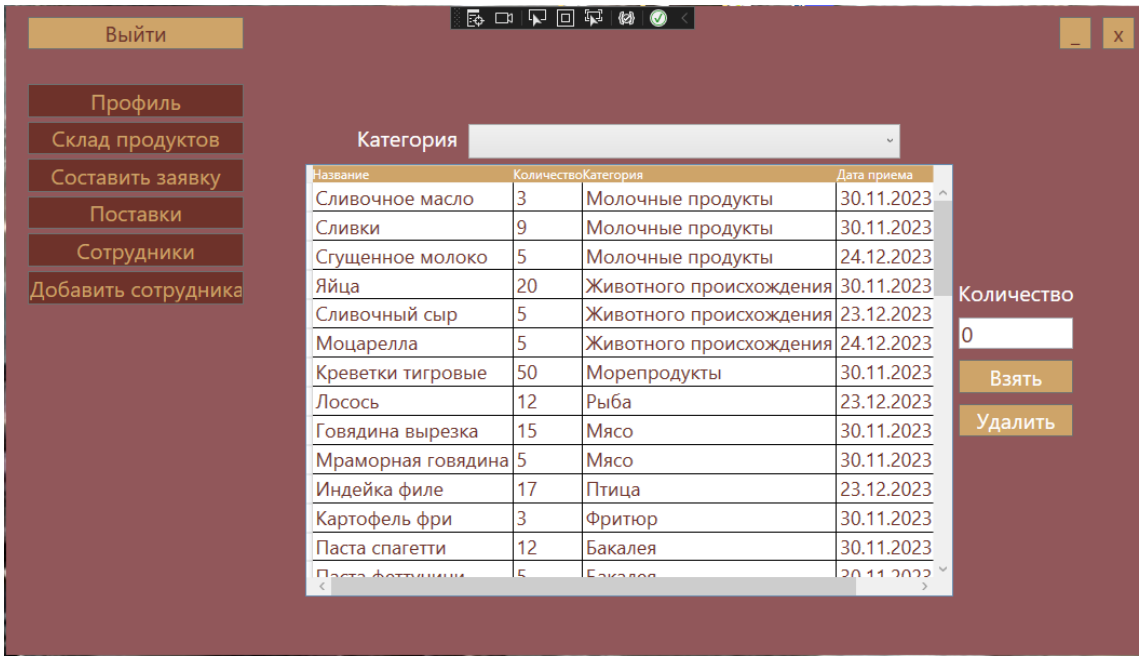


Рисунок 11 – Раздел «Склад продуктов»

При необходимости можно отсортировать продукты по категориям, используя выпадающий список (рисунок 12). Для выбора продукта необходимо навести и кликнуть по нему мышкой.

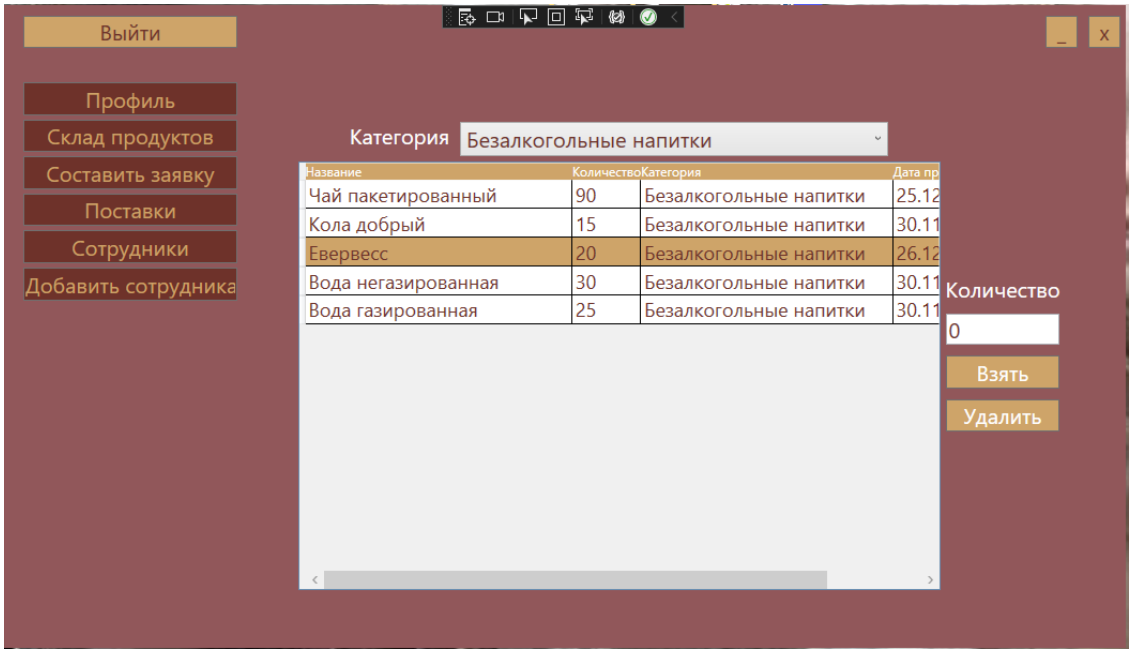


Рисунок 12 – Сортировка по категориям в разделе «Склад продуктов»

После выбора продукта необходимо указать допустимое количество и нажать кнопку «Взять» или «Удалить».

Кликнув по кнопке «Составить заявку», пользователь откроет соответствующую страницу. Аналогично, можно отсортировать продукты по категориям или поставщикам. После указания количества и нажатия на кнопку «+», продукт добавляется в состав заявки. Выбрав продукт из заявки, можно его удалить, кликнув по кнопке «Удалить из заявки». Чтобы отправить заявку необходимо нажать на кнопку «Заказать». На рисунке 13 представлен раздел «Составить заявку».

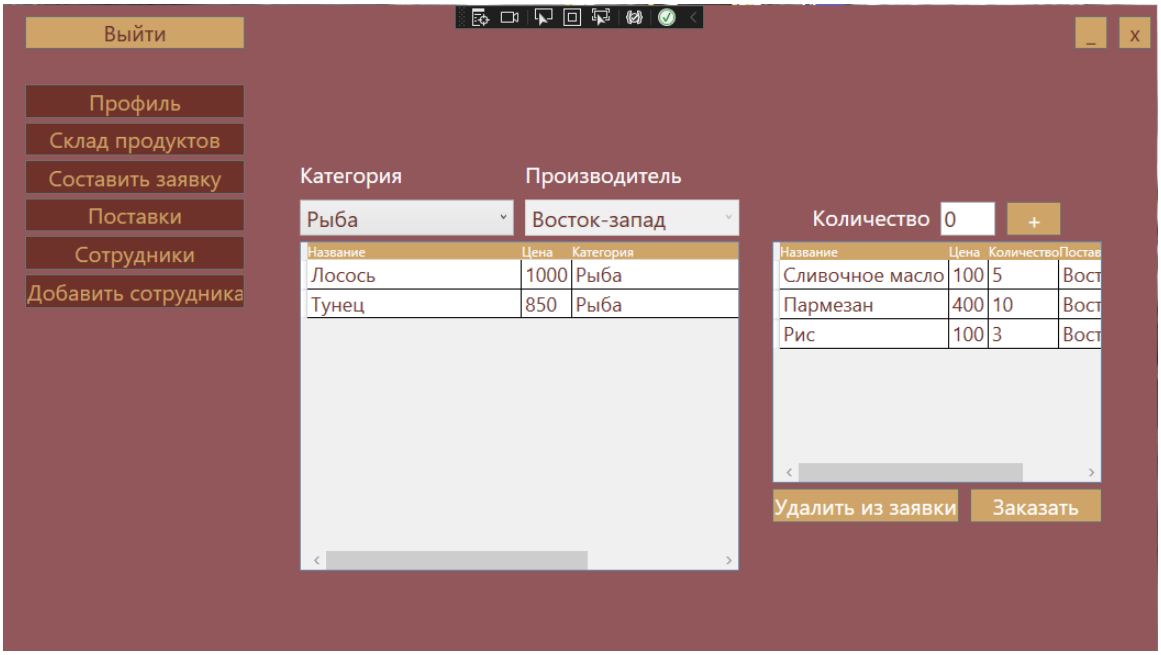


Рисунок 13 - Раздел «Составить заявку»

В разделе «Поставки» (рисунок 14) можно просмотреть список поставок и продуктов, входящих в них, и выбрать необходимую поставку. После нажатия на кнопку «Принять» продукты автоматически попадут на склад. А поставка перейдет в раздел принятые.

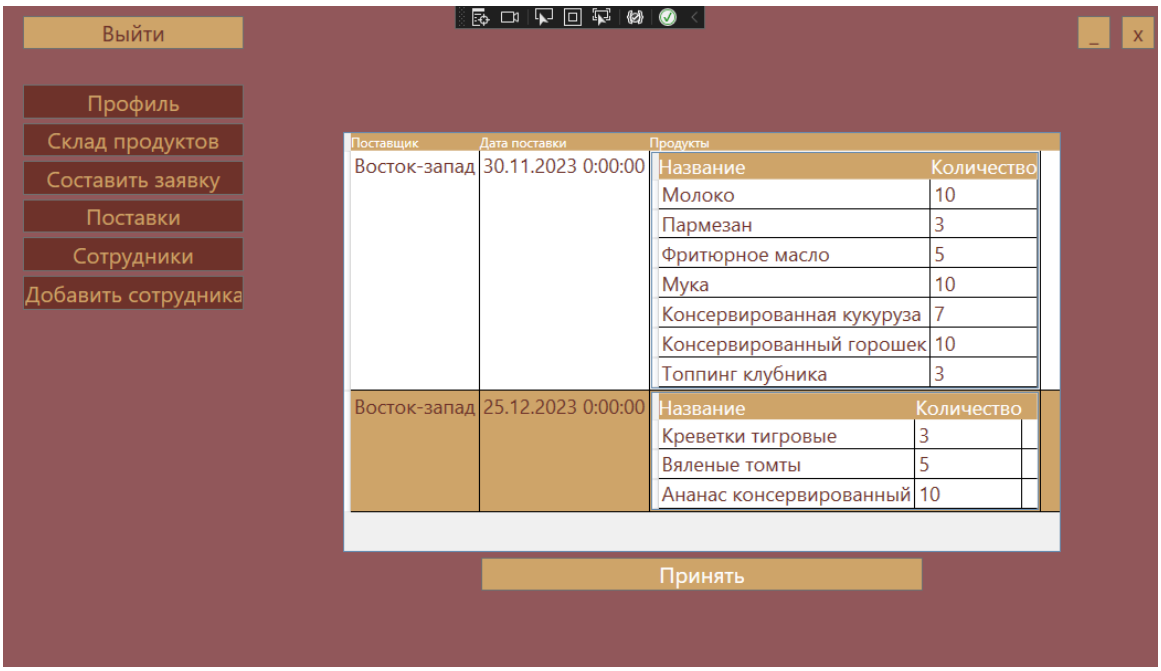


Рисунок 14 - Раздел «Поставки»

На рисунке 15 представлен раздел «Сотрудники», в котором можно просмотреть список сотрудников, отсортировать его по должности и при нажатии на кнопку «Удалить» - удалить пользователя.

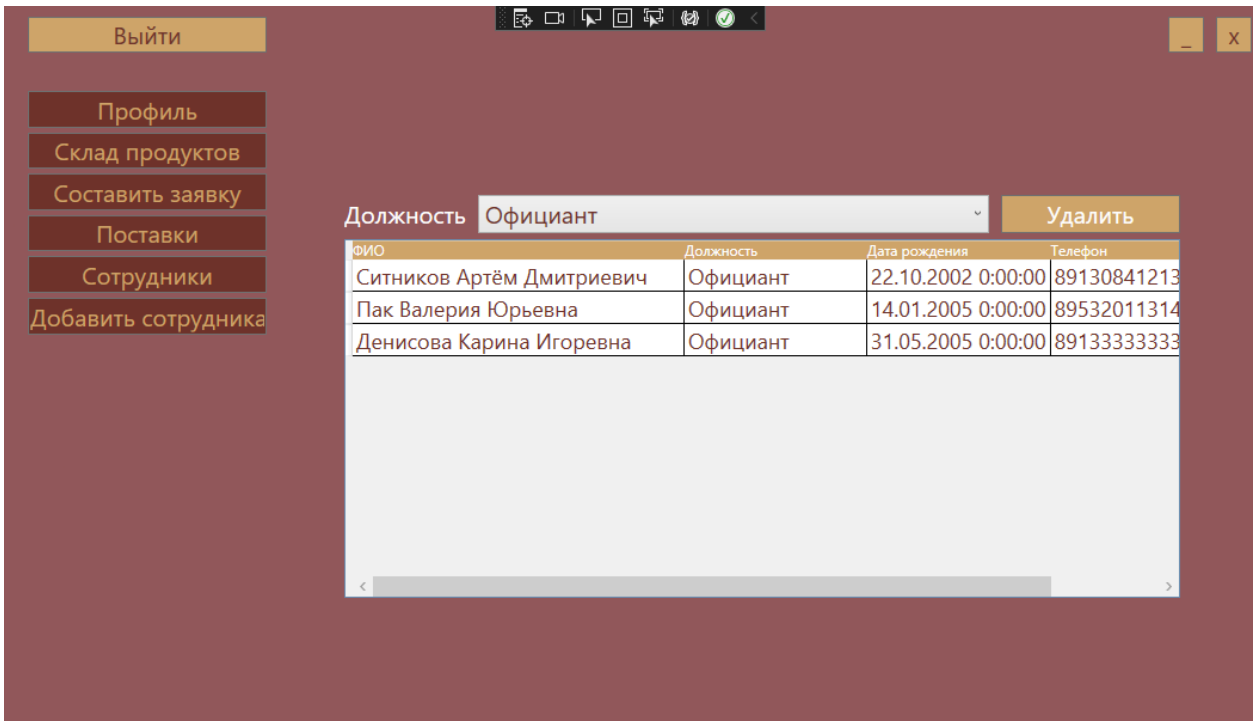


Рисунок 15 – Раздел «Сотрудники»

Заключительная страница – раздел «Добавить сотрудника». Здесь можно добавить нового пользователя, заполнив все поля и нажав на кнопку «Добавить». Данный раздел представлен на рисунке 16.

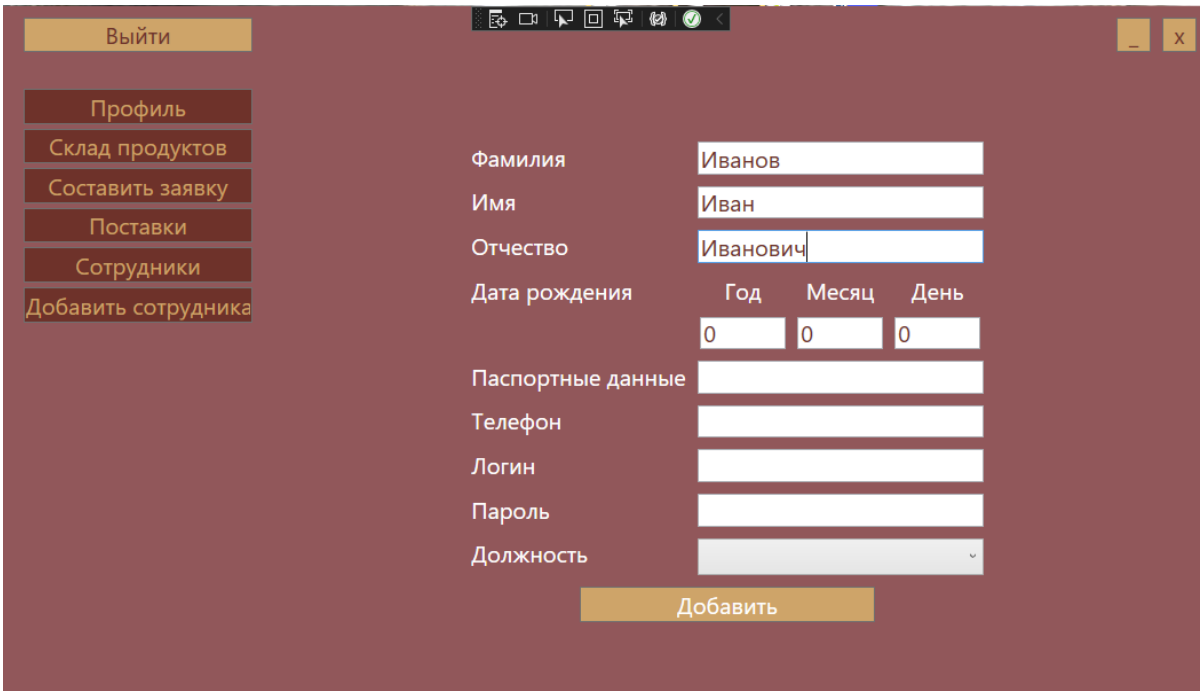


Рисунок 16 – Раздел «Добавить сотрудника»

Для закрытия приложения пользователю необходимо нажать на кнопку выхода («X»), чтобы свернуть приложение – на кнопку «_».

В случае возникновения сбоев или ошибок, перезапустите приложение.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была поставлена задача разработки приложения для ресторана. Основной целью было создание функционального и удобного в использовании программного средства, способного автоматизировать ключевые процессы заведения.

В процессе выполнения курсовой работы успешно решены поставленные задачи по разработке программного средства для ресторана. Основной акцент был сделан на создании интуитивно понятного, функционального и эффективного приложения, способного оптимизировать операционные процессы в сфере управления рестораном.

Разработанное программное средство представляет собой комплексное приложение, охватывающее основные аспекты управления рестораном и организации работы кухни. Это включает в себя систему заказа и приема продуктов, отслеживание расходов продуктов, а также поддержку базы данных сотрудников. Процесс тестирования приложения подтвердил его успешное функционирование.

Полученное программное средство является расширяемым, и его функционал может быть легко дополнен новыми модулями, такими как бронирование столов гостями, прием заказов, передача заказов на бар и кухню и др. Это создает перспективы для дальнейшего усовершенствования и адаптации приложения под разнообразные потребности пользователей.

Таким образом, цель по разработке приложения для ресторана успешно достигнута, и созданное программное средство является практически значимым решением для оптимизации деятельности ресторанного бизнеса.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Что такое база данных. URL: https://www.oracle.com/cis/database/what-is-database/
2. Десктоп приложения. URL: https://freematiq.com/uslugi/desktop-prilozheniya/
3. Что такое C#: плюсы и минусы языка. URL: https://gb.ru/blog/chto-takoe-c/
4. Почему стоит выбрать Visual Studio: преимущества и возможности. URL: https://uchet-jkh.ru/i/pocemu-stoit-vybrat-visual-studio-preimushhestva-i-vozmoznosti
5. Руководство по классическим приложениям (WPF .NET) URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/overview/?view=netdesktop-8.0
6. Отличия WPF от Windows Forms. URL: https://uchet-jkh.ru/i/otliciya-wpf-ot-windows-forms
7. Подводные камни WPF. URL: https://habr.com/ru/articles/262299/
8. Microsoft sql server: преимущества и недостатки. URL: https://astv.ru/news/materials/microsoft-sql-server-preimushestva-i-nedostatki
9. Паттерн MVVM. URL: https://metanit.com/sharp/wpf/22.1.php
10. Привязка данных и MVVM. URL: https://learn.microsoft.com/ru-ru/windows/uwp/data-binding/data-binding-and-mvvm

ПРИЛОЖЕНИЯ

Приложение А: DataBase.cs

```
using System;
using System.Data;
using Microsoft.Data.SqlClient;

namespace MyRestaurant.ViewModel.BaseVM
{
    public class DataBase
    {
        SqlConnection sqlConnection = new
SqlConnection(@"Server=COMPUTER\SQLEXPRESS;Database=Restaurant;Trusted_Connection=true;MultipleActiv
eResultSets=true;TrustServerCertificate=true;encrypt=false");
        public void OpenConnection()
        {
            if (sqlConnection.State == System.Data.ConnectionState.Closed)
            {
                sqlConnection.Open();
            }
        }
        public void CloseConnection()
        {
            if (sqlConnection.State == System.Data.ConnectionState.Open)
            {
                sqlConnection.Close();
            }
        }
        public SqlConnection GetConnection()
        {
            return sqlConnection;
        }
        public DataTable SqlSelect(String s)
        {
            OpenConnection();
            SqlCommand sqlCommand = new SqlCommand(s);
            sqlCommand.Connection = GetConnection();
            SqlDataAdapter adapter = new SqlDataAdapter(sqlCommand);
            DataTable dt = new DataTable();
            adapter.Fill(dt);
            CloseConnection();
            return dt;
        }
        public void SqlQuery(String s)
        {
            OpenConnection();
            SqlCommand sqlCommand = new SqlCommand(s);
            sqlCommand.Connection = GetConnection();
            sqlCommand.ExecuteNonQuery();
            CloseConnection();
        }
    }
}
```

Приложение Б: LoginWindowVM.cs

```
using MyRestaurant.View.Windows;
using MyRestaurant.Model;
using MyRestaurant.ViewModel.BaseVM;
using System;
using System.Data;
using System.Windows;
using System.Windows.Input;

namespace MyRestaurant.ViewModel.WindowsVM
{
   public class LoginWindowVM : ViewModelBase
   {
      public DataBase dataBase;
      public DataTable dataTable;
      public Employee CurrentEmployee;

      private string currentLogin;
      public string CurrentLogin
      {
         get { return currentLogin; }
         set { currentLogin = value; OnPropertyChanged("CurrentLogin"); }
      }

      private string currentPassword;
      public string CurrentPassword
      {
         get { return currentPassword; }
         set { currentPassword = value; OnPropertyChanged("CurrentPassword"); }
      }

      public string messageText;
      public string MessageText
      {
         get { return messageText; }
         set { messageText = value; OnPropertyChanged("MessageText"); }
      }

      /// <summary>
      /// Конструктор
      /// </summary>
      public LoginWindowVM()
      {
         dataBase = new DataBase();
      }

      /// <summary>
      /// Команды
      /// </summary>
      public ICommand OpenControlWindow
      {
         get
         {
            return new RelayCommand(obj =>
            {
               if (CheckEmployee())
               {
                  CreateEmployee();
                  CleanWindow();
                  MessageText = "";
                  OpenControlWindowMethod();
```

```
                }
                else
                {
                    CleanWindow();
                    MessageText = "Неверный логин или пароль";
                }
            }
            );
        }
    }
    public ICommand CloseWindow
    {
        get
        {
            return new RelayCommand(obj =>
            {
                CloseWindowMethod();
            }
            );
        }
    }
    public ICommand ResizeWindow
    {
        get
        {
            return new RelayCommand(obj =>
            {
                ResizeWindowMethod();
            }
            );
        }
    }

    /// <summary>
    /// Функции
    /// </summary>
    private void OpenControlWindowMethod()
    {
        ControlWindow controlWindow = new ControlWindow();
        controlWindow.Owner = Application.Current.MainWindow;
        controlWindow.WindowStartupLocation = WindowStartupLocation.CenterOwner;
        controlWindow.Show();
        controlWindow.Owner.Hide();
        controlWindow.DataContext = new ControlWindowVM(CurrentEmployee, controlWindow.Owner);
    }
    private void CleanWindow()
    {
        CurrentLogin = "";
        CurrentPassword = "";
    }
    private bool CheckEmployee()
    {
        dataTable = dataBase.SqlSelect("select Employees.*, Positions.namePosition " +
                    "from Employees, Positions " +
                    "where Employees.idPosition = Positions.idPosition and loginEmployee = '" + CurrentLogin
+
                    "' and passwordEmployee = '" + CurrentPassword + "'");
        return dataTable.Rows.Count == 0 ? false : true;
    }
    private void CreateEmployee()
    {
        CurrentEmployee = new Employee(Convert.ToInt32(dataTable.Rows[0][0]),
dataTable.Rows[0][1].ToString(), dataTable.Rows[0][2].ToString(),
```

```
                dataTable.Rows[0][3].ToString(),                dataTable.Rows[0][4].ToString(),
dataTable.Rows[0][5].ToString(), dataTable.Rows[0][6].ToString(),
                Convert.ToInt32(dataTable.Rows[0][7]), dataTable.Rows[0][8].ToString());
        }
        private void CloseWindowMethod()
        {
            Window window = Application.Current.MainWindow;
            window.Close();
        }
        private void ResizeWindowMethod()
        {
            Window window = Application.Current.MainWindow;
            window.WindowState = WindowState.Minimized;
        }
    }
}
```

Приложение B: ControlWindowVM.cs

```
using MyRestaurant.View.Pages;
using MyRestaurant.Model;
using MyRestaurant.ViewModel.BaseVM;
using MyRestaurant.ViewModel.PagesVM;
using System.Data;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;

namespace MyRestaurant.ViewModel.WindowsVM
{
    public class ControlWindowVM : ViewModelBase
    {
        public DataBase dataBase;
        public DataTable dataTable;
        public Window loginWindow;
        private Page ControlPageUsers;
        private Page ControlPageSuchef;
        private Page ControlPageChef;

        private Page currentPage;
        public Page CurrentPage
        {
            get { return currentPage; }
            set { currentPage = value; OnPropertyChanged("CurrentPage"); }
        }

        private Employee currentEmployee;
        public Employee CurrentEmployee
        {
            get { return currentEmployee; }
            set
            {
                currentEmployee = value;
                OnPropertyChanged("CurrentEmployee");
            }
        }
        public ControlWindowVM(Employee employee, Window window)
        {
            dataBase = new DataBase();
            ControlPageUsers = new ControlPageUsers();
```

```
    ControlPageSuchef = new ControlPageSuchef();
    ControlPageChef = new ControlPageChef();
    CurrentEmployee = employee;
    loginWindow = window;
    ChoosePage();
}

private void ChoosePage()
{
    switch (CurrentEmployee.IdPosition)
    {
        case 1:
            {//Шеф-повар
                CurrentPage = ControlPageChef;
            }
            break;
        case 2:
            {//Су-шеф
                CurrentPage = ControlPageSuchef;
            }
            break;
        default:
            {//Остальные должности
                CurrentPage = ControlPageUsers;
            }
            break;
    }
    CurrentPage.DataContext = new ControlPagesVM(CurrentEmployee, loginWindow);
}
private void CloseWindowMethod()
{
    loginWindow.Close();
}
private void ResizeWindowMethod()
{
    Window window = loginWindow.OwnedWindows[0];
    window.WindowState = WindowState.Minimized;
}
private void BackWindowMethod()
{
    loginWindow.Show();
    Window window = loginWindow.OwnedWindows[0];
    window.Close();
}

public ICommand CloseWindow
{
    get
    {
        return new RelayCommand(obj =>
        {
            CloseWindowMethod();
        }
        );
    }
}
public ICommand ResizeWindow
{
    get
    {
        return new RelayCommand(obj =>
        {
            ResizeWindowMethod();
```

```
            }
            );
        }
    }
    public ICommand BackBtn_Click
    {
        get
        {
            return new RelayCommand(obj =>
            {
                BackWindowMethod();
            }
            );
        }
    }
  }
}
```

Приложение Г: ControlPagesVM.cs

```
using MyRestaurant.Model;
using MyRestaurant.View.Pages;
using MyRestaurant.ViewModel.BaseVM;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;

namespace MyRestaurant.ViewModel.PagesVM
{
  public class ControlPagesVM : ViewModelBase
  {
    private Page Profile;
    private Page ProductsStock;
    private Page CreateApplication;
    private Page ProductsSupplies;
    private Page UsersList;
    private Page AddUser;
    public Window loginWindow;

    private Page currentPage;
    public Page CurrentPage
    {
      get { return currentPage; }
      set { currentPage = value; OnPropertyChanged("CurrentPage"); }
    }

    private Employee currentEmployee;
    public Employee CurrentEmployee
    {
      get { return currentEmployee; }
      set
      {
        currentEmployee = value;
        OnPropertyChanged("CurrentEmployee");
      }
    }

    public ControlPagesVM(Employee employee, Window window)
    {
      CurrentEmployee = employee;
```

```
    Profile = new Profile();
    ProductsStock = new ProductsStock();
    CreateApplication = new CreateApplication();
    ProductsSupplies = new ProductsSupplies();
    UsersList = new UsersList();
    AddUser = new AddUser();
    loginWindow = window;
}

public ICommand ProfileBtn_Click
{
    get
    {
        return new RelayCommand(obj =>
        {
            CurrentPage = Profile;
            CurrentPage.DataContext = new ProfileVM(CurrentEmployee);
        }
        );
    }
}
public ICommand ProductsStockBtn_Click
{
    get
    {
        return new RelayCommand(obj =>
        {
            CurrentPage = ProductsStock;
            CurrentPage.DataContext = new ProductsStockVM(CurrentEmployee);
        }
        );
    }
}
public ICommand CreateApplicationBtn_Click
{
    get
    {
        return new RelayCommand(obj =>
        {
            CurrentPage = CreateApplication;
            CurrentPage.DataContext = new CreateApplicationVM(CurrentEmployee);
        }
        );
    }
}
public ICommand ProductsSuppliesBtn_Click
{
    get
    {
        return new RelayCommand(obj =>
        {
            CurrentPage = ProductsSupplies;
            CurrentPage.DataContext = new ProductsSuppliesVM(CurrentEmployee);
        }
        );
    }
}
public ICommand UsersListBtn_Click
{
    get
    {
        return new RelayCommand(obj =>
```

```
            {
                CurrentPage = UsersList;
                CurrentPage.DataContext = new UsersListVM(CurrentEmployee);
            }
            );
        }
    }
    public ICommand AddUserBtn_Click
    {
        get
        {
            return new RelayCommand(obj =>
            {
                CurrentPage = AddUser;
                CurrentPage.DataContext = new AddUserVM();
            }
            );
        }
    }
  }
}
```

Приложение Д: ProductsStockVM.cs

```
using MyRestaurant.Model;
using MyRestaurant.ViewModel.BaseVM;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Data;
using System.Windows;
using System.Windows.Input;

namespace MyRestaurant.ViewModel.PagesVM
{
  public class ProductsStockVM : ViewModelBase
  {
    private Employee currentEmployee;
    public Employee CurrentEmployee
    {
      get { return currentEmployee; }
      set
      {
        currentEmployee = value;
        OnPropertyChanged("CurrentEmployee");
      }
    }
    private List<string> categories;
    public List<string> Categories
    {
      get { return categories; }
      set
      {
        categories = value;
        OnPropertyChanged("Categories");
      }
    }
    private string selectedCategory;
    public string SelectedCategory
    {
```

```
    get { return selectedCategory; }
    set
    {
        selectedCategory = value;
        OnPropertyChanged("SelectedCategory");
    }
}

public ObservableCollection<Product> Products { get; set; }
private Product selectedProduct;
public Product SelectedProduct
{
    get { return selectedProduct; }
    set
    {
        selectedProduct = value;
        OnPropertyChanged("SelectedProduct");
    }
}

private int productsCount;
public int ProductsCount
{
    get { return productsCount; }
    set
    {
        productsCount = value;
        OnPropertyChanged("ProductsCount");
    }
}

DataBase dataBase;
DataTable dataTable;

/// <summary>
/// Конструктор
/// </summary>
public ProductsStockVM(Employee employee)
{
    CurrentEmployee = employee;
    dataBase = new DataBase();
    Products = new ObservableCollection<Product>();
    FillCombobox();
    RefreshDataGrid();
}

/// <summary>
/// Функции
/// </summary>
private void FillCombobox()
{
    dataTable = dataBase.SqlSelect("select distinct ProductsCategories.nameCategory " +
                        "from SuppliersProducts, ProductsStock, ProductsCategories " +
                        "where SuppliersProducts.idCategory = ProductsCategories.idCategory " +
                        "and ProductsStock.idProduct = SuppliersProducts.idProduct");
    Categories = new List<string>();
    for (int i = 0; i < dataTable.Rows.Count; i++)
    {
        Categories.Add(dataTable.Rows[i][0].ToString());
    }
}
private void RefillDataGrid()
{
```

```
            dataTable = dataBase.SqlSelect("select ProductsStock.idProduct, SuppliersProducts.nameProduct, " +
                        "ProductsStock.receiptDate, ProductsStock.amount, SuppliersProducts.idCategory, " +
                        "ProductsCategories.nameCategory " +
                        "from ProductsCategories, ProductsStock, SuppliersProducts " +
                        "where ProductsCategories.idCategory = SuppliersProducts.idCategory " +
                        "and SuppliersProducts.idProduct = ProductsStock.idProduct and " +
                        "ProductsCategories.nameCategory = '" + SelectedCategory + "'");
            Products.Clear();
            Product product;
            for (int i = 0; i < dataTable.Rows.Count; i++)
            {
                product = new Product(Convert.ToInt32(dataTable.Rows[i][0]), dataTable.Rows[i][1].ToString(),
dataTable.Rows[i][2].ToString(),
                    Convert.ToInt32(dataTable.Rows[i][3]), Convert.ToInt32(dataTable.Rows[i][4]),
dataTable.Rows[i][5].ToString());
                Products.Add(product);
            }
        }
        private void RefreshDataGrid()
        {
            dataTable = dataBase.SqlSelect("select ProductsStock.idProduct, SuppliersProducts.nameProduct, " +
                        "ProductsStock.receiptDate, ProductsStock.amount, SuppliersProducts.idCategory, " +
                        "ProductsCategories.nameCategory " +
                        "from ProductsCategories, ProductsStock, SuppliersProducts " +
                        "where ProductsCategories.idCategory = SuppliersProducts.idCategory " +
                        "and SuppliersProducts.idProduct = ProductsStock.idProduct");
            Products.Clear();
            Product product;
            for (int i = 0; i < dataTable.Rows.Count; i++)
            {
                product = new Product(Convert.ToInt32(dataTable.Rows[i][0]), dataTable.Rows[i][1].ToString(),
dataTable.Rows[i][2].ToString(),
                    Convert.ToInt32(dataTable.Rows[i][3]), Convert.ToInt32(dataTable.Rows[i][4]),
dataTable.Rows[i][5].ToString());
                Products.Add(product);
            }
        }
        private bool CheckProduct()
        {
            return (SelectedProduct == null || ProductsCount == 0 || ProductsCount > SelectedProduct.Amount) ?
false : true;
        }

        private void TakeProductMethod()
        {
            string date = (DateTime.Now).ToString();
            dataBase.SqlQuery("insert into EmployeesIssuances values (" + SelectedProduct.IdProduct + ", " +
                CurrentEmployee.IdEmployee + ", " + ProductsCount + ", '" + date + "')");
            ProductMethod("Продукт выдан");
        }
        private void DeleteProductMethod()
        {
            ProductMethod("Продукт удален");
        }
        private void ProductMethod(string message)
        {
            SelectedProduct.Amount -= ProductsCount;
            ProductsCount = 0;
            if (SelectedProduct.Amount == 0)
            {
                dataBase.SqlQuery("delete from ProductsStock where idProduct = " + selectedProduct.IdProduct);
            }
            else
```

```
                {
                    dataBase.SqlQuery("update ProductsStock set amount = " + selectedProduct.Amount + " where
idProduct = " +
                        selectedProduct.IdProduct);
                }
                SelectedCategory = null;
                SelectedProduct = null;
                RefreshDataGrid();
                MessageBox.Show(message);
            }

            /// <summary>
            /// Команды
            /// </summary>
            public ICommand CategoryChanged
            {
                get
                {
                    return new RelayCommand(obj =>
                    {
                        RefillDataGrid();
                    }
                    );
                }
            }
            public ICommand TakeProduct
            {
                get
                {
                    return new RelayCommand(obj =>
                    {
                        if (CheckProduct())
                        {
                            TakeProductMethod();
                        }
                        else
                        {
                            MessageBox.Show("Выберете продукт и укажите допустимое количество");
                        }
                    }
                    );
                }
            }
            public ICommand DeleteProduct
            {
                get
                {
                    return new RelayCommand(obj =>
                    {
                        if (CheckProduct())
                        {
                            DeleteProductMethod();
                        }
                        else
                        {
                            MessageBox.Show("Выберете продукт и укажите допустимое количество");
                        }
                    }
                    );
                }
            }
        }
    }
```

Приложение Е: Employee.cs

```
namespace MyRestaurant.Model
{
   public class Employee
   {
      public int IdEmployee { get; set; }
      public string FullName { get; set; }
      public string DateOfBirth { get; set; }
      public string Passport { get; set; }
      public string PhoneNumber { get; set; }
      public string LoginEmployee { get; set; }
      public string PasswordEmployee { get; set; }
      public int IdPosition { get; set; }
      public string NamePosition { get; set; }
      public Employee(int idEmployee, string fullName, string dateOfBirth, string passport,
         string phoneNumber, string loginEmployee, string passwordEmployee, int idPosition, string
namePosition)
      {
         IdEmployee = idEmployee;
         FullName = fullName;
         DateOfBirth = dateOfBirth;
         Passport = passport;
         PhoneNumber = phoneNumber;
         LoginEmployee = loginEmployee;
         PasswordEmployee = passwordEmployee;
         IdPosition = idPosition;
         NamePosition = namePosition;
      }
   }
}
```

Приложение Ж: EmployeeVM.cs

```
using MyRestaurant.Model;
using MyRestaurant.ViewModel.BaseVM;

namespace MyRestaurant.ViewModel.ModelsVM
{
   public class EmployeeVM : ViewModelBase
   {
      private Employee employee;
      public EmployeeVM(Employee emp)
      {
         employee = emp;
      }
      public int IdEmpoyee
      {
         get { return employee.IdEmployee; }
         set
         {
            employee.IdEmployee = value;
            OnPropertyChanged("IdEmpoyee");
         }
      }
      public string FullName
      {
         get { return employee.FullName; }
         set
```

```
    {
        employee.FullName = value;
        OnPropertyChanged("FullName");
    }
}
public string DateOfBirth
{
    get { return employee.DateOfBirth; }
    set
    {
        employee.DateOfBirth = value;
        OnPropertyChanged("DateOfBirth");
    }
}
public string Passport
{
    get { return employee.Passport; }
    set
    {
        employee.Passport = value;
        OnPropertyChanged("Passport");
    }
}
public string PhoneNumber
{
    get { return employee.PhoneNumber; }
    set
    {
        employee.PhoneNumber = value;
        OnPropertyChanged("PhoneNumber");
    }
}
public string LoginEmployee
{
    get { return employee.LoginEmployee; }
    set
    {
        employee.LoginEmployee = value;
        OnPropertyChanged("LoginEmployee");
    }
}
public string PasswordEmployee
{
    get { return employee.PasswordEmployee; }
    set
    {
        employee.PasswordEmployee = value;
        OnPropertyChanged("PasswordEmployee");
    }
}
public int IdPosition
{
    get { return employee.IdPosition; }
    set
    {
        employee.IdPosition = value;
        OnPropertyChanged("IdEmpoyee");
    }
}
public string NamePosition
{
    get { return employee.NamePosition; }
    set
```

```
            {
                employee.NamePosition = value;
                OnPropertyChanged("NamePosition");
            }
        }
    }
}
```