**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 | |
| «ТОВАРНЫЙ МОНИТОР» | |
| Л109. 24КП01. 013 ПЗ | |

(Обозначение документа)

|  |
| --- |
| МДК.02.01 Технология разработки |
| программного обеспечения |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИСПП-11 | | 05.12.2024 | А.М. Некрасов |
|  | (Группа) | (Подпись) | (Дата) | (И.О. Фамилия) |
| Преподаватель |  |  | 05.12.2024 | Ю.С. Маломан |
|  |  | (Подпись) | (Дата) | (И.О. Фамилия) |

Архангельск 2024

**СОДЕРЖАНИЕ**

Перечень сокращений и обозначений 3

Введение 4

1 Анализ и разработка требований 6

1.1 Назначение и область применения 6

1.2 Постановка задачи 6

1.3 Описание алгоритма функционирования системы 6

1.4 Выбор состава программных и технических средств 7

2 Проектирование программного обеспечения 9

2.1 Проектирование интерфейса пользователя 9

2.2 Разработка архитектуры программного обеспечения 9

2.3 Проектирование базы данных 10

3 Разработка и интеграция модулей программного обеспечения 11

3.1 Разработка программных модулей 11

3.2 Реализация интерфейса пользователя 13

3.3 Разграничение прав доступа пользователей 17

3.4 Экспорт данных 18

4 Тестирование и отладка программного обеспечения 21

4.1 Структурное тестирование 21

4.2 Функциональное тестирование 23

5 Инструкция по эксплуатации программного обеспечения 24

5.1 Установка программного обеспечения 24

5.2 Инструкция по работе 25

Заключение 29

Список использованных источников 30

**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

HTTP - протокол передачи гипертекста

ID - идентефикатор

IDE - интегрированная среда разработки

JSON - текстовый формат обмена данными

JWT - веб токен

REST - Представительское состояние передачи

WPF - платформа пользовательского интерфейса, которая создаёт клиентские приложения для настольных компьютеров.

**ВВЕДЕНИЕ**

Актуальность разрабатываемого проекта заключается в том, что он предоставит решение важной проблемы в области управления товарными запасами и взаимодействия с клиентами. В условиях современного рынка, где конкуренция возрастает, а потребительские предпочтения меняются, эффективное управление товарами становится ключевым фактором успеха для бизнеса.

Современные технологии предоставляют множество возможностей для автоматизации процессов. Разработка приложения с интуитивно понятным и удобным интерфейсом будет способствовать более эффективному взаимодействию пользователей с системой. Взаимодействие между приложением и сервером обеспечит эффективное храниние и обработку информации о товарах и заказах. Это позволит администраторам в реальном времени отслеживать состояние запасов, анализировать продажи и прогнозировать потребности, что, в свою очередь, поможет избежать как избытка, так и нехватки товаров. Таким образом, проект "Товарный монитор" не только отвечает на актуальные потребности бизнеса, но и использует современные подходы для достижения поставленных целей.

Целью курсового проектирования является разработка комплексного решения для управления товарами, которое позволит администраторам эффективно управлять своим ассортиментом, отслеживать заказы и взаимодействовать с клиентами.

Для достижения поставленной цели требуется решить следующие задачи:

* провести сбор требований целевой аудитории,
* проанализировать информационные источники по предметной области,
* спроектировать архитектуру приложения,
* спроектировать диаграмму вариантов использования приложения,
* выбрать состав программных и технических средств для реализации приложения,
* спроектировать БД,
* спроектировать интерфейс оконного приложения,
* создать БД в выбранной СУБД,
* разработать API для некоторых функций приложения,
* реализовать разграничение прав доступа пользователей,
* разработать интерфейс оконного приложения,
* разработать оконное приложение,
* реализовать экспорт данных в виде файлов Excel,
* реализовать работу приложения с сервером БД при помощи REST API,
* выполнить структурное тестирование ПО,
* выполнить функциональное тестирование ПО,
* разработать программную и эксплуатационную документацию.

В результате выполнения поставленных задач будет разработано оконное приложение, которое обеспечит эффективное управление товарами.

**1 Анализ и разработка требований**

* 1. **Назначение и область применения**

Разрабатываемое оконное приложение «Товарный монитор» предназначено для работников магазина, желающих оптимизировать процесс управления товарными запасами. Приложение упростит процесс добавления и редактирования информации о товарах, а также просмотра и управления заказами. Администраторы смогут эффективно управлять своим ассортиментом, отслеживать заказы и взаимодействовать с клиентами, что повысит общую эффективность работы бизнеса.

* 1. **Постановка задачи**

Необходимо разработать оконное приложение, которое предоставит доступ к следующей функциональности:

- авторизации пользователей,

- добавлению и редактированию информации о товарах,

- просмотру информации о заказах,

- генерации отчетов по продажам.

* 1. **Описание алгоритма функционирования системы**

При запуске приложения отображается начальная страница, на которой присутствует возможность авторизоваться, после авторизации открывается главное окно приложения.

Администратор может перейти на страницы «Товары», «Заказы», «Категории», «Клиенты».

Администратор может добавлять и редактировать информацию о товарах и категорях.

На рисунке 1 изображена диаграмма вариантов использования приложения.

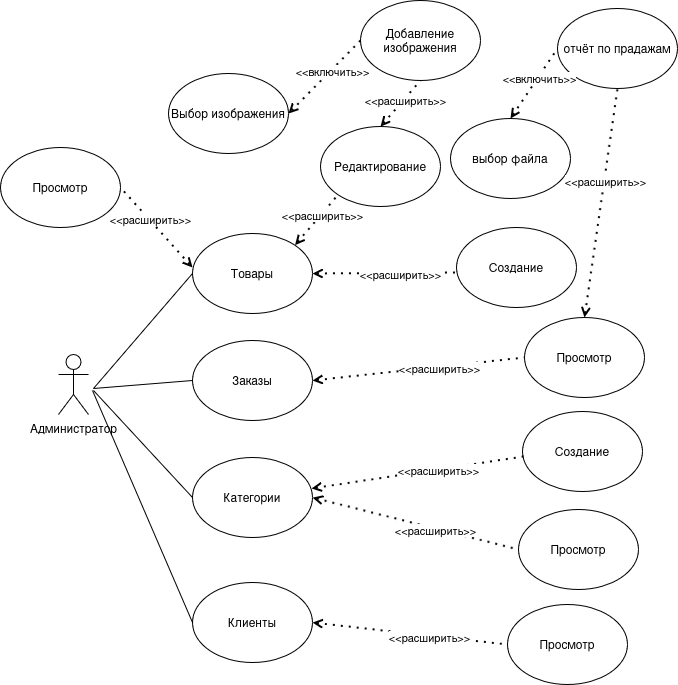


Рисунок 1 – Диаграмма вариантов использования

* 1. **Выбор состава программных и технических средств**

Согласно цели проекта требуется создать оконное приложение для учёта товаров и заказов.

Работа с оконным приложением будет осуществляться на компьютере с установленной операционной системой Windows 10 с интернет-подключением.

В качестве СУБД выбрана MySQL 8.0.40, эта СУБД была выбрана за свою простоту в развёртывании и удобство в использовании.

Приложение будет написано на языке программирования C# с использованием фреймворка WPF, который предоставляет удобные инструменты для создания оконных приложений.

Для разработки приложения будет использоваться IDE Visual Studio 22, так как она имеет функцию быстрой перезагрузки и удобный предпросмотр страницы.

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

* ОС Ubuntu версии 18.04 и выше,
* сервер БД: MySQL не ниже 8.0.40,
* процессор с 2 ядрами по 2 ГГц,
* оперативная память объемом 2 ГБ,
* ПО для конфигурирования, управления и администрирования сервера БД: MySQL Workbench последней версии,
* программное обеспечение для работы API: Docker, docker-compose.

Для функционирования системы на стороне клиента достаточны следующие программные и технические средства:

* ОС Windows версей не ниже 10,
* процессор с частотой 1 ГГц или быстрее,
* оперативная память в объеме 1 ГБ и выше,
* свободное место в хранилище 40 МБ,
* постоянное интернет-подключение.

**2 Проектирование программного обеспечения**

**2.1 Проектирование интерфейса пользователя**

В рамках разработки оконного приложения «BrosShop» спроектирован интерфейс пользователя при помощи сайта draw.io. Эти визуальные представления позволяют наглядно увидеть структуру приложения, его основные элементы и функциональность.

Страница «Товары» приложения показана рисунке 2.

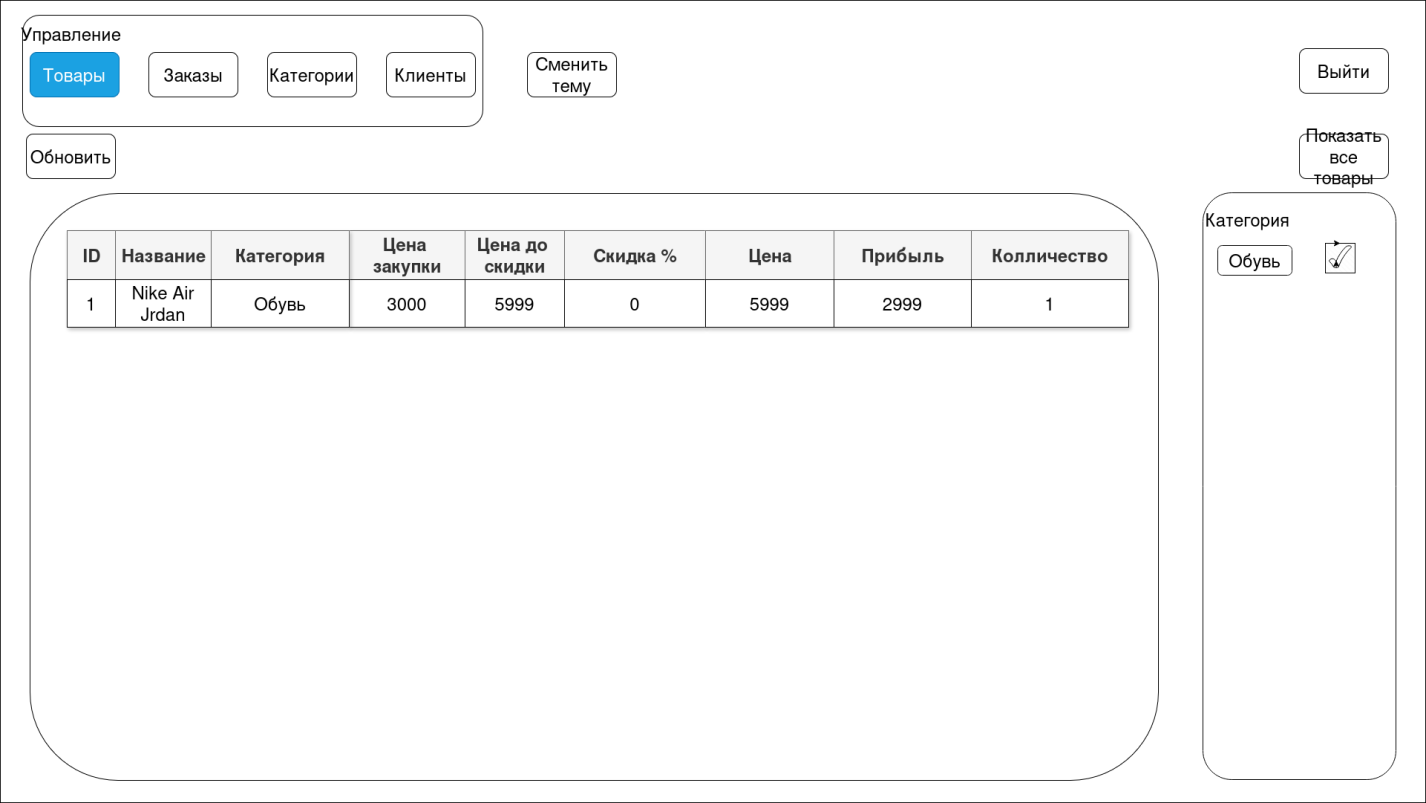


Рисунок 2 – draw.io. Вид страницы «Товары» приложения BrosShop

**2.2 Разработка архитектуры программного обеспечения**

Приложение предназначено для визуального добавления в БД данных о товарах и просмотре информации о товарах и заказах. Архитектура приложения построена на основе клиент-серверной модели и включает в себя несколько ключевых компонентов: серверная часть приложения, оконное приложение, БД[5].

Для серверной части будет создан API, позволяющий администратору взаимодействовать с сервером.

В качестве БД будет использовано СУБД MySQL. БД должна хранить в себе информацию о товарах, заказах, клиентах и связанные таблицы.

Приложение будет написано на фреймворке WPF. Интерфейс разработан с учетом удобства и простоты использования. Взаимодействие с сервером будет происходить при помощи HTTP-запросов к API, ответы будут приходить в формате JSON и File. Для отправки запросов будет использоваться веб-клиент HttpClient. Токен авторизации будет храниться в свойствах проложения, которые учитываются при запуске.

**2.3 Проектирование базы данных**

Требуется разработать БД для хранения и управления данными[4].

Модели БД созданы при помощи MySQL Workbench. На рисунке 3 в виде ERD показана физическая модель БД.

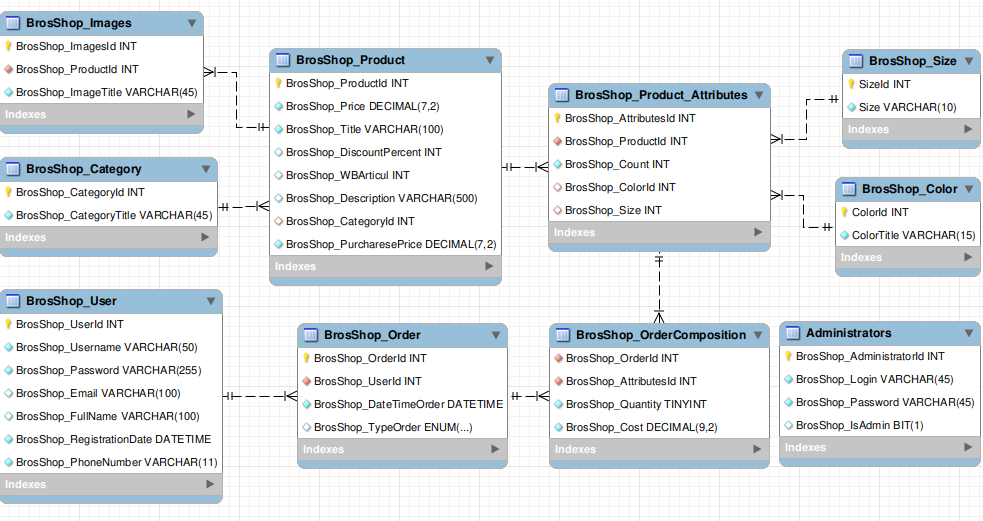


Рисунок 3 – MySQL Workbench. Физическая модель БД

**3 Разработка и интеграция модулей программного обеспечения**

**3.1 Разработка программных модулей**

Для курсового проекта разработано приложение на C# с использованием фреймворка WPF в Visual Studio 22[2].

Для реализации получения изображений с сервера использован класс HttpClient. Код метода загрузки изображения для представлен листингом 1.

Листинг 1 – Код метода для отправки GET-запроса на сервер

|  |
| --- |
| /// <summary>  /// Загружает изображение по его идентификатору.  /// </summary>  /// <param name="imageId">Идентификатор изображения.</param>  ///<returns>BitmapImage загруженного изображения.</returns> public async Task<BitmapImage> LoadImageAsync(int imageId)  {  var apiString = \_configuration["ApiSettings:BaseUrl"];  var response = await  \_httpClient.GetAsync($"{apiString}{imageId}");  if (response.IsSuccessStatusCode)//проверяем  {  var imageBytes = await response.Content.ReadAsByteArrayAsync();  var bitmapImage = new BitmapImage();  using (var stream = new MemoryStream(imageBytes))  {  bitmapImage.BeginInit();  bitmapImage.StreamSource = stream;  bitmapImage.CacheOption = BitmapCacheOption.OnLoad;  bitmapImage.EndInit();  bitmapImage.Freeze();// Замораживаем изображение  //для использования в разных потоках  }  return bitmapImage;// Возвращаем изображение  }  return null; // Возвращаем null в случае неудачи  } |

Код функции API[3] для возвращения изображений представлен листингом 2.

Листинг 2 – Код метода для отправки изображения в API

|  |
| --- |
| /// <summary>  /// Возвращает изображение по его идентификатору.  /// </summary>  /// <param name="id">Идентификатор изображения.</param>  ///<returns>File, представляет файл изображения</returns> [HttpGet("{id}")]  public async Task<ActionResult<BrosShopImage>>  GetBrosShopImage(int id)  {  var brosShopImage = await  \_context.BrosShopImages.FindAsync(id);  //Получаем информацию об изображении  if (brosShopImage == null)//Если нету записи в БД  {  return NotFound($"Image with ID {id} not found.");  //Возвращаем “Не найдено по ID”  }    var imagePath = Path.Combine(\_fileDirectory,  brosShopImage.BrosShopImageTitle);  //Формируем путь к файлу    if (!System.IO.File.Exists(imagePath))  //Если файла не существует  {  return NotFound($"File not found at path:  {imagePath};//Возвращаем Ничего не найдено  }    var contentType = "application/octet-stream";  using (var file = new FileStream(  imagePath,FileMode.Open,  FileAccess.Read, FileShare.Read, 0, true))  {  return File(file, contentType,  brosShopImage.BrosShopImageTitle);  //Возвращаем файл  }  } |

**3.2 Реализация интерфейса пользователя**

Интерфейс разработан с использованием постраничной навигации, что позволяет пользователю легко перемещатся меду различными разделами приложения. В приложении разработаны различные элементы управления, стили, которые упрощают работу и делают интерфейс более интуитивно понятным. Навигация в приложении реализована с помощью элемента Frame, который управляет элементами Page, представляющих страницы в приложении.

После успешной авторизации пользователь попадает на страницу с товарами. В верхней навигационной панели расположенны ссылки, позволяющие легко перейти к страницам заказов, категорий и клиентов. Кроме того, пользователь может воспользоваться функцией двойного нажатия или нажатием соответсвующих кнопок может перейти к окнам создания и редактирования. Вид страниц с товарами и заказами представлен на рисунках 4 и 5.

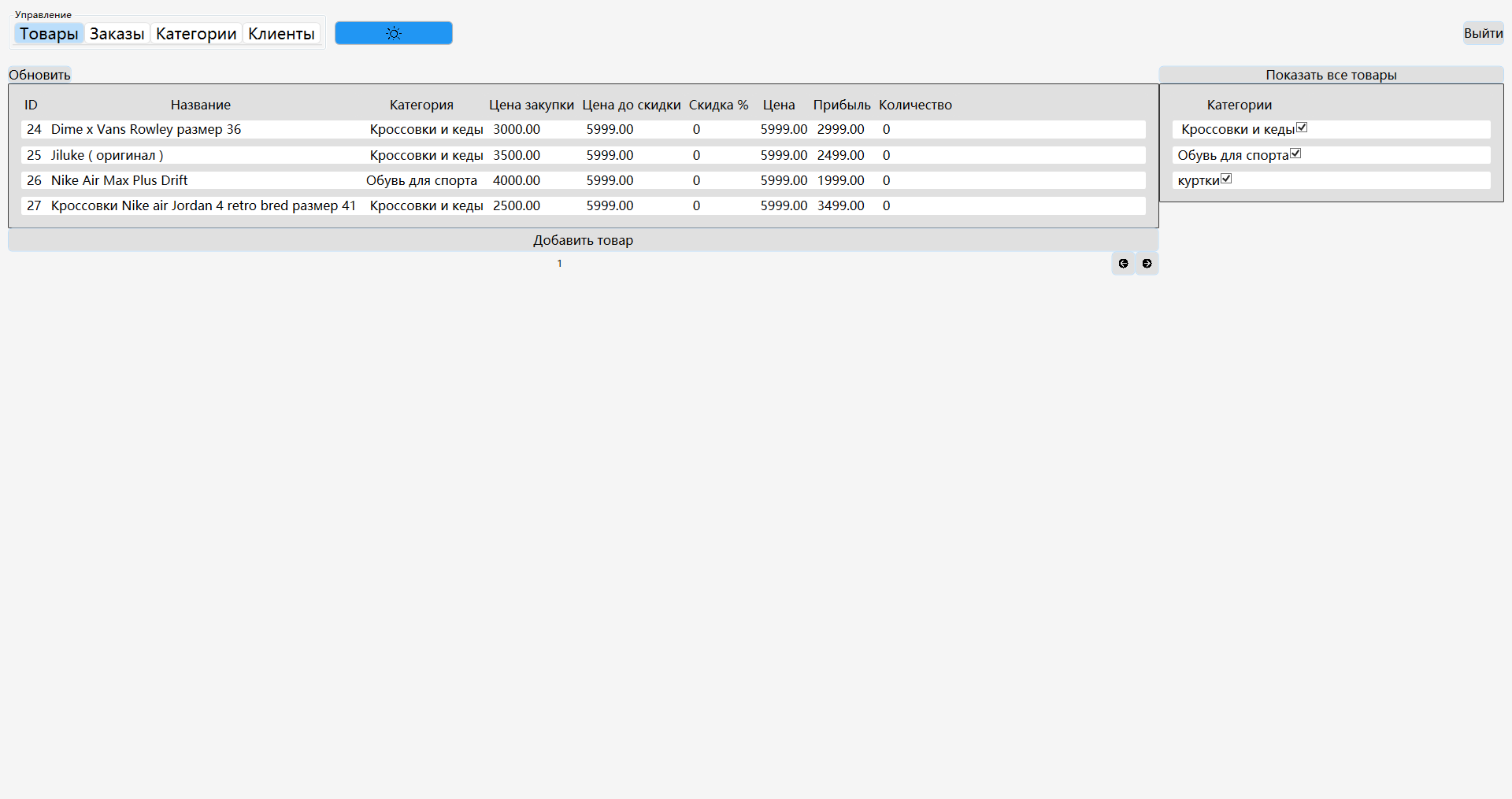


Рисунок 4 – BrosShop. Вид страницы «Товары»

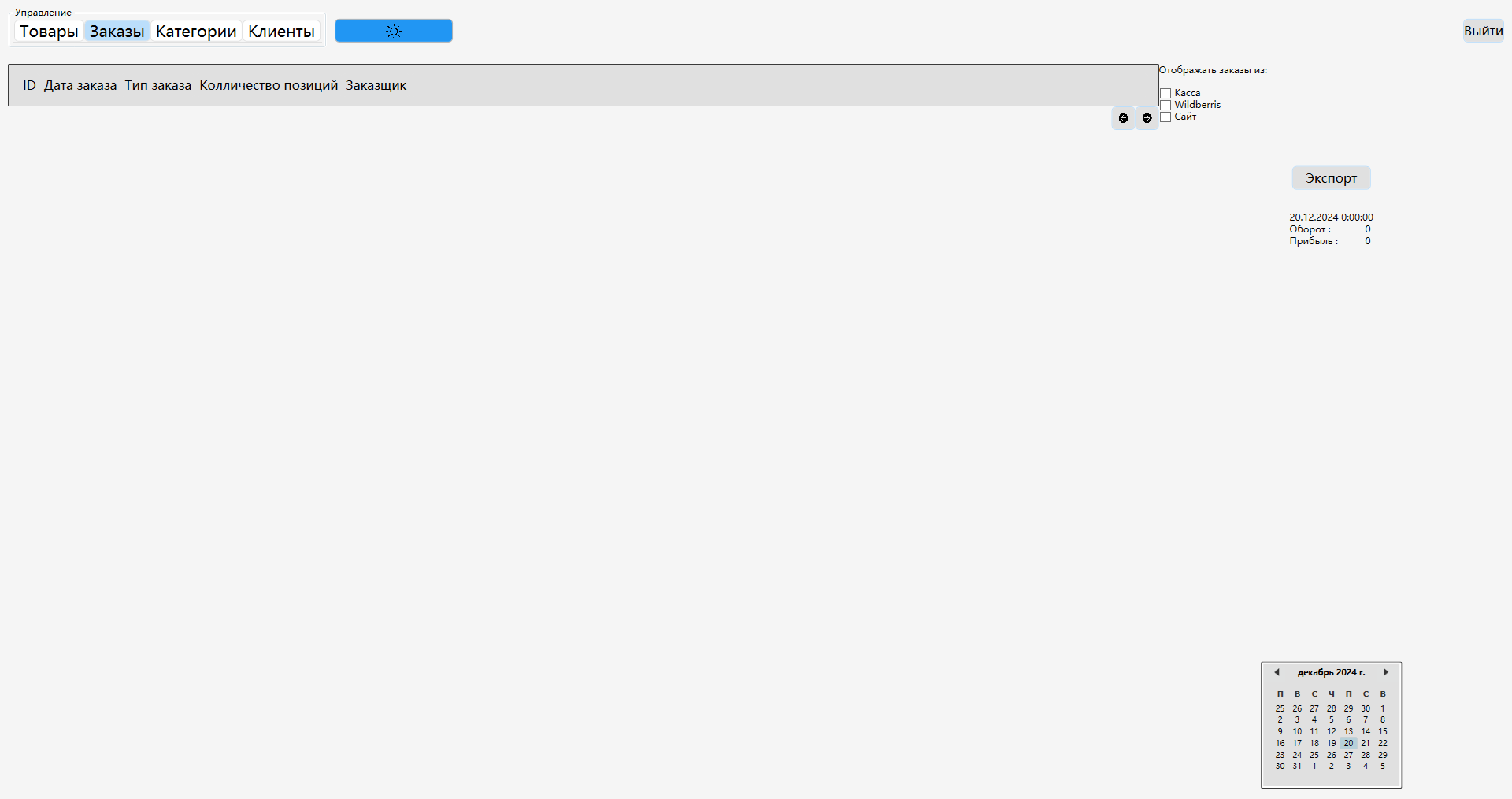


Рисунок 5 – BrosShop. Вид страницы «Заказы»

Для отображения информации о товарах разработана страница ShowProductsPage.xaml, код которой представлен листингом 3.

Листинг 3 – Код страницы ShowProductsPage

|  |
| --- |
| <Page x:Class="BrosShop.ShowProductsPage" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/  presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:local="clr-namespace:BrosShop" mc:Ignorable="d"  Title="Каталог товаров"><!--Название страницы-->  <Grid><!--Основной контейнер-->  <Grid.ColumnDefinitions><!--Определение столбцов-->  <ColumnDefinition/><!--Столбец 1-->  <ColumnDefinition Width="0.3\*"/><!--Столбец 2-->  </Grid.ColumnDefinitions>  <StackPanel><!--Контейнер-->  <Grid VerticalAlignment="Bottom"<!--выравнивание  вертикальное-->  HorizontalAlignment="Left"><!--выравнивание  горизонтальное-->  <Grid.ColumnDefinitions>  <ColumnDefinition Width="Auto"/>  <ColumnDefinition Width="\*"/>  <ColumnDefinition Width="Auto"/>  </Grid.ColumnDefinitions>  <Button x:Name="refreshButton" Grid.Column="3" Content="Обновить" Click="RefreshButton\_Click"/>  </Grid>  <ListView x:Name="productsListView" MouseDoubleClick="ProductsListView\_MouseDoubleClick">  <ListView.View>  <GridView>  <GridViewColumn Header="ID" DisplayMemberBinding="{Binding BrosShopProductId}"/>  <!--Столбец отображения ID-->  <GridViewColumn Header="Название" DisplayMemberBinding="{Binding BrosShopTitle}"/>  <!--Столбец отображения Название-->  <GridViewColumn Header="Категория" DisplayMemberBinding="{Binding BrosShopCategoryTitle}"/>  <!--Столбец отображения Название категории-->  <GridViewColumn Header="Цена закупки" DisplayMemberBinding="{Binding BrosShopPurcharesePrice}" />  <!--Столбец отображения Цену закупки-->  <GridViewColumn Header="Цена до скидки" DisplayMemberBinding="{Binding BrosShopPrice}"/>  <!--Столбец отображения Цены до скидки-->  <GridViewColumn Header="Скидка %" DisplayMemberBinding="{Binding BrosShopDiscountPercent}"/>  <!--Столбец отображения Скидки в %-->  <GridViewColumn Header="Цена " DisplayMemberBinding="{Binding BrosShopDiscountPrice}"/>  <!--Столбец отображения Цены-->  <GridViewColumn Header="Прибыль" DisplayMemberBinding="{Binding BrosShopProfit}" />  <!--Столбец отображения Прибыли-->  <GridViewColumn Header="Количество" DisplayMemberBinding="{Binding BrosShopCount}"/>  <!--Столбец отображения Количества-->  </GridView>  </ListView.View>  </ListView>  <Button x:Name="addProductButton" Height="30"  Content="Добавить товар" Click="AddProductButton\_Click"/>  <Grid>  <Grid.ColumnDefinitions>  <ColumnDefinition Width="0.7\*"/>  <ColumnDefinition Width="Auto"/>  <ColumnDefinition Width="Auto"/>  <ColumnDefinition Width="Auto"/>  </Grid.ColumnDefinitions>  <Button x:Name="previousButton"  <!--Кнопка Назад-->  Click="PreviousButton\_Click" Grid.Column="1"  Width="30" Height="30">  <StackPanel Orientation="Horizontal">  <Image Source="../Icons/arrow.png"  Width="16" Height="16" RenderTransformOrigin="0.5,0.5"/>  </StackPanel>  </Button>  <TextBlock x:Name="currentPageTextBlock"  Grid.Column="0" HorizontalAlignment="Center"  VerticalAlignment="Center"/>  <!--Текущая страница-->  <Button x:Name="nextButton" Grid.Column="2"  <!--Кнопка вперёд-->  Click="NextButton\_Click" Width="30" Height="30">  <StackPanel>  <Image Source="../Icons/arrow.png"  Width="16" Height="16" RenderTransformOrigin="0.5,0.5">  <Image.RenderTransform>  <RotateTransform Angle="-180"/>  </Image.RenderTransform>  </Image>  </StackPanel>  </Button>  </Grid>  </StackPanel>  <StackPanel Grid.Column="1" Grid.Row="1">  <Button x:Name="showAllProductsButton"  <!--Кнопка все товары-->  Content="Показать все товары"  Click="ShowAllProductsButton\_Click"/>  <ListView x:Name="categoryListView">  <!--Контейнер для отображения категорий-->  <ListView.View>  <GridView>  <GridViewColumn Header="Категории">  <GridViewColumn.CellTemplate>  <DataTemplate>  <StackPanel  Orientation="Horizontal">  <TextBlock  Text="{Binding BrosShopCategoryTitle}" />  <!--Название категорий-->  <CheckBox  IsChecked="{Binding BrosShopCategoryIsActive, Mode=TwoWay}"  Checked="CategoryCheckBox\_ChangeChecked"  Unchecked="CategoryCheckBox\_ChangeChecked" Tag="{Binding  BrosShopCategoryId}"/>  <!--Флажок для категорий-->  </StackPanel>  </DataTemplate>  </GridViewColumn.CellTemplate>  </GridViewColumn>  </GridView>  </ListView.View>  </ListView>  </StackPanel>  </Grid><!--Конец контейнера-->  </Page>} <!--Конец страницы--> |

**3.3 Разграничение прав доступа пользователей**

В приложении разработано разграничение прав доступа пользователей с помощью JWT. Для этого в приложении реализована авторизация. Пользователю с ролью администратора доступны все возможности приложения. Неавторизованному пользователю не доступны никакие действия.

Код авторизации с использованием JWT представлен листингом 4.

Листинг 4 – Код авторизации пользователей

|  |
| --- |
| /// <summary>  /// Возвращает токен авторизации.  /// </summary>  /// <param name="username">Имя пользователя</param>  /// <param name="password">Пароль пользователя</param>  ///<returns>AuthToken, токен</returns>  public async Task<AuthToken> AuthenticateAsync(string username, string password)  {  using (var client = new HttpClient())//объявление HttpClient  {  var adminDto = new AdminDto {Login = username, Password = password };//формируем объект передачи  var apiString = \_configuration["ApiSettings:AuthUrl"];  var content = new StringContent(JsonConvert.SerializeObject(adminDto), Encoding.UTF8, "application/json");  var response = await //ожидаем ответ client.PostAsync($"{apiString}login", content);  if(response.IsSuccessStatusCode)//если авторизация успешна  {  var json = await response.Content.ReadAsStringAsync();  return //Возвращаем токен JsonConvert.DeserializeObject<AuthToken>(json);  }  else//иначе  {  throw new Exception("Authentication failed");  //авторизация не успешна  }  }  } |

**3.4 Экспорт данных**

В приложении реализован экспорт данных при помощи библиотеки OfficeOpenXml.

Пример экспорта в файл Excel данных продаж за месяц представлен листингом 5.

Листинг 5 – Код экспорта данных

|  |
| --- |
| private async Task SaveOrdersToExcel()  {  try  {  ExcelPackage.LicenseContext =  LicenseContext.NonCommercial;  // Получаем заказы за текущий месяц  var startDate = new DateTime(DateTime.Now.Year,  DateTime.Now.Month, 1);  var endDate = startDate.AddMonths(1).AddDays(-1);  var orders = \_context.BrosShopOrders  .AsNoTracking()  .Where(o => o.BrosShopDateTimeOrder >= startDate  && o.BrosShopDateTimeOrder <= endDate)  .Include(o => o.BrosShopUser)  .Select(o => o.BrosShopOrderId)  .ToHashSet();  var ordersComposition = await  \_context.BrosShopOrderCompositions  .AsNoTracking()  .Where(oc =>  orders.Contains(oc.BrosShopOrderId)) // Сравниваем с idOrder  .Include(oc => oc.BrosShopOrder)  .Include(oc =>  oc.BrosShopAttributes.BrosShopProduct) // Загружаем продукты  //для каждой составной части  .ToListAsync();  // Создаем Excel файл  using (var package = new ExcelPackage())  {  var worksheet = package.Workbook.Worksheets.Add("Orders");  // Заголовки  worksheet.Cells[1, 1].Value = "Номер  заказа";  worksheet.Cells[1, 2].Value = "Время  заказа";  worksheet.Cells[1, 3].Value = "Общая сумма  заказа";  worksheet.Cells[1, 4].Value = "Имя товара";  worksheet.Cells[1, 5].Value = "Цена";  worksheet.Cells[1, 6].Value = "Количество";  worksheet.Column(2).Width = 20;  worksheet.Column(4).Width = 25;  int row = 2;  foreach (var order in ordersComposition)  {  // Сумма для текущего заказа  var orderCompositions =  ordersComposition.Where(oc => oc.BrosShopOrderId ==  order.BrosShopOrderId).ToList();  var orderSum = orderCompositions.Sum(c  => c.BrosShopCost \* c.BrosShopQuantity);  // Записываем данные о заказе  worksheet.Cells[row, 1].Value =  order.BrosShopOrderId;  worksheet.Cells[row, 2].Value =  order.BrosShopOrder.BrosShopDateTimeOrder;  worksheet.Cells[row,  2].Style.Numberformat.Format = "yyyy-mm-dd hh:mm:ss";  worksheet.Cells[row, 3].Value =  orderSum;  row++;  // Добавляем составные части заказа  foreach (var composition in  orderCompositions)  {  worksheet.Cells[row, 1].Value = "";  // Идентефикатор заказа  worksheet.Cells[row, 2].Value = "";  // Дата и время заказа  worksheet.Cells[row, 3].Value = "";  // Сумма  worksheet.Cells[row, 4].Value =  composition.BrosShopAttributes.BrosShopProduct.BrosShopTitle;  // Название  worksheet.Cells[row, 5].Value =  composition.BrosShopCost; // Цена на момент продажи  worksheet.Cells[row, 6].Value =  composition.BrosShopQuantity; // Количество товаров  row++;  }  }  // Сохраняем файл  var saveFileDialog = new  Microsoft.Win32.SaveFileDialog  {  Filter = "Excel Files|\*.xlsx",  Title = "Сохранить заказы"  };  if (saveFileDialog.ShowDialog() == true)  {  var file = new  FileInfo(saveFileDialog.FileName);  package.SaveAs(file);  MessageBox.Show("Заказы успешно сохранены!");  }  }  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка при сохранении:{ex.Message}");  }  } |

1. **Тестирование и отладка программного обеспечения**

**4.1 Структурное тестирование**

Во время курсового проектирования проведенно структурное тестирование[1] для функции UploadBrosShopImage в API. Для него использованны библиотеки Moq, Xunit. Код unit-теста для загрузки изображения представлен листингом 6.

На рисунке 6 изображена консоль, где отображается информация о результатах модульного тестирования.

Листинг 6 – Код unit-теста для UploadBrosShopImage

|  |
| --- |
| using System.IO;  using System.Threading.Tasks;  using Microsoft.AspNetCore.Http;  using Microsoft.AspNetCore.Mvc;  using Microsoft.EntityFrameworkCore;  using Xunit;  using courseProjAPI.Controllers;  using courseProjAPI.Models;  using Moq;  //Подключение необходимых библиотек и простарств имён  namespace APITest.Tests;  public class UnitTest1 // Определение класса для юнит-тестов  {  private readonly ImagesController \_controller;  // Контроллер, который будет тестироваться  private readonly BrosShopDbContext \_context;  // Контекст базы данных для тестирования  public UnitTest1() // Конструктор класса  {  // Настройка in-memory базы данных для тестирования  var options = new  DbContextOptionsBuilder<BrosShopDbContext>()  .UseInMemoryDatabase(databaseName: "TestDatabase")  .Options;  \_context = new BrosShopDbContext(options);  // Инициализация контекста базы данных  \_controller = new ImagesController(\_context);  // Инициализация контроллера с контекстом базы данных  }  [Fact] // Атрибут, указывающий, что это тестовый метод  public async Task  UploadBrosShopImage\_ReturnsBadRequest\_WhenFileIsNull()  {  //Подготовка данных для теста  int productId = 1; // ID продукта  IFormFile file = null; // Файл, который будет передан в  //метод, равен null  // Вызов тестируемого метода  var result = await  \_controller.UploadBrosShopImage(productId, file);  // Проверка результата  var badRequestResult =  Assert.IsType<BadRequestObjectResult>  (result.Result); // Проверка, что результат -  //BadRequest  Assert.Equal("Файл не загружен.",  badRequestResult.Value);  // Проверка, что сообщение об ошибке  //соответствует ожидаемому  }  [Fact] // Атрибут, указывающий, что это тестовый метод  public async Task  UploadBrosShopImage\_ReturnsNotFound\_  WhenProductDoesNotExist()  {  // Подготовка данных для теста  int productId = 1; // ID продукта  var fileMock = new Mock<IFormFile>();  // Создание мок-объекта для IFormFile  fileMock.Setup(f => f.FileName).Returns("test.jpg");  // Настройка имени файла  fileMock.Setup(f => f.Length).Returns(1);  // Настройка длины файла  // Вызов тестируемого метода  var result = await  \_controller.UploadBrosShopImage(productId,  fileMock.Object);  // Проверка результата  var notFoundResult =  Assert.IsType<NotFoundObjectResult>(result.Result);  // Проверка, что результат - NotFound  Assert.Equal($"Product with ID {productId} not found.",  notFoundResult.Value);  // Проверка, что сообщение об ошибке  //соответствует ожидаемому  }  } |

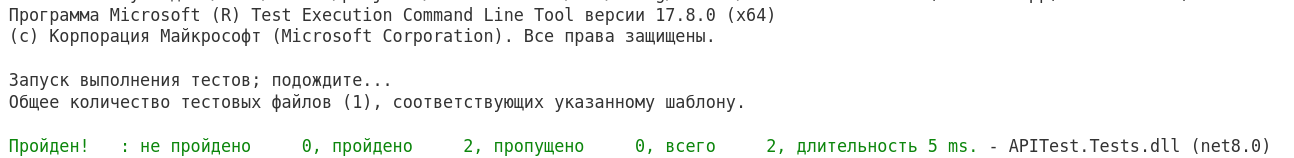


Рисунок 6 – Visual Studio Code. Результаты unit-тестирования

**4.2 Функциональное тестирование**

Во время курсового проектирования проведено функциональное тестирование приложения методом «черного ящика», результаты тестирования представлены в таблице 1.

Таблица 1 – Набор тестов для приложения

|  |  |  |
| --- | --- | --- |
| Действие | Ожидаемый результат | Фактический результат |
| Авторизоватся с пустыми полями | Визуальное отображение неверно заполненых полей | Совпадает с ожидаемым |
| Авторизоватся только с 1 пустым полем | Визуальное отображение неверно заполненых полей | Совпадает с ожидаемым |
| Нажать на кнопку смены темы и затем открыть другое окно | В новом окне будет сменённая тема | Совпадает с ожидаемым |
| Нажать на флажок 1 из категорий | Будут отображатся только товары с активными флажками | Совпадает с ожидаемым |
| На странице «Заказы» сменить день в календаре | В сводке продаж за день изменятся данные | Совпадает с ожидаемым |

По результатам тестирования можно сделать вывод, что разработанное программное обеспечение работает корректно и согласно ожиданиям.

1. **Инструкция по эксплуатации программного обеспечения**
   1. **Установка программного обеспечения**

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

* ОС – Ubuntu 22.04,
* процессор – 2 ядра по 2 ГГц,
* оперативная память – 2 ГБ,
* свободное место на диске – 10 ГБ,
* необходимые компоненты: MySQL 8.0.40, Docker, Nginx.

Для развёртывания БД нужно подключиться к серверу MySQL, с помощью MySQL Workbench, вставить и запустить скрипт из репозитория.

Для установки серверной части требуется перейти в терминале в желаемою папку для API, склонировать репозиторий, создать файл appsettings.json по примеру файла appsettings.Development.json и в терминале использовать команду представленную листингом 6

Листинг 6 – Код запуска API средствами docker-compose

|  |
| --- |
| docker-compose up -d --build |

Для функционирования оконного приложения достаточны следующие минимальные программные и технические средства:

* ОС – Windows 10 и выше,
* процессор – 1 ГГц,
* оперативная память – 4 ГБ,
* свободное место в хранилище – 40 МБ,
* дополнительное - постоянное интернет-подключение.

Для установки оконного приложения требуется распаковать BrosShop.zip и заменить appsettings.json с учётом нового сервера.

В приложении используются слудующие учётные данные:  
 – логин: Denis ,

– пароль: Denis .

* 1. **Инструкция по работе**

При запуске приложения, администратора встречает окно авторизации. Для авторизации требуется ввести учётные данные в поля ввода логина и пароля и нажать на кнопку «Авторизоваться». Окно авторизации показано на рисунке 6.

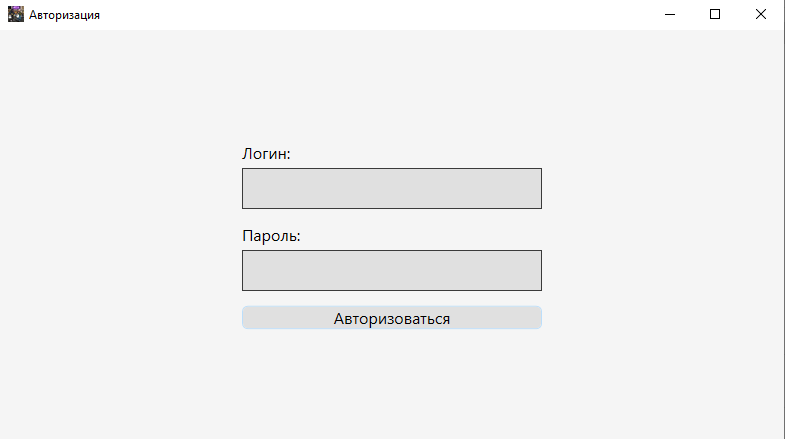


Рисунок 6 – BrosShop. Вид окна «Авторизация»

После авторизации пользователь перенаправляется на главное окно, где ему доступны страницы «Товары», «Заказы», «Категории» и «Клиенты». По умолчанию в главном окне открыта страница «Товары», в которой он может просмотреть информацию о существующих и добавить новые товары. Окно с подробной информацией о товаре показанно на рисунке 7.

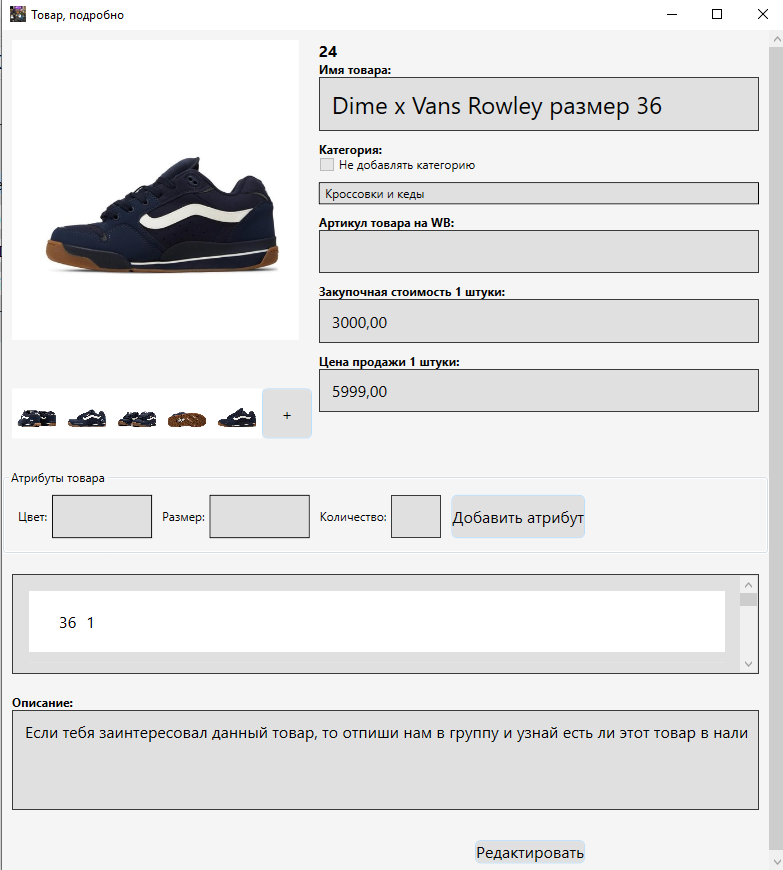


Рисунок 7 – BrosShop. Вид окна «Товар, подробно»

Для того, что бы посмотреть статистику продаж, нужно перейти на страницу «Заказы», по умолчанию будет стоять сегодняшняя дата, пример отображенния данных показан на рисунке 8.



Рисунок 8 – BrosShop. Модуль окна «Заказы» статистика продаж

Для просмотра информации о заказе нужно дважды кликнуть на интересующий вас заказ, тогда вам отобразится окно с подробной информацией о заказе, пример страницы «Заказы» и отображение состава заказа показаны на рисунке 9 и 10 соответсвенно.

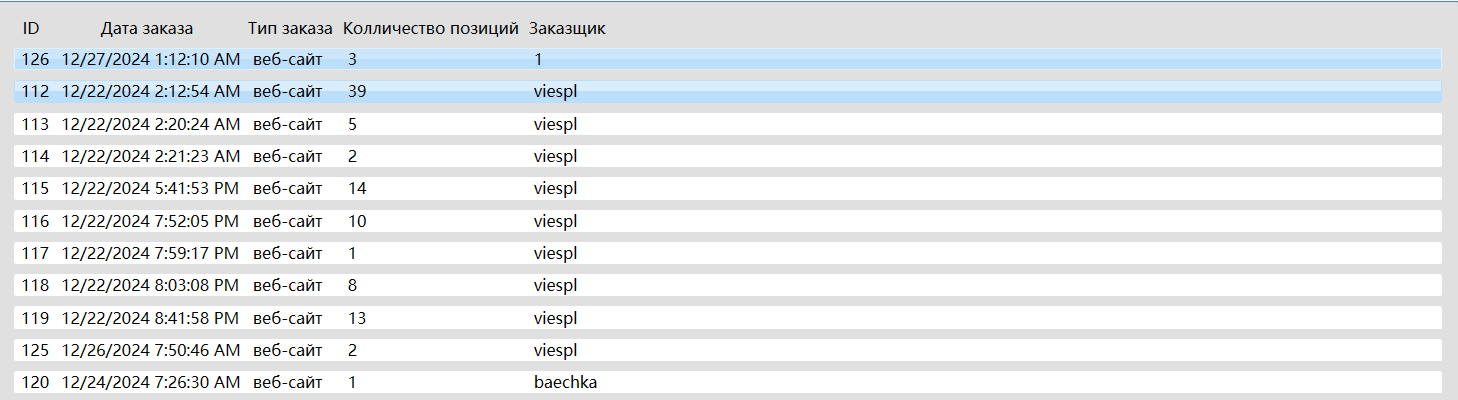


Рисунок 9 – BrosShop. Модуль окна с таблицей заказов

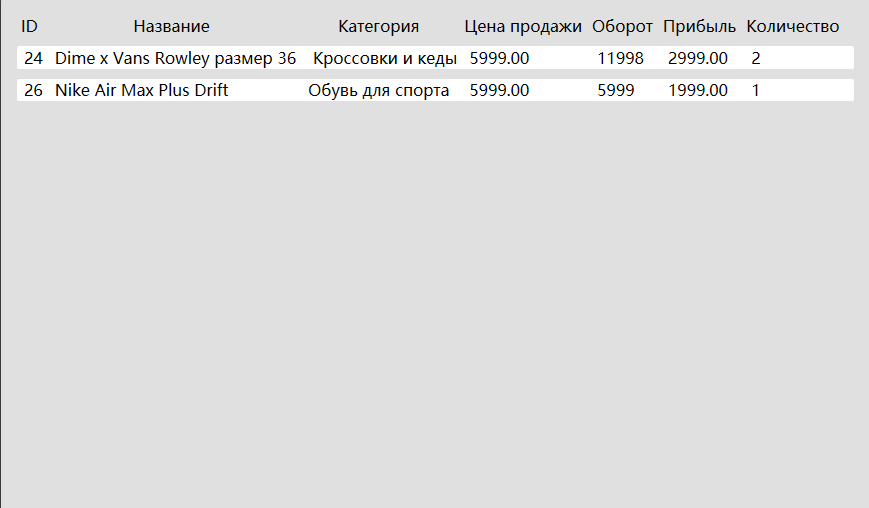


Рисунок 10 – BrosShop. Окно «Состав заказа»

**ЗАКЛЮЧЕНИЕ**

В ходе курсового проектирования достигнута поставленная цель: разработана подсистема «Товарный монитор», которая поможет обеспечить управление товарами и заказами. Кроме того, решены все поставленные задачи:

* проведён сбор требований целевой аудитории,
* проанализированы информационные источники по предметной области,
* спроектирована архитектура приложения,
* спроектирована диаграмма вариантов использования приложения,
* выбран состав программных и технических средств для реализации приложения,
* спроектирована БД,
* спроектирован интерфейс оконного приложения,
* создана БД в выбранной СУБД,
* разработано API для некоторых функций приложения,
* реализовано разграничение прав доступа пользователей,
* разработан интерфейс оконного приложения,
* разработано оконное приложение,
* реализован экспорт данных в виде файлов Excel,
* реализована работа приложения с сервером БД при помощи REST API,
* выполнено структурное тестирование ПО,
* выполнено функциональное тестирование ПО,
* разработана программная и эксплуатационная документация.

В результате выполнения поставленных задач разработано оконное приложение обеспечивающее эффективное управление товарами и заказами, отвечающее современным тенденциям и требованиям заказчика.

**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: 05.11.2024). – 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, 2023. – 400 с. – URL: https://znanium.com/catalog/product/1895679 (дата обращения: 06.11.2024). – Режим доступа: по подписке. – Текст : электронный.
3. Гивакс, Д. Д. Паттерны проектирования API. – Санкт-Петербург : Питер, 2023. – 512 с. – URL: https://ibooks.ru/bookshelf/390212/reading (дата обращения: 07.11.2024). – Режим доступа: для зарегистрир. пользователей. – Текст: электронный.
4. Дадян, Э. Г. Данные: хранение и обработка : учебник. – Москва : ИНФРА-М, 2020. – 205 с. – URL: https://znanium.com/catalog/product/1045133 (дата обращения: 11.11.2024). – Режим доступа: по подписке. – Текст : электронный.
5. 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, 2023. — 368 с. - URL: https://znanium.com/catalog/product/1912454 (дата обращения: 13.11.2024). – Режим доступа: по подписке. — Текст : электронный.