**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**
**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**
**(СПбГУТ)**

**АРХАНГЕЛЬСКИЙ КОЛЛЕДЖ ТЕЛЕКОММУНИКАЦИЙ**
**ИМ. Б.Л. РОЗИНГА (ФИЛИАЛ) СПбГУТ**
**(АКТ (ф) СПбГУТ)**

# КУРСОВОЙ ПРОЕКТ

НА ТЕМУ

РАЗРАБОТКА ПОДСИСТЕМЫ

«ГАСТРОБАР. УЧЁТ ПРОДУКТОВ КУХНИ»

Л109. 25КП01. 010 ПЗ

(Обозначение документа)

МДК.02.01 Технология разработки

программного обеспечения

| | | | | |
|---|---|---|---|---|
| Студент | ИСПП-21 | | 08.12.2025 | И.Н. Кленин |
| | (Группа) | (Подпись) | (Дата) | (И.О. Фамилия) |
| Преподаватель | | | 09.12.2025 | Ю.С. Маломан |
| | | (Подпись) | (Дата) | (И.О. Фамилия) |

Архангельск 2025

# СОДЕРЖАНИЕ

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем курсовом проекте применяют следующие сокращения и обозначения:

БД – база данных

ОС – операционная система

ПО – программное обеспечение

СУБД – система управления базами данных

ER-модель – модель «сущность-связь»

IDE – интегрированная среда разработки

SQL – язык структурированных запросов

WPF – Windows Presentation Foundation

XAML – eXtensible Application Markup Language

SSMS – SQL Server Management Studio

# ВВЕДЕНИЕ

В современных условиях развития индустрии общественного питания особое значение приобретает эффективное управление товарно-материальными ценностями. Для гастробаров, где основной ассортимент представлен разнообразными алкогольными и безалкогольными напитками, ингредиентами для коктейлей, закусками и расходными материалами, критически важны точный учёт остатков и своевременное пополнение запасов. Ручные методы учёта и использование разрозненных электронных таблиц приводят к ошибкам, потерям и затрудняют анализ деятельности заведения.

Актуальность темы курсового проекта обусловлена необходимостью автоматизации складских и учётных операций гастробара, повышения прозрачности движения товарно-материальных ценностей и оперативного получения отчётной информации для управленческих решений. Разработка специализированной подсистемы «Гастробар. Учёт продуктов кухни» позволит систематизировать процессы приёма, хранения, списания и инвентаризации, а также обеспечить контроль остатков.

Целью курсового проекта является разработка программного продукта «BarInventoryApp», реализующего функции учёта ингредиентов и оформления заказов ингредиентов для гастробара в условиях закрытой локальной сети предприятия.

Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать предметную область, определить требования к функциональности, надёжности и безопасности разрабатываемого программного обеспечения;
- спроектировать архитектуру приложения, пользовательский интерфейс и структуру базы данных;

- реализовать основные программные модули, обеспечивающие авторизацию пользователей, ведение таблиц, создание заказов, формирование отчётов;

- обеспечить разграничение прав доступа в зависимости от ролей пользователей;

- разработать механизм экспорта и импорта данных;

- провести тестирование программного обеспечения, оценить его соответствие заданным требованиям;

- подготовить инструкцию по установке и эксплуатации подсистемы «Гастробар. Учёт продуктов кухни».

Объектом исследования являются процессы учёта ингредиентов и оформление заказов на ингредиенты в гастробаре. Предметом исследования является информационная система, реализующая автоматизацию указанных процессов.

Практическая значимость работы заключается в возможности применения разработанного программного продукта в реальном гастробаре для повышения эффективности управления запасами, снижения потерь и оптимизации расходов на закупку продукции.

# 1 Анализ и разработка требований

## 1.1 Назначение и область применения

Программный продукт «BarInventoryApp» представляет собой специализированную информационную систему для учёта ингредиентов и автоматизации оформления заказов ингредиентов для гастробара. Система предназначена для работы в локальной вычислительной сети предприятия без подключения к сети Интернет, что повышает уровень информационной безопасности и снижает риски несанкционированного удалённого доступа к данным.

Область применения программного продукта – локальная вычислительная сеть гастробара или сети небольших предприятий общественного питания, использующих единые стандарты учёта барной продукции. Система может быть адаптирована для различных форматов заведений (бар, ресторан, кафе, гастробар) с учётом специфики их ассортимента и бизнес-процессов[5].

## 1.2 Постановка задачи

Программный продукт «BarInventoryApp» должен обеспечивать автоматизацию следующих процессов:

- просмотр, добавление, редактирование и удаление записей об ингредиентах;
- просмотр, добавление, редактирование и удаление записей о заказах на ингредиенты;
- импорт и экспорт данных;
- разграничение прав доступа пользователей к функциональным возможностям системы в зависимости от их ролей.

Тип приложения: веб-приложение, работающее с централизованной БД, размещённой на локальном сервере или выделенном рабочем месте. Подключение пользователей осуществляется по локальной сети предприятия. В приложении реализованы:

- удобный графический интерфейс пользователя, адаптированный к рабочим процессам гастробара;
- хранение данных в БД;
- защиту данных за счёт аутентификации пользователей, разграничения прав доступа и работы в закрытой сети.

Разграничение прав доступа по ролям:

- для бармена: просмотр остатков ингредиентов;
- для менеджера: просмотр остатков ингредиентов, просматривать, создавать, редактировать, удалять и экспортировать данные о заказах;
- для администратора: просматривать, создавать, редактировать, удалять и импортировать записи об ингредиентах, просматривать, создавать, редактировать, удалять и экспортировать данные о заказах, управление учётными записями пользователей, настройка ролей и прав доступа.

Запуск приложения осуществляется с рабочего места пользователя в локальной сети гастробара. Для авторизации каждый пользователь вводит логин и пароль, после чего в соответствии с его ролью отображаются доступные разделы и функции.

На рисунке 1 показана диаграмма вариантов использования.

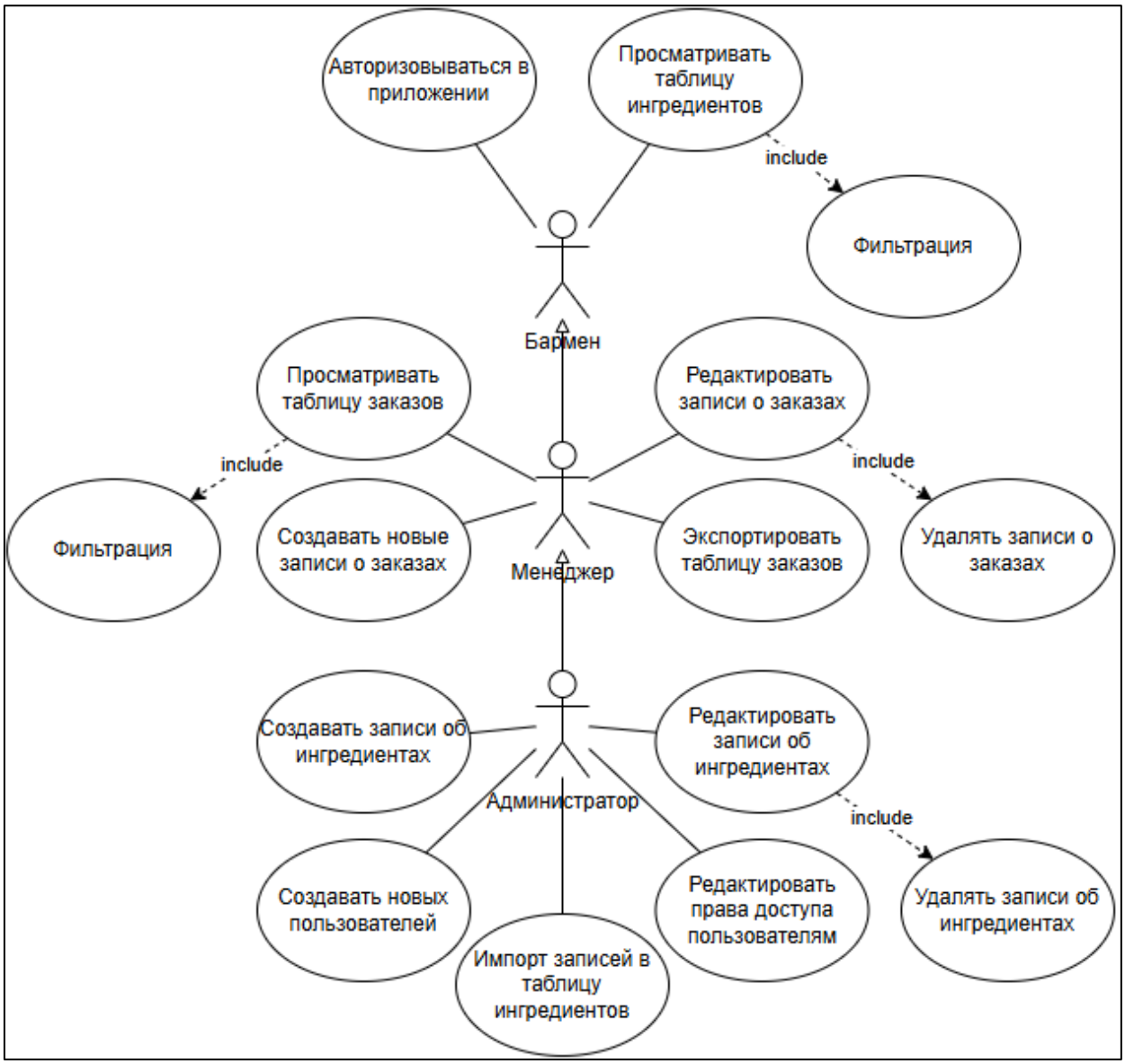


Рисунок 1 – Диаграмма вариантов использования

### 1.3 Выбор состава программных и технических средств

Работа с оконным приложением будет осуществляться на ПК и ноутбуках с ОС Windows (Windows 10 версии 1809 и новее, Windows 11).

В качестве СУБД выбрана Microsoft SQL Server, эта СУБД имеет прямую интеграцию с Visual Studio, удобна в использовании, обладает высокой производительностью, что позволяет эффективно обрабатывать данные о приёмах и пациентах в реальном времени.

Клиентская часть приложения будут разработаны на C#, так как с помощью этого языка можно эффективно создавать современные приложения с использованием технологии WPF.

Для разработки оконного приложения будет использоваться IDE Visual Studio 2022, так как эта среда предлагает удобные инструменты для работы с C#, включая инструменты для работы с Git и средства отладки.

Для функционирования системы на стороне сервера необходимы следующие программные и технические средства:

- ОС Windows 10 и выше,
- процессор частотой 2 ГГц,
- свободная оперативная память 4 ГБ,
- свободное место на диске не менее 500 МБ.
- ПО для конфигурирования, управления и администрирования сервера СУБД SSMS

Для функционирования системы на стороне сервера необходимы следующие программные и технические средства:

- БД SQL Server
- ОС Windows 10 и выше,
- процессор частотой 2 ГГц,
- свободная оперативная память 4 ГБ,
- свободное место на диске не менее 500 МБ.

# 2 Проектирование программного обеспечения

## 2.1 Проектирование интерфейса пользователя

Пользовательский интерфейс программного продукта «BarInventoryApp» разрабатывается с учётом принципов удобства использования, минимизации количества действий для выполнения типовых операций и визуальной ясности. Основными экранами системы являются:

- окно авторизации пользователя;
- главное окно с меню доступа к подсистемам;
- окно отображения текущих остатков ингредиентов;
- окно отображения текущих заказов ингредиентов;
- окно разграничения прав доступа.

Wireframe-макеты интерфейса показаны на рисунке 2.

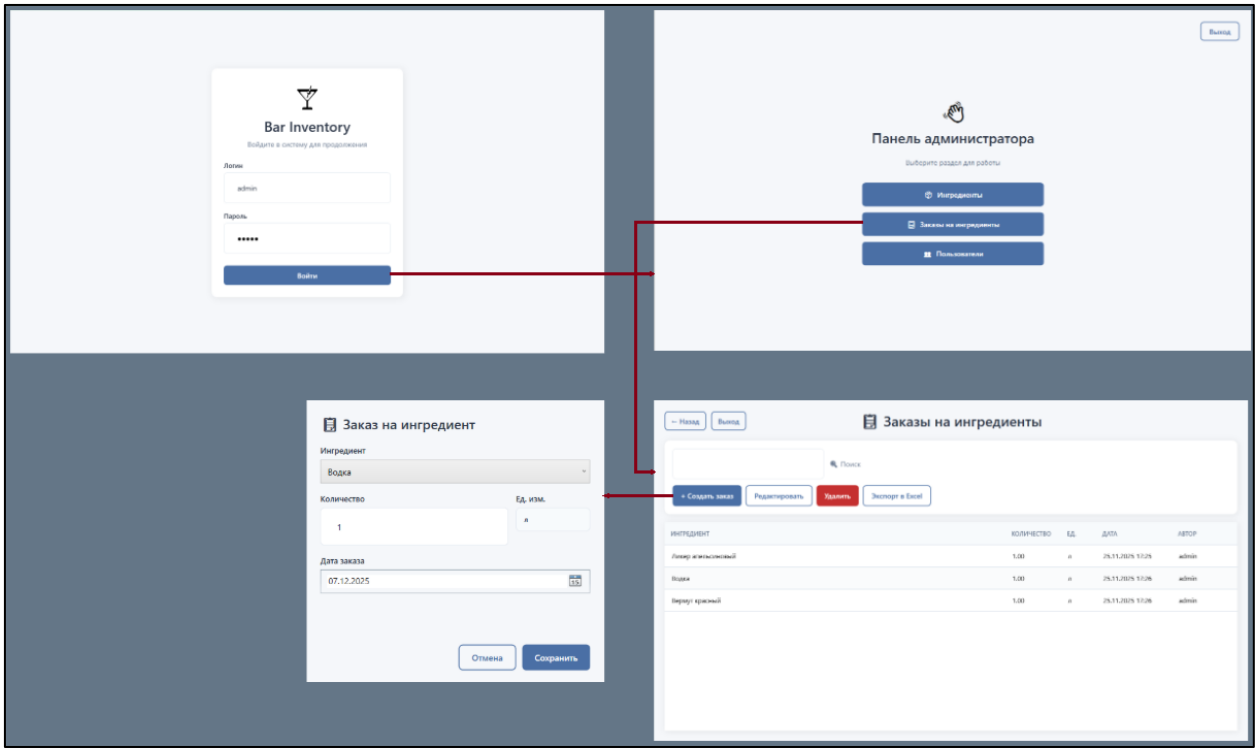


Рисунок 2 – Прототипы пользовательских окон подсистемы

Для обеспечения визуальной целостности приложения подобран единый набор цветов и типографики, используемы во всех формах WPF.

Цветовая схема включает:

- основной фон интерфейса – светлый (оттенки белого, светло-серого и голубого), позволяющий комфортно работать в течение длительного времени;
- акцентные элементы (кнопки основных операций, выделение активных элементов меню) – в оттенках синего или красного цвета;
- основной шрифт интерфейса – Segoe UI / Arial, размер 10–12 пт;
- для выделения заголовков разделов интерфейса используются полужирные начертания.

## 2.2 Разработка архитектуры программного обеспечения

Архитектура системы построена на клиент-серверном принципе и включает несколько взаимосвязанных элементов: серверную часть БД и клиентское WPF-приложение. Клиент взаимодействует с сервером обращения к БД[2].

Диаграмма развёртывания элементов программного комплекса представлена на рисунке 5.

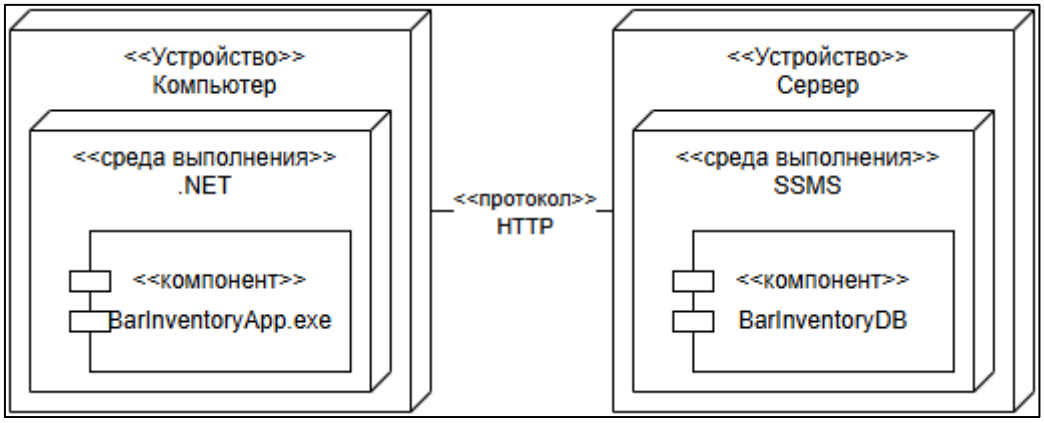


Рисунок 3 – Диаграмма развёртывания элементов программного комплекса

### 2.3 Проектирование базы данных

Для хранения данных, связанных с функционалом гастробара, была спроектирована реляционная БД. В БД представлены сущности для управления пользователями, ролями, ингредиентами и заказами[3].

ER-диаграмма предметной области отражает связи между таблицами и демонстрирует их структуру. Физическая модель базы данных разработана на базе СУБД SSMS

ER-диаграмма представлена на рисунке 6.

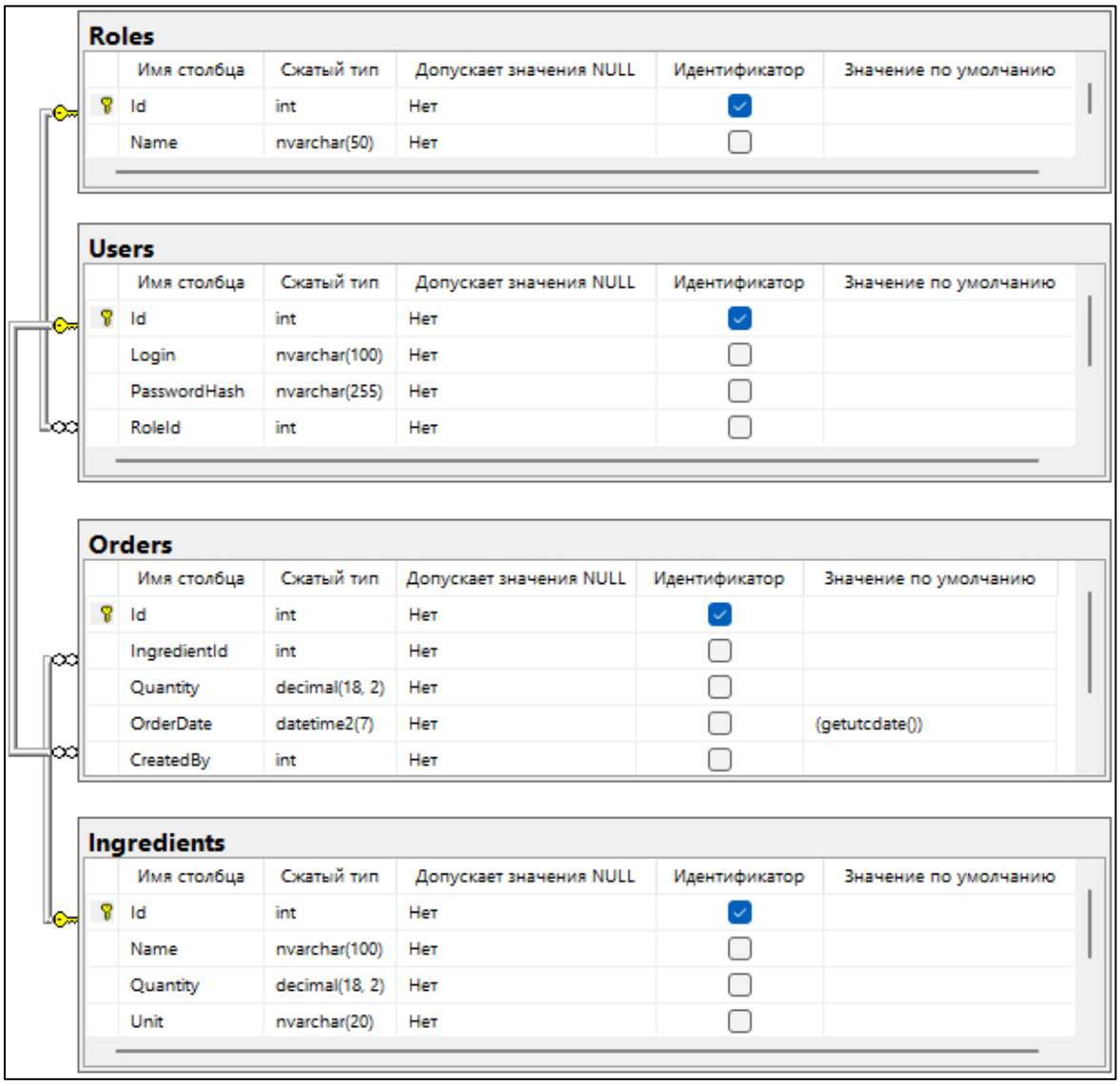


Рисунок 6 – ER-диаграмма

# 3 Разработка и интеграция модулей ПО

## 3.1 Разработка программных модулей

Для разработки программного продукта «Гастробар. Учёт продуктов кухни» используется среда разработки Microsoft Visual Studio 2022 с поддержкой .NET 8.0. Приложение построено на базе платформы WPF для создания настольного приложения с графическим интерфейсом пользователя. Архитектура приложения реализована по паттерну Model-View-View-Model (MVVM), что обеспечивает четкое разделение логики, данных и интерфейса, упрощает тестирование и поддержку кода.

Основной функционал реализован в клиентском приложении BarInventory, которое напрямую взаимодействует с БД через Entity Framework Core.

Код класса AuthService.cs представлен листингом 1. В нем прописана логика получения данных из БД о пользователях и их ролях.

Листинг 1 – Код метода AuthService.cs

```
using BarInventoryApp.DataContexts;
using BarInventoryApp.Models;
using Microsoft.EntityFrameworkCore;

namespace BarInventoryApp.Services;

// Сервис для аутентификации пользователей.
// Отвечает за поиск пользователя в базе данных по логину и
проверку пароля.
public class AuthService
{
    // Поле для хранения контекста базы данных (EF Core).
    // Через этот объект выполняются все запросы к таблицам БД.
    private readonly AppDbContext _context;
        // Сохраняем переданный контекст БД.
        // Если он не был передан (null) – выбрасываем
исключение.
        _context = context ?? throw new
ArgumentNullException(nameof(context));
    }
```

```
    public async Task<User?> AuthenticateAsync(string login,
string password)
    {
        // Проверяем, что логин не пустой и не состоит только из
пробелов.
        if (string.IsNullOrWhiteSpace(login))
            throw new ArgumentException("Логин не может быть
пустым.", nameof(login));

        // Проверяем, что пароль не пустой и не состоит только
из пробелов.
        if (string.IsNullOrWhiteSpace(password))
            throw new ArgumentException("Пароль не может быть
пустым.", nameof(password));
        // Формируем асинхронный запрос к базе данных:
        // _context.Users – это DbSet<User>, соответствующий
таблице Users.
        // Include(u => u.Role) – подгружаем связанную сущность
Role (JOIN с таблицей ролей),
        // чтобы потом сразу знать роль пользователя.
        // FirstOrDefaultAsync(...) – возвращает первого
найденного пользователя по логину
        // или null, если такой записи нет.
        var user = await _context.Users
            .Include(u => u.Role)
            .FirstOrDefaultAsync(u => u.Login == login);
        // Если пользователь найден и сохранённый в базе
хэш/пароль
        // совпадает с переданным значением – аутентификация
успешна.
        // (В реальном приложении здесь обычно сравнивается хэш
пароля, а не «голый» пароль.)
        if (user != null && user.PasswordHash == password)
            return user;
        // Если запись не найдена или пароль не совпал –
возвращаем null,
        // что выше по стеку будет воспринято как неуспешный
вход.
        return null;
    }
}
```

Метод выполняет запрос к БД с использованием Entity Framework Core для получения данных о пользователях и их ролях. Метод использует загрузку связанных данных для получения информации.

Код компонента MainViewModel представлен листингом 2. В нем реализована логика навигации между страницами.

Листинг 2 – Код компонента MainViewModel

```
using BarInventoryApp.Pages;
using Microsoft.Extensions.DependencyInjection;
using System.Windows.Controls;
namespace BarInventoryApp.ViewModels;
// ViewModel главного окна приложения, отвечающая за навигацию между страницами.
// Использует паттерн Dependency Injection для создания страниц через IServiceProvider.
public class MainViewModel
{
    // Поле для хранения ссылки на Frame (контейнер для отображения страниц в WPF).
    // Frame – это элемент управления, который может содержать одну страницу (Page) и поддерживает навигацию.
    private Frame? _frame;
    // Провайдер сервисов (Dependency Injection контейнер).
    // Через него получаем экземпляры страниц, зарегистрированные в контейнере зависимостей.
    private readonly IServiceProvider _serviceProvider;
    // Конструктор MainViewModel.
    public MainViewModel(IServiceProvider serviceProvider)
    {
        // Сохраняем провайдер сервисов.
        // Если он не передан (null) – выбрасываем исключение, так как без него навигация невозможна.
        _serviceProvider = serviceProvider ?? throw new ArgumentNullException(nameof(serviceProvider));
    }
    // Устанавливает Frame для навигации и сразу переходит на страницу авторизации.
    public void SetFrame(Frame frame)
    {
        // Проверяем, что Frame передан (не null).
        // Если null – выбрасываем исключение, так как без Frame навигация невозможна.
        _frame = frame ?? throw new ArgumentNullException(nameof(frame));
        // Сразу после установки Frame выполняем навигацию на страницу авторизации,
        // чтобы пользователь увидел форму входа при запуске приложения.
        NavigateTo<AuthorizationPage>();
    }
    /// Выполняет навигацию на страницу указанного типа.
    /// Страница создаётся через Dependency Injection контейнер.
    public void NavigateTo<T>() where T : Page
    {
        // Проверяем, что Frame был установлен ранее через метод SetFrame.
        // Если Frame не установлен – выбрасываем исключение с понятным сообщением.
```

```
        if (_frame == null)
            throw new InvalidOperationException("Frame не установлен. Вызовите SetFrame перед навигацией.");

        // Получаем экземпляр страницы указанного типа из контейнера зависимостей.
        // GetRequiredService<T>() создаёт объект страницы, автоматически разрешая все её зависимости
        // (например, если странице нужен AuthService, он будет автоматически подставлен).
        var page = _serviceProvider.GetRequiredService<T>();

        // Устанавливаем созданную страницу как содержимое Frame.
        // Это приводит к отображению страницы в главном окне приложения.
        _frame.Content = page;
    }

    // Выполняет навигацию на уже созданную страницу (переданную как параметр).
    // Используется, когда страница уже существует и не нужно создавать её через DI.
    public void Navigate(Page page)
    {
        // Проверяем, что Frame был установлен ранее через метод SetFrame.
        if (_frame == null)
            throw new InvalidOperationException("Frame не установлен. Вызовите SetFrame перед навигацией.");

        // Проверяем, что переданная страница не null.
        // Если null – выбрасываем исключение, так как нельзя отобразить пустую страницу.
        if (page == null)
            throw new ArgumentNullException(nameof(page));

        // Устанавливаем переданную страницу как содержимое Frame.
        // Это приводит к немедленному отображению страницы в главном окне приложения.
        _frame.Content = page;
    }
}
```

### 3.2 Реализация интерфейса пользователя

Пользовательский интерфейс реализован с использованием технологии WPF, что позволяет создавать современные оконные приложения с поддержкой привязки данных и гибкой настройкой внешнего вида. Для

описания внешнего вида форм и элементов управления используется язык разметки XAML[4].

В целях обеспечения единообразного стиля во всём приложении определены глобальные ресурсы:

- цветовая палитра (основной, фоновый, цвет акцентов);
- стили для кнопок, полей ввода, заголовков и таблиц;
- шаблоны данных для отображения строк таблиц и элементов списков.

Для работы с заказами разработан компонент OrdersPage, который представляет из себя страницу с данными, получаемыми из БД и кнопками, отображающими своё функциональное значение.

Вид страницы с заказами представлен на рисунке 7.

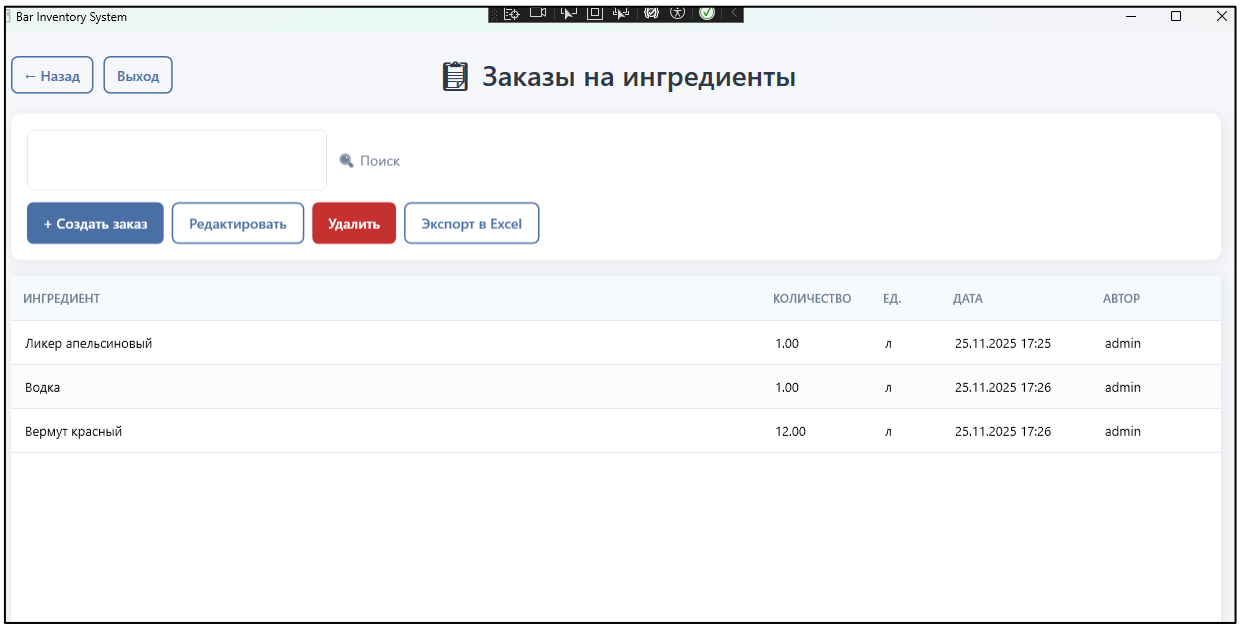


Рисунок 7 – Bar Inventory System. Вид страницы «Заказы на ингредиенты»

### 3.3 Разграничение прав доступа пользователей

Разграничение прав доступа реализовано при помощи таблицы Roles в БД. Система поддерживает следующие роли: Администратор, Менеджер и Бармен. Каждая роль определяет набор доступных функций и элементов

интерфейса для пользователя. Права администратора у пользователя появляются при Role.Name равном "Администратор", что позволяет получать доступ ко всем функциям системы, включая назначение ролей пользователям

Для аутентификации пользователей разработан метод OnLogin, который выполняет проверку учетных данных и определяет роль пользователя. Код метода представлен листингом 3.

Листинг 3 – Код метода OnLogin

```
// Асинхронный обработчик действия «Войти» (кнопка логина)
private async void OnLogin()
{
    // Проверяем, что поля логина и пароля не пустые и не
состоят только из пробелов
    if (string.IsNullOrWhiteSpace(Login) ||
string.IsNullOrWhiteSpace(Password))
    {
        // Если хотя бы одно поле пустое – показываем сообщение
пользователю
MessageBox.Show(ApplicationConstants.Messages.LoginPasswordRequi
red);
        // Прерываем выполнение метода, так как нет смысла
продолжать аутентификацию
        return;
    }
    try
    {
        // Обращаемся к сервису аутентификации и передаём
введённые логин и пароль
        // Метод AuthenticateAsync выполняется асинхронно
(например, обращается к БД)
        var user = await _authService.AuthenticateAsync(Login,
Password);

        // Если сервис вернул null – значит, пользователь с
такими данными не найден
        if (user == null)
        {
            // Сообщаем пользователю о неверных учётных данных

MessageBox.Show(ApplicationConstants.Messages.InvalidCredentials
);
            // Прерываем выполнение метода
            return;
        }
```

```
        // Если аутентификация успешна – сохраняем информацию о
текущем пользователе в сессионном контексте
        Session.CurrentUser = user;
        // В зависимости от роли аутентифицированного
пользователя выполняем навигацию на нужную страницу
        switch (user.Role.Name)
        {
            // Если роль – бармен, открываем страницу учёта
ингредиентов/продуктов кухни
            case ApplicationConstants.Roles.Barmen:
                _mainViewModel.NavigateTo<IngredientsPage>();
                break;
            // Если роль – менеджер, открываем панель управления
менеджера (отчёты, аналитика и т.п.)
            case ApplicationConstants.Roles.Manager:

_mainViewModel.NavigateTo<ManagerDashboardPage>();
                break;
            // Если роль – администратор, открываем панель
администратора (управление пользователями и настройками)
            case ApplicationConstants.Roles.Admin:
                _mainViewModel.NavigateTo<AdminDashboardPage>();
                break;
            // Если роль не распознана – информируем
пользователя об ошибке в настройках ролей
            default:

MessageBox.Show(ApplicationConstants.Messages.UnknownRole);
                break;
        }
    }
    // Перехватываем любые непредвиденные ошибки, возникшие при
аутентификации или навигации
    catch (Exception ex)
    {
        // Выводим сообщение об ошибке с текстом исключения,
чтобы упростить диагностику
        MessageBox.Show(

string.Format(ApplicationConstants.Messages.LoginError,
ex.Message),
            "Ошибка",
            MessageBoxButton.OK,
            MessageBoxImage.Error);
    }
}
```

### 3.4 Экспорт и импорт данных

В приложении реализована функция экспорта заказов на ингредиенты и импорта записей ингредиентов в формате Excel с использованием библиотеки ClosedXML. Компонент для реализации экспорта ExcelExpotService представлен листингом 4.

Листинг 4 – ExcelExportService

```
using BarInventoryApp.Models;
using ClosedXML.Excel;
using Microsoft.Win32;
using System.Windows;
using BarInventoryApp.Constants;
namespace BarInventoryApp.Services;
// Сервис для экспорта списка заказов в файл формата Excel
(.xlsx).
// Использует библиотеку ClosedXML для создания и сохранения
Excel-документов.
public class ExcelExportService
{
    // Фильтр для диалога сохранения файла (показывает только
файлы .xlsx)
    private const string ExcelFileFilter = "Excel files
(*.xlsx)|*.xlsx";

    // Название листа (worksheet) в Excel-файле, куда будут
записаны данные
    private const string WorksheetName = "Заказы";

    // Формат отображения даты и времени в ячейках Excel
    private const string DateFormat = "dd.MM.yyyy HH:mm";

    // Префикс для имени файла при сохранении
    private const string FileNamePrefix = "Заказы_";

    // Формат даты и времени для имени файла (без пробелов и
двоеточий, чтобы имя было валидным)
    private const string FileNameDateFormat = "yyyy-MM-dd_HH-
mm";

    // Заголовок окна сообщения об успешном экспорте
    private const string SuccessTitle = "Успех";

    // Экспортирует список заказов в Excel-файл.
    public void ExportOrders(List<Order> orders)
```

```
    {
        // Проверяем, что список заказов не null.
        // Если null – выбрасываем исключение, так как
экспортировать нечего.
        if (orders == null)
            throw new ArgumentNullException(nameof(orders));
        // Создаём новый пустой Excel-документ (workbook).
        var workbook = new XLWorkbook();
        // Добавляем в документ новый лист с указанным именем.
        // Этот лист будет содержать таблицу с данными заказов.
        var worksheet = workbook.Worksheets.Add(WorksheetName);
        // Заполняем заголовки таблицы в первой строке листа:
        // Ячейка (1, 1) – первый столбец, первая строка –
название ингредиента
        worksheet.Cell(1, 1).Value = "Ингредиент";
        // Ячейка (1, 2) – второй столбец, первая строка –
количество
        worksheet.Cell(1, 2).Value = "Количество";
        // Ячейка (1, 3) – третий столбец, первая строка –
единица измерения
        worksheet.Cell(1, 3).Value = "Ед.";
        // Ячейка (1, 4) – четвёртый столбец, первая строка –
дата заказа
        worksheet.Cell(1, 4).Value = "Дата";
        // Ячейка (1, 5) – пятый столбец, первая строка – автор
заказа (кто создал)
        worksheet.Cell(1, 5).Value = "Автор";
        // Проходим по всем заказам из переданного списка
        for (int i = 0; i < orders.Count; i++)
        {
            // Получаем текущий заказ из списка
            var order = orders[i];
            // Вычисляем номер строки в Excel (i + 2, так как
первая строка – заголовки).
            // Например, для первого заказа (i=0) это будет
строка 2.
            var row = i + 2;
            // Заполняем ячейки строки данными из заказа:
            // Название ингредиента (если ингредиент есть, иначе
пустая строка)
            worksheet.Cell(row, 1).Value =
order.Ingredient?.Name ?? string.Empty;
            // Количество заказанного ингредиента
            worksheet.Cell(row, 2).Value = order.Quantity;
            // Единица измерения ингредиента (если есть, иначе
пустая строка)
            worksheet.Cell(row, 3).Value =
order.Ingredient?.Unit ?? string.Empty;
            // Дата заказа, отформатированная в указанном
формате (например, "25.11.2025 14:30")
            worksheet.Cell(row, 4).Value =
order.OrderDate.ToString(DateFormat);
```

```
            // Логин пользователя, создавшего заказ (если есть,
иначе пустая строка)
            worksheet.Cell(row, 5).Value =
order.CreatedByNavigation?.Login ?? string.Empty;

        // Создаём диалог сохранения файла (стандартное окно
Windows "Сохранить как...")
        var dialog = new SaveFileDialog
        {
            // Устанавливаем фильтр, чтобы пользователь видел
только файлы .xlsx
            Filter = ExcelFileFilter,
            // Предлагаем имя файла по умолчанию: "Заказы_2025-
11-25_14-30.xlsx"
            // (текущая дата и время в формате без пробелов)
            FileName =
$"{FileNamePrefix}{DateTime.Now:FileNameDateFormat}.xlsx"
        };
        // Показываем диалог сохранения файла.
        // Если пользователь выбрал место сохранения и нажал
"Сохранить" - ShowDialog() вернёт true.
        if (dialog.ShowDialog() == true)
        {
            // Сохраняем созданный Excel-документ по указанному
пользователем пути.
            workbook.SaveAs(dialog.FileName);
            // Показываем сообщение об успешном экспорте данных.
            MessageBox.Show(

ApplicationConstants.Messages.ExcelExportSuccess,
                SuccessTitle,
                MessageBoxButton.OK,
                MessageBoxImage.Information);
        }
        // Если пользователь отменил сохранение (нажал "Отмена")
- ничего не делаем,
        // файл не сохраняется, метод завершается без ошибок.
    }
}
```

# 4 Тестирование и отладка ПО

## 4.1 Структурное тестирование

Структурное тестирование программного кода «BarInventoryApp» выполнено методом «белого ящика» с использованием модульных тестов для ключевых компонентов программной логики. Для реализации модульных тестов применён фреймворк xUnit[1].

Основное внимание уделено тестированию:

- редактированию записей о заказах;
- удаление записей о заказах;
- проверок корректности вводимых данных.

На листинге 6 показан код xUnit – теста, редактирование и удаление записей со страницы – заказы.

Листинг 6 – код xUnit теста

```
using BarInventoryApp.Models;
using BarInventoryApp.Services;
using BarInventoryApp.ViewModels;
using Moq;
using System.Windows;
using Xunit;
namespace BarInventoryApp.Tests.ViewModels;
// Тесты для операций редактирования и удаления заказов в
OrdersViewModel
public class OrdersViewModelEditDeleteTests
{
    private readonly Mock<OrderService> _mockOrderService;
    private readonly Mock<IngredientService>
_mockIngredientService;
    private readonly Mock<UserService> _mockUserService;
    private readonly Mock<ExcelExportService> _mockExcelService;
    private readonly Mock<MainViewModel> _mockMainViewModel;
    private readonly OrdersViewModel _viewModel;
    public OrdersViewModelEditDeleteTests()
    {
        _mockOrderService = new Mock<OrderService>();
        _mockIngredientService = new Mock<IngredientService>();
        _mockUserService = new Mock<UserService>();
        _mockExcelService = new Mock<ExcelExportService>();
```

```
        _mockMainViewModel = new Mock<MainViewModel>();
        _viewModel = new OrdersViewModel(
            _mockOrderService.Object,
            _mockIngredientService.Object,
            _mockUserService.Object,
            _mockExcelService.Object,
            _mockMainViewModel.Object);
    }
    // Тест команды редактирования заказа.
    /// Проверяет, что при выборе заказа открывается диалог
редактирования.
    [Fact]
    public void EditCommand_WithSelectedOrder_OpensEditDialog()
    {
        // Arrange
        var testOrder = new Order
        {
            Id = 1,
            IngredientId = 1,
            Quantity = 5.0m,
            OrderDate = DateTime.Now,
            CreatedBy = 1,
            Ingredient = new Ingredient { Id = 1, Name =
"Водка", Unit = "литры" },
            CreatedByNavigation = new User { Id = 1, Login =
"manager" }
        };
        _viewModel.SelectedItem = testOrder;
        // Act - вызываем команду редактирования через
reflection
        var onEditMethod =
typeof(OrdersViewModel).GetMethod("OnEdit",
System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance);
        // Проверяем, что метод существует и может быть вызван
        Assert.NotNull(onEditMethod);
        // Assert - в реальном приложении здесь должен открыться
диалог
        // В тесте мы проверяем, что SelectedItem установлен
корректно
        Assert.Equal(testOrder, _viewModel.SelectedItem);
    }
    // Тест команды редактирования без выбранного заказа.
    // Проверяет, что при отсутствии выбранного заказа
появляется сообщение.
    [Fact]
    public void EditCommand_WithoutSelectedOrder_ShowsMessage()
    {
        // Arrange
        _viewModel.SelectedItem = null;
        // Act & Assert - метод OnEdit должен корректно
обработать отсутствие выбранного заказа
```

```
        var onEditMethod =
typeof(OrdersViewModel).GetMethod("OnEdit",
System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance);
        Assert.NotNull(onEditMethod);
        // В реальном приложении здесь появится MessageBox с
сообщением "Выберите заказ для редактирования"
    }
    // Тест команды удаления заказа.
    // Проверяет, что заказ удаляется через сервис и список
обновляется.
    [Fact]
    public void
DeleteCommand_WithSelectedOrder_DeletesSuccessfully()
    {
        // Arrange
        var testOrder = new Order
        {
            Id = 1,
            IngredientId = 1,
            Quantity = 5.0m,
            OrderDate = DateTime.Now,
            CreatedBy = 1,
            Ingredient = new Ingredient { Id = 1, Name =
"Водка", Unit = "литры" },
            CreatedByNavigation = new User { Id = 1, Login =
"manager" }
        };
        // Устанавливаем тестовые данные в приватное поле
_allOrders
        var allOrdersField =
typeof(OrdersViewModel).GetField("_allOrders",
System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance);
        var testOrders = new List<Order> { testOrder };
        allOrdersField?.SetValue(_viewModel, testOrders);
        _viewModel.SelectedItem = testOrder;
        // Настраиваем мок для успешного удаления
        _mockOrderService.Setup(s =>
s.DeleteAsync(1)).ReturnsAsync(true);
        _mockOrderService.Setup(s =>
s.GetAllWithDetailsAsync()).ReturnsAsync(new List<Order>());
        // Act - вызываем команду удаления
        var onDeleteMethod =
typeof(OrdersViewModel).GetMethod("OnDelete",
System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance);
        onDeleteMethod?.Invoke(_viewModel, null);
        // Assert - проверяем, что сервис удаления был вызван с
правильным ID
        _mockOrderService.Verify(s => s.DeleteAsync(1),
Times.Once);
```

```
        _mockOrderService.Verify(s =>
s.GetAllWithDetailsAsync(), Times.Once);
    }
    /// Тест команды удаления без выбранного заказа.
    /// Проверяет, что при отсутствии выбранного заказа ничего
не происходит.
    [Fact]
    public void DeleteCommand_WithoutSelectedOrder_DoesNothing()
    {
        // Arrange
        _viewModel.SelectedItem = null;
        // Act - вызываем команду удаления
        var onDeleteMethod =
typeof(OrdersViewModel).GetMethod("OnDelete",
System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance);
        onDeleteMethod?.Invoke(_viewModel, null);
        // Assert - проверяем, что сервис не был вызван
        _mockOrderService.Verify(s =>
s.DeleteAsync(It.IsAny<int>()), Times.Never);
    }
    /// <summary>
    /// Тест обработки ошибки при удалении заказа.
    /// Проверяет, что ошибка удаления обрабатывается корректно.
    /// </summary>
    [Fact]
    public void DeleteCommand_WhenDeleteFails_HandlesError()
    {
        // Arrange
        var testOrder = new Order
        {
            Id = 1,
            IngredientId = 1,
            Quantity = 5.0m,
            OrderDate = DateTime.Now,
            CreatedBy = 1,
            Ingredient = new Ingredient { Id = 1, Name =
"Водка", Unit = "литры" },
            CreatedByNavigation = new User { Id = 1, Login =
"manager" }
        };
        _viewModel.SelectedItem = testOrder;
        // Настраиваем мок для имитации ошибки удаления
        _mockOrderService.Setup(s => s.DeleteAsync(1))
            .ThrowsAsync(new Exception("Database error"));
        // Act - вызываем команду удаления
        var onDeleteMethod =
typeof(OrdersViewModel).GetMethod("OnDelete",
System.Reflection.BindingFlags.NonPublic |
System.Reflection.BindingFlags.Instance);
        onDeleteMethod?.Invoke(_viewModel, null);
        // Assert - проверяем, что сервис был вызван, несмотря
на ошибку
```

```
        _mockOrderService.Verify(s => s.DeleteAsync(1),
Times.Once);
        // В реальном приложении здесь появится MessageBox с
ошибкой
    }
    // Тест проверки доступности команд редактирования и
удаления.
    // Проверяет, что команды всегда доступны (CanExecute
возвращает true).
    [Fact]
    public void EditAndDeleteCommands_AreAlwaysEnabled()
    {
        // Assert - команды редактирования и удаления всегда
доступны
        Assert.True(_viewModel.EditCommand.CanExecute(null));
        Assert.True(_viewModel.DeleteCommand.CanExecute(null));
    }
}
```

### 4.2 Функциональное тестирование

Функциональное тестирование выполнено методом «чёрного ящика» на уровне пользовательских сценариев. Определены и протестированы основные сценарии работы пользователей различными ролями (бармен, менеджер, администратор). Тестирование методом «чёрного ящика» показано в таблице 2.

Таблица 2 – Тестирование методом «чёрного ящика»

| № | Действие | Ожидаемый результат | Полученный результат |
|---|---|---|---|
| 1 | Ввод логина(ivanov) и пароль(hello) в соответствующие поля в окне авторизации. Роль – бармен. | Пользователь переходит на страницу «Остатки ингредиентов». | Ожидаемый результат соответствует полученному. |

| | | | |
|---|---|---|---|
| 2 | Открыть форму редактирования заказа на ингредиенты, выбор ингредиенты из выпадающего списка, количество оставляем пустым. Роль – менеджер. | Количество изменилось на минимальное значение в 1 единицу. | Ожидаемый результат соответствует полученному. |
| 3 | Открыть форму управления пользователями, в логин написать «new», пароль оставить пустым, нажать кнопку создать. Роль – администратор. | Появляется MessageBox, с требованием заполнить поля логина и пароля. | Ожидаемый результат соответствует полученному. |

# 5 Инструкция по эксплуатации ПО

## 5.1 Установка программного обеспечения

Для установки информационной подсистемы «Гастробар. Учёт продуктов кухни» на стороне сервера необходимо:

- убедиться, что сервер соответствует минимальным системным требованиям Microsoft SQL Server 2019 Express;
- установить Microsoft SQL Server 2019 Express;
- осуществить подключение к серверу и выполнить SQL-скрипт по созданию базы данных (файл «BarInventoryDB.sql»).

Для установки приложения на стороне клиента необходимо: установить клиентское приложение; установить .NET 8.0 Runtime (если не установлен); при необходимости настроить строку подключения к базе данных в файле конфигурации приложения.

Логины и пароли всех типов пользователей:

- Бармен – логин: ivanov; пароль: hello;
- Менеджер – логин: petrova; пароль: peanut;
- Администратор – логин: admin; пароль: admin

## 5.2 Инструкция по работе

Для запуска приложения требуется запустить исполняемый файл приложения. Пользователю откроется окно авторизации (рисунок 8), в котором необходимо указать учетные данные пользователя или перейти к регистрации новой учетной записи.

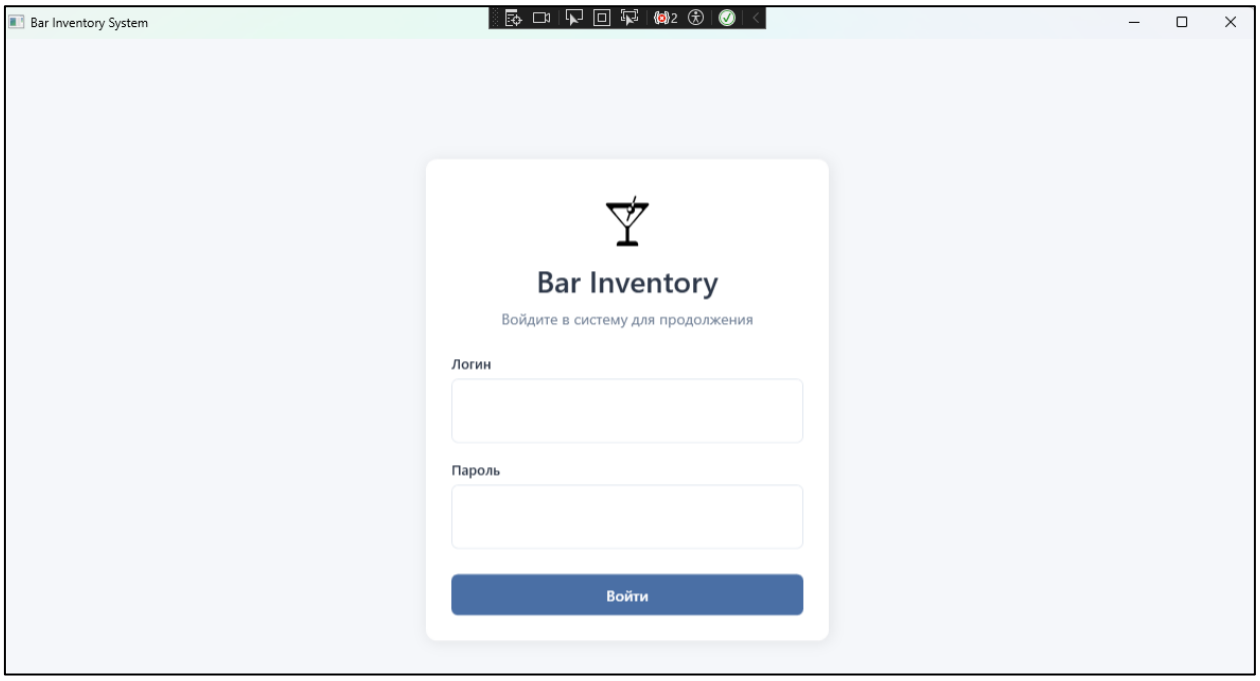


Рисунок 8 – Bar Inventory System. Вид страницы «Авторизация»

После успешной авторизации открывается главное окно приложения с навигацией. Доступные пункты меню зависят от роли пользователя в системе. На рисунке 9 предоставлена страница менеджера.

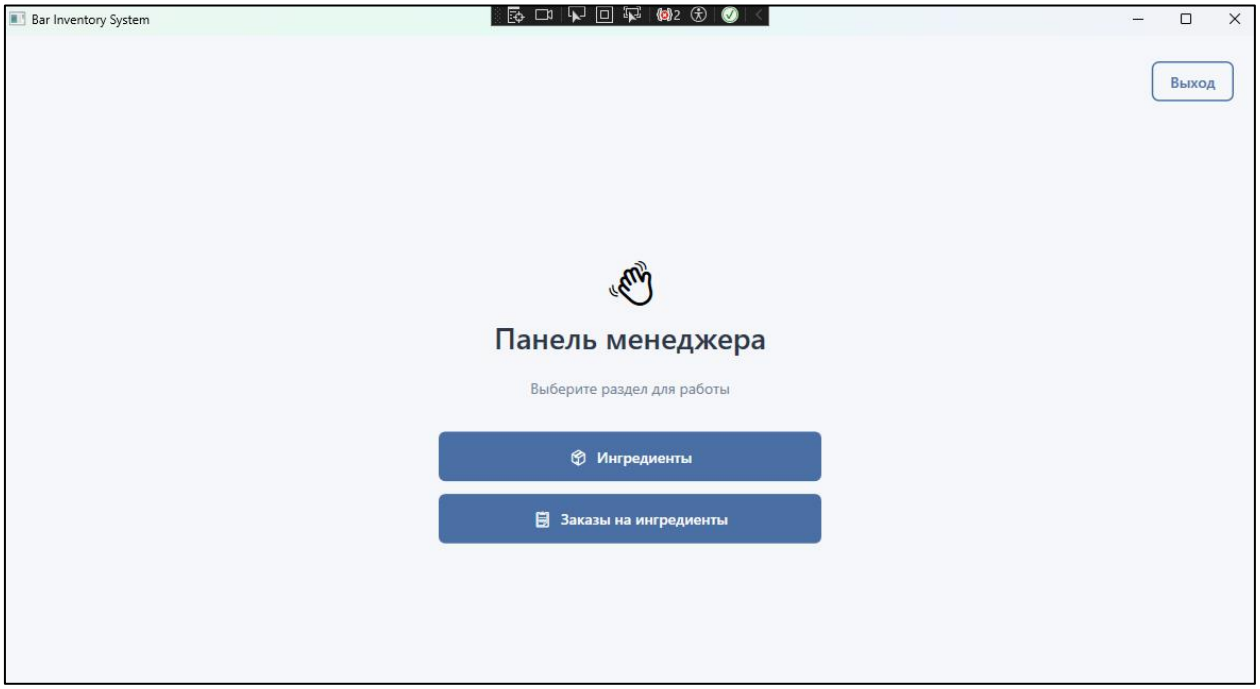


Рисунок 9 – Bar Inventory System. Вид страницы «Панель менеджера»

Для роли бармен не предусмотрено окно с навигаций, у него сразу же открывается окно с остатками ингредиентов.

Для менеджера предусмотрен неполный список меню: «Ингредиенты» и «Заказы на ингредиенты». На странице «Ингредиенты» он может просматривать остатки ингредиентов, а на странице «Заказы на ингредиенты» он может просматривать, создавать, редактировать, удалять и экспортировать данные о заказах.

Для роли администратора доступны все пункты меню: «Ингредиенты», «Заказы на ингредиенты» и «Пользователи». На странице «Ингредиенты» он может добавлять, редактировать, удалять и импортировать данные. На странице «Пользователи» он может создавать новых пользователей и менять роли другим пользователям.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта разработана подсистема «Гастробар. Учёт продуктов кухни» в составе программного продукта «BarInventoryApp», предназначенная для автоматизации учёта ингредиентов и заказов на ингредиенты для гастробара. В ходе работы был проведён анализ предметной области, сформулированы функциональные и нефункциональные требования к системе, спроектированы архитектура программного обеспечения, пользовательский интерфейс и структура базы данных.

Реализованы основные программные модули, обеспечивающие учёт ингредиентов, формирование заказов на ингредиенты, экспорт данных о заказах и распределение прав доступа. Обеспечена работа приложения в условиях закрытой локальной сети предприятия, что повышает уровень информационной безопасности и снижает риски несанкционированного доступа к данным.

Проведены структурное и функциональное тестирование программного обеспечения, подтвердившие работоспособность и соответствие системы заданным требованиям. Разработана инструкция по установке и эксплуатации, позволяющая внедрить подсистему в практику работы гастробара.

Практическая значимость разработанного решения заключается в возможности его использования для повышения эффективности управления запасами продукции, сокращения потерь и оптимизации процессов закупки и списания товарно-материальных ценностей. В дальнейшем возможна доработка подсистемы, включая интеграцию с системой учёта продаж, расширение функций аналитики и внедрение мобильного интерфейса для проведения инвентаризаций.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Бек, К. Экстремальное программирование: разработка через тестирование. – Санкт-Петербург : Питер, 2021. – 224 с. – URL: https://ibooks.ru/bookshelf/376974/reading (дата обращения: 24.11.2025). – Режим доступа: для зарегистрир. пользователей. – Текст: электронный.

2 Гагарина, Л. Г. Технология разработки программного обеспечения : учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул ; под ред. Л. Г. Гагариной. – Москва : ФОРУМ : ИНФРА-М, 2025. – 400 с. – URL: https://znanium.ru/catalog/product/2178802 (дата обращения: 24.11.2025). – Режим доступа: по подписке. – Текст : электронный.

3 Мартишин, С. А. Базы данных. Практическое применение СУБД SQL и NoSQL-типа для проектирования информационных систем : учебное пособие / С. А. Мартишин, В. Л. Симонов, М. В. Храпченко. – Москва : ФОРУМ : ИНФРА-М, 2024. – 368 с. – Текст : электронный. – URL: https://znanium.ru/catalog/product/2096940 (дата обращения: 24.11.2025). – Режим доступа: по подписке.

4 Тидвелл, Д. Разработка интерфейсов. Паттерны проектирования. 3-е изд. – Санкт-Петербург : Питер, 2022. – 560 с. – Текст : электронный. – URL: https://ibooks.ru/bookshelf/386796/reading (дата обращения: 24.11.2025). – Режим доступа: для зарегистрир. пользователей.

5 Федорова, Г. Н. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности : учебное пособие. — Москва : КУРС : ИНФРА-М, 2024. — 336 с. – URL: https://znanium.ru/catalog/product/2083407 (дата обращения: 24.11.2025). – Режим доступа: по подписке. – Текст : электронный.