**Государственное бюджетное профессиональное образовательное учреждение**

**«Нижегородский Губернский колледж»**

Методическая комиссия «Информатика и вычислительная техника»

Допущена к защите:

Преподаватели

\_\_\_\_\_\_\_\_\_\_\_\_ Л.В.Мухина,

\_\_\_\_\_\_\_\_\_\_\_\_ Е.П. Голубева,

«21» марта 2025 г.

**ОТЧЕТ** **ПО**

**УЧЕБНОЙ ПРАКТИКЕ**

ПМ.01 РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ

Руководитель: \_\_\_\_\_\_\_\_\_\_ Мухина Л.В., Голубева Е.П. 21.03.2025 г.

Студентка: \_\_\_\_\_\_\_\_\_\_\_\_\_ Любивая А.А. 21.03.2025 г.

Специальность, группа: 09.02.07, 43П.

Нижний Новгород

2025 г.

1. Оглавление

[Введение 2](#_Toc192889378)

[1. Создание базы данных 4](#_Toc192889379)

[2. Разработка информационной системы 14](#_Toc192889380)

[2.1. Создание проекта 14](#_Toc192889381)

[2.2. Создание страницы авторизации 15](#_Toc192889382)

[2.3. Создание страницы вывода продуктов 20](#_Toc192889383)

[2.4. Создание страницы Корзина 26](#_Toc192889384)

[3. Разработка и тестирование dll библиотеки 30](#_Toc192889385)

[3.1. Разработка библиотеки 30](#_Toc192889386)

[3.2. Написание Юнит тестов 35](#_Toc192889387)

[Заключение 43](#_Toc192889388)

[Приложения 44](#_Toc192889389)

[Приложение А 44](#_Toc192889390)

[Приложение Б 46](#_Toc192889391)

Введение

Учебная практика по направлению ПМ.01 проходила в ГБПОУ «Нижегородский Губернский колледж» с 1 по 21 марта 2025 года. Основной целью практики было освоение как общих, так и профессиональных компетенций, среди которых:

ПК 1.1. Формирование алгоритмов разработки программных модулей в соответствии с техническим заданием.

ПК 1.2. Разработка программных модулей в соответствии с техническим заданием.

ПК 1.3. Выполнение отладки программных модулей с использованием специализированных программных средств.

ПК 1.4. Тестирование программных модулей.

ПК 1.5. Рефакторинг и оптимизация программного кода.

ПК 1.6. Разработка модулей программного обеспечения для мобильных платформ.

Для достижения поставленной цели были определены следующие задачи:

1. Изучение технического задания и формирование алгоритма разработки.
2. Создание базы данных в соответствии с техническим заданием.
3. Разработка пользовательского интерфейса.
4. Реализация функционала приложения.
5. Проведение отладки и тестирования готового приложения.
6. Рефакторинг и оптимизация кода.

Практическим результатом данной учебной практики стал отчет, в котором представлены выполненные задания, а также графические материалы, такие как рисунки, таблицы и диаграммы.

1. Создание базы данных

Таблица Kategory

Назначение: Эта таблица предназначена для хранения категорий продуктов. Каждая категория может включать в себя несколько продуктов.

Структура:

id SERIAL PRIMARY KEY: Уникальный идентификатор категории, автоматически увеличивается при добавлении новой записи.

name VARCHAR(100) NOT NULL: Название категории, не может быть пустым.

Связи:

Имеет связь с таблицей Product через поле kategory, что позволяет каждому продукту принадлежать к определенной категории.

Таблица Orders

Назначение: Таблица предназначена для хранения информации о заказах. Каждый заказ может содержать информацию о дате заказа, дате доставки, месте получения и статусе заказа.

Структура:

id SERIAL PRIMARY KEY: Уникальный идентификатор заказа.

date\_order DATE NOT NULL: Дата, когда заказ был сделан, не может быть пустым.

date\_delivery DATE NOT NULL: Дата, когда заказ должен быть доставлен, не может быть пустым.

pick\_up INT NOT NULL: Идентификатор пункта выдачи, где заказ может быть забран, не может быть пустым.

FIO VARCHAR(200): ФИО клиента, может быть пустым.

code INT NOT NULL: Код заказа, не может быть пустым.

status INT NOT NULL: Статус заказа, не может быть пустым.

Связи:

FOREIGN KEY (pick\_up) REFERENCES Pick\_up\_point(id): Связь с таблицей Pick\_up\_point, указывающая на пункт выдачи.

FOREIGN KEY (status) REFERENCES OrderStatus(StatusID): Связь с таблицей OrderStatus, указывающая на статус заказа.

Таблица Product

Назначение: Эта таблица предназначена для хранения информации о продуктах, которые могут быть заказаны. Она включает в себя детали о каждом продукте, такие как название, цена и количество на складе.

Структура:

id SERIAL PRIMARY KEY: Уникальный идентификатор продукта.

article\_number VARCHAR(100) NOT NULL: Артикул продукта, не может быть пустым.

name VARCHAR(255) NOT NULL: Название продукта, не может быть пустым.

unit VARCHAR(50) NOT NULL: Единица измерения продукта (например, "кг", "шт."), не может быть пустым.

cost DECIMAL(19, 4) NOT NULL: Цена продукта, не может быть пустой.

max\_discount\_amount INT: Максимальная сумма скидки для продукта, может быть пустой.

production INT NOT NULL: Идентификатор производства, где был произведен продукт, не может быть пустым.

provider INT NOT NULL: Идентификатор поставщика продукта, не может быть пустым.

kategory INT NOT NULL: Идентификатор категории продукта, не может быть пустым.

current\_discount DECIMAL(19, 4): Текущая скидка на продукт, может быть пустой.

quantity\_in\_stock INT NOT NULL: Количество продукта на складе, не может быть пустым.

description TEXT: Описание продукта, может быть пустым.

image VARCHAR(100): Путь к изображению продукта, может быть пустым.

Связи:

FOREIGN KEY (provider) REFERENCES Provider(id): Связь с таблицей Provider, указывающая на поставщика продукта.

FOREIGN KEY (kategory) REFERENCES Kategory(id): Связь с таблицей Kategory, указывающая на категорию продукта.

FOREIGN KEY (production) REFERENCES Production(id): Связь с таблицей Production, указывающая на производство продукта.

Таблица Production

Назначение: Эта таблица предназначена для хранения информации о производителях продуктов. Каждый производитель может производить несколько продуктов.

Структура:

id SERIAL PRIMARY KEY: Уникальный идентификатор производителя, автоматически увеличивается при добавлении новой записи.

name VARCHAR(100) NOT NULL: Название производителя, не может быть пустым.

Связи:

Имеет связь с таблицей Product через поле production, что позволяет каждому продукту быть связанным с определенным производителем.

Таблица Provider

Назначение: Таблица предназначена для хранения информации о поставщиках продуктов. Поставщики могут предоставлять разные продукты.

Структура:

id SERIAL PRIMARY KEY: Уникальный идентификатор поставщика, автоматически увеличивается при добавлении новой записи.

name VARCHAR(100) NOT NULL: Название поставщика, не может быть пустым.

Связи:

Имеет связь с таблицей Product через поле provider, что позволяет каждому продукту быть связанным с определенным поставщиком.

Таблица Role

Назначение: Эта таблица предназначена для хранения ролей пользователей в системе. Роли могут определять права доступа и функциональные возможности пользователей.

Структура:

RoleID SERIAL PRIMARY KEY: Уникальный идентификатор роли, автоматически увеличивается при добавлении новой записи.

RoleName VARCHAR(100) NOT NULL: Название роли, не может быть пустым.

Связи:

Имеет связь с таблицей Users через поле User Role, что позволяет каждому пользователю иметь определенную роль.

Таблица Users

Назначение: Таблица предназначена для хранения информации о пользователях системы. Каждый пользователь имеет свои данные и роль.

Структура:

User ID int primary key: Уникальный идентификатор пользователя.

User Surname varchar(100) not null: Фамилия пользователя, не может быть пустой.

User Name varchar(100) not null: Имя пользователя, не может быть пустым.

User Patronymic varchar(100) not null: Отчество пользователя, не может быть пустым.

User Login text not null: Логин пользователя, не может быть пустым.

User Password text not null: Пароль пользователя, не может быть пустым.

User Role int not null: Идентификатор роли пользователя, не может быть пустым.

Связи:

foreign key (User Role) references Role(RoleID): Связь с таблицей Role, указывающая на роль пользователя.

Таблица OrderProduct

Назначение: Эта таблица предназначена для хранения информации о продуктах, входящих в заказы. Она связывает заказы с конкретными продуктами и их количеством.

Структура:

id SERIAL PRIMARY KEY: Уникальный идентификатор записи о продукте в заказе, автоматически увеличивается при добавлении новой записи.

OrderID INT NOT NULL: Идентификатор заказа, не может быть пустым.

ProductArticleNumber VARCHAR(100) NOT NULL: Артикул продукта, не может быть пустым.

Quantity INT NOT NULL: Количество данного продукта в заказе, не может быть пустым.

Связи:

FOREIGN KEY (OrderID) REFERENCES Orders(id) ON DELETE CASCADE: Связь с таблицей Orders, указывающая на заказ. При удалении заказа все связанные записи в этой таблице также будут удалены.

FOREIGN KEY (ProductArticleNumber) REFERENCES Product(article\_number) ON DELETE CASCADE: Связь с таблицей Product, указывающая на продукт. При удалении продукта все связанные записи в этой таблице также будут удалены.

Предметная область, основанная на описанных таблицах, охватывает управление продуктами, пользователями и заказами в контексте системы, связанной с продажами или поставками товаров. Вот краткий обзор ключевых аспектов этой области: (см. Рисунок 1 – 10, Приложение А)

1. Продукты

Система управляет информацией о товарах, включая их производителей и поставщиков. Каждому продукту присваивается уникальный артикул, что позволяет отслеживать его наличие и характеристики.

2. Производители и Поставщики

Производители отвечают за создание товаров, а поставщики обеспечивают их доставку. Система позволяет связывать продукты с конкретными производителями и поставщиками, что упрощает управление цепочками поставок.

3. Пользователи и Роли

Система включает пользователей, которые могут выполнять различные функции в зависимости от назначенных ролей. Роли определяют права доступа и функциональные возможности, что позволяет организовать работу команды и контролировать действия пользователей.

4. Заказы

Система также управляет процессом заказов, позволяя отслеживать, какие продукты были заказаны, в каком количестве и кем. Это важно для управления запасами и удовлетворения потребностей клиентов.

5. Связи между элементами

Связи между таблицами обеспечивают целостность данных. Например, удаление продукта автоматически удаляет все связанные с ним записи в заказах, что помогает поддерживать актуальность информации.

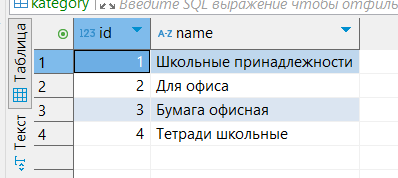


Рисунок №1 – Таблица 1

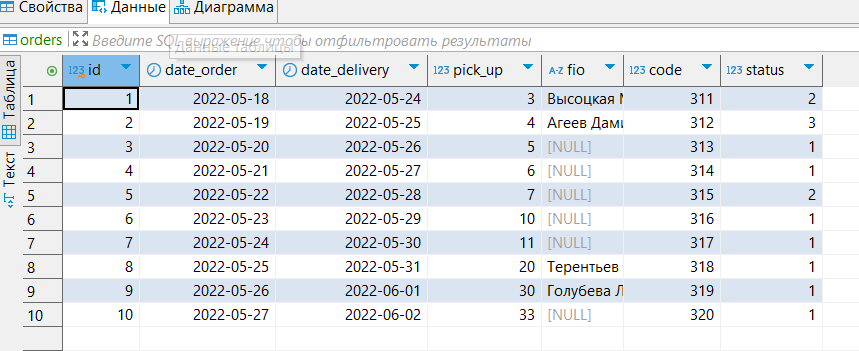


Рисунок №2 – Таблица 2

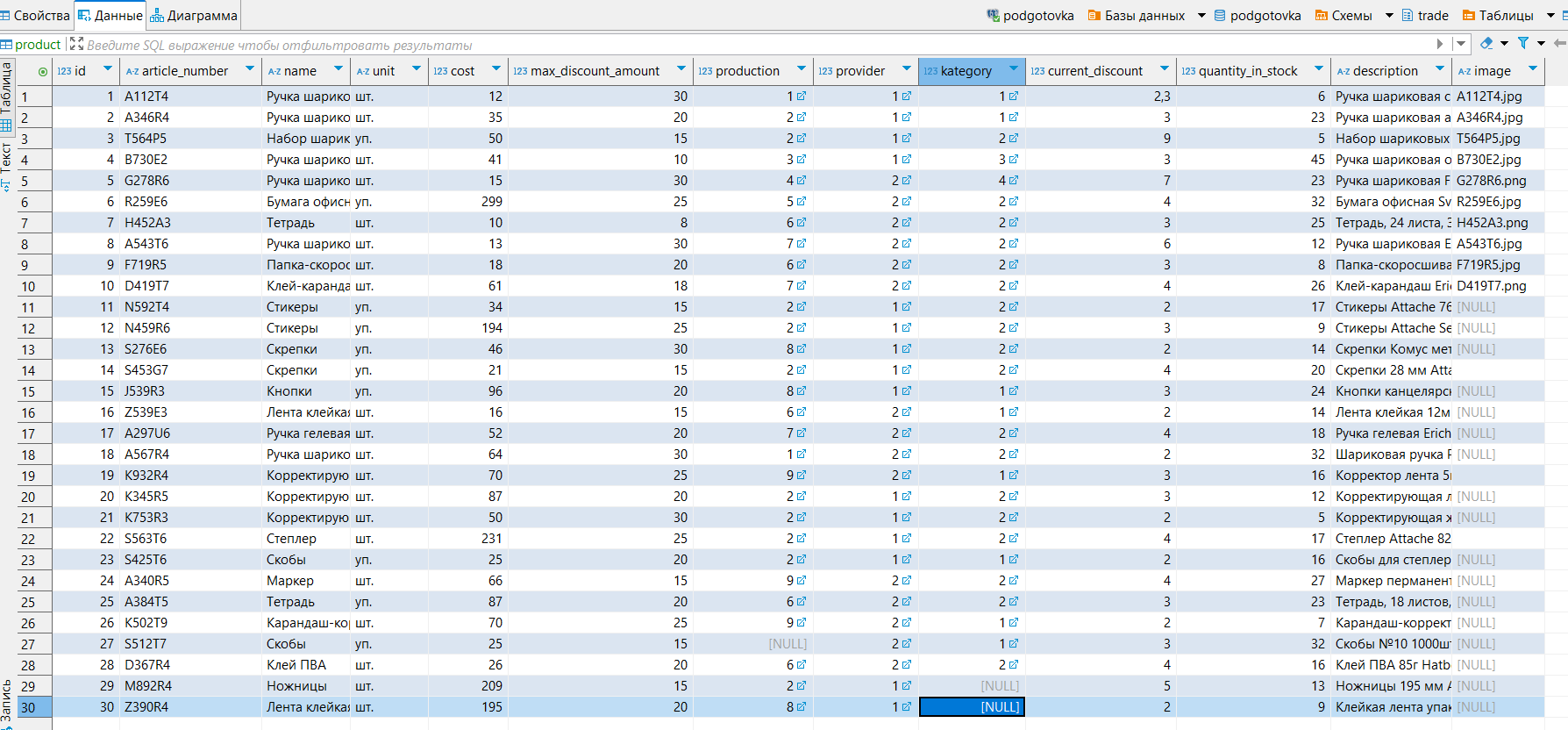


Рисунок №3 – Таблица 3

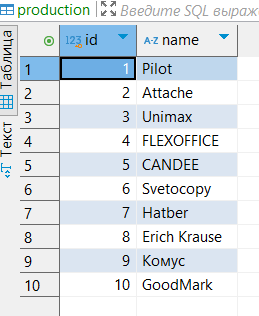


Рисунок №4 – Таблица 4

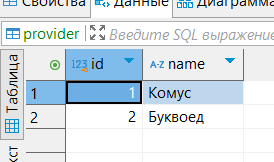


Рисунок №5 – Таблица 5

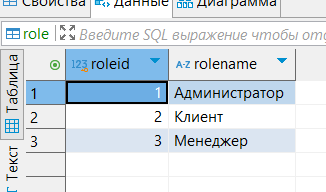


Рисунок №6 – Таблица 6

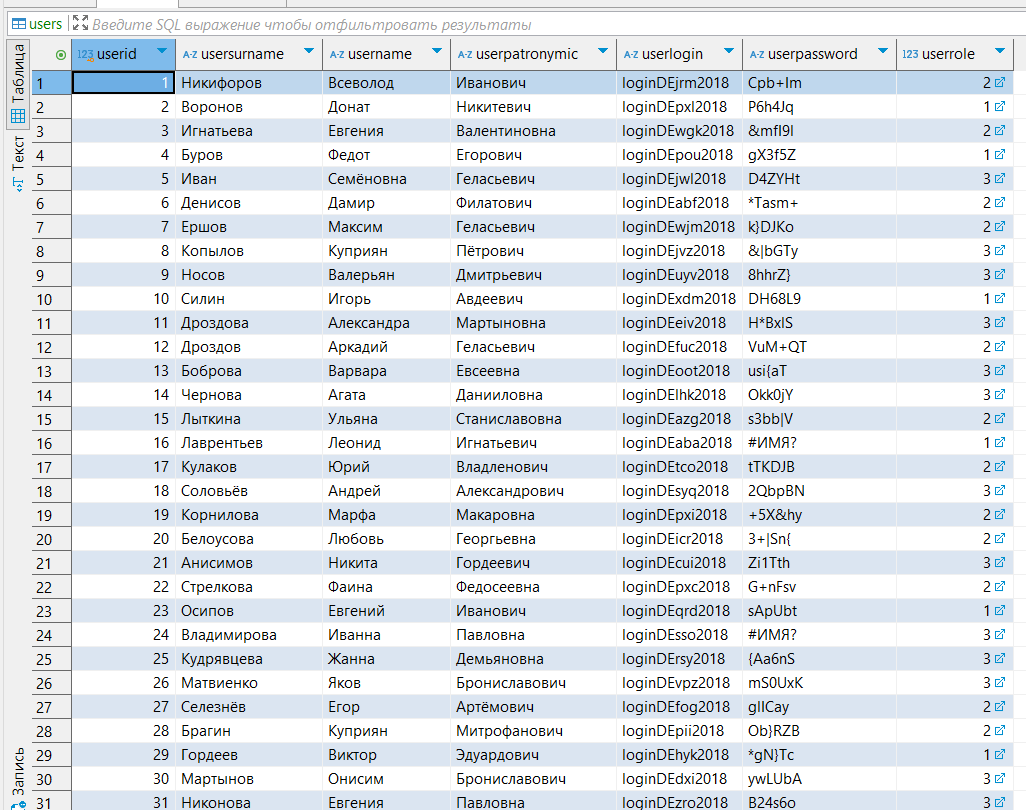


Рисунок №7 – Таблица 7

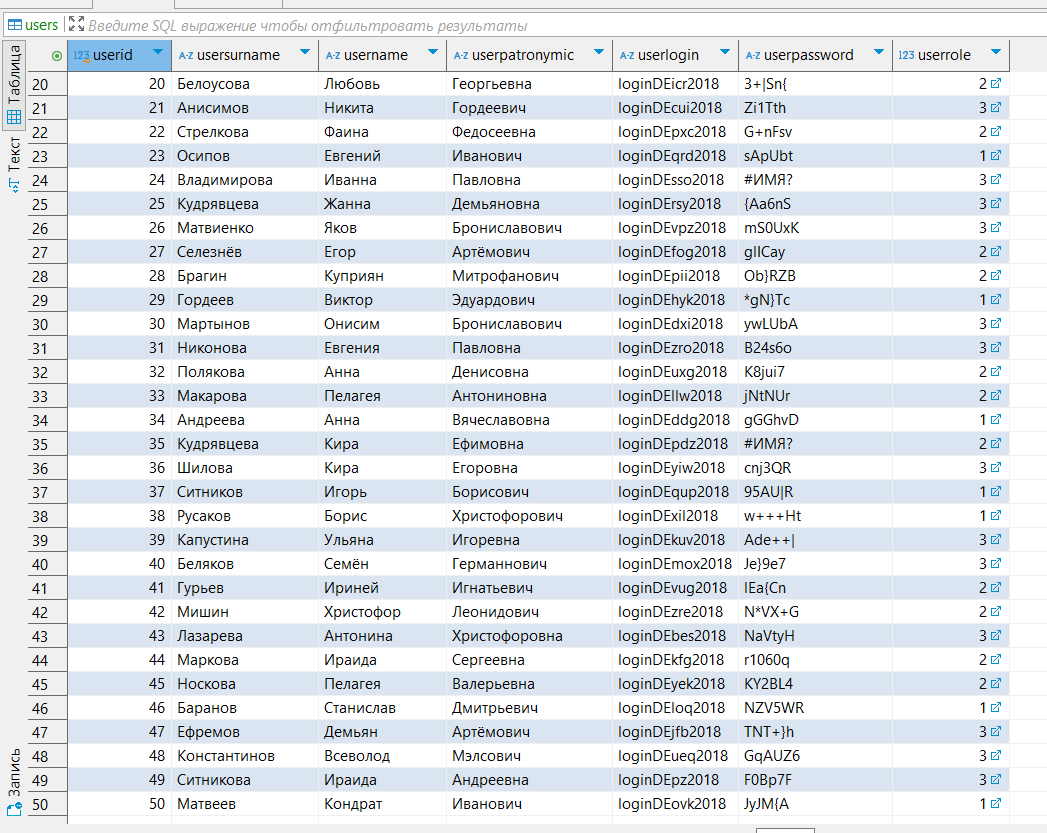


Рисунок №8 – Таблица 8

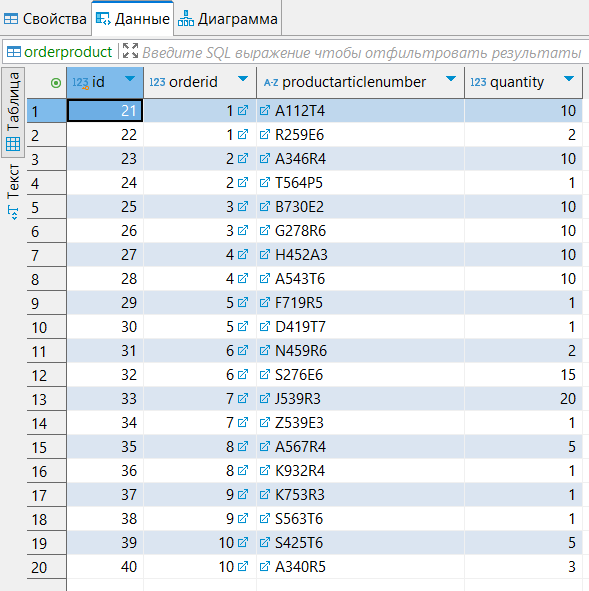


Рисунок №9 – Таблица 9

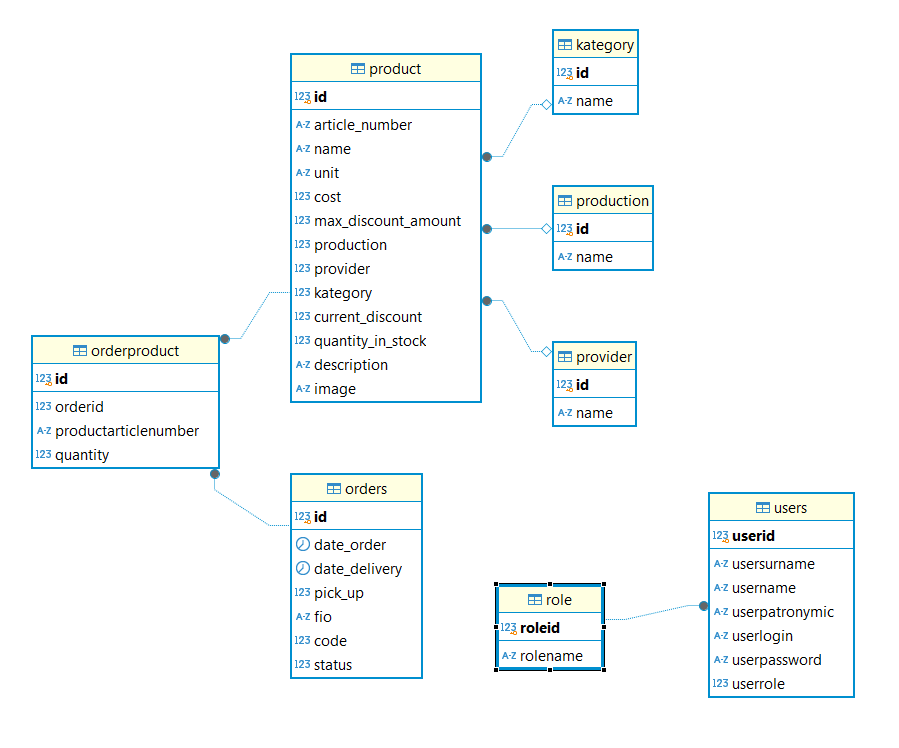


Рисунок №10 – Схема базы данных

1. Разработка информационной системы
   1. Создание проекта

Информационная система разработана на основе фреймворка Avalonia UI, используя архитектурный шаблон MVVM



Рисунок №11 – Название проекта



Рисунок №12 – Иконка

* 1. Создание страницы авторизации

Общая структура

Экран авторизации состоит из двух основных частей:

Заголовок: Отображает текст "Авторизация".

Форма авторизации: Содержит элементы для ввода данных пользователя и кнопки для выполнения действий.

Элементы управления

Заголовок

<TextBlock Grid.Row="0" Text="Авторизация" HorizontalAlignment="Center" FontSize="20" FontWeight="Bold" Margin="10"/>

TextBlock: Отображает заголовок "Авторизация" в центре верхней части экрана. У него задан размер шрифта 20 и жирное начертание.

Основная форма

<ScrollViewer Grid.Row="1" Margin="10" BorderThickness="2" BorderBrush="Green" VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Disabled">

<Border BorderBrush="Gray" BorderThickness="1" Margin="10" Padding="10" Background="RGB(118, 227, 131)">

<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">

...

</StackPanel>

</Border>

</ScrollViewer>

ScrollViewer: Позволяет прокручивать содержимое, если оно превышает размеры видимой области. У него есть зеленая рамка и возможность вертикальной прокрутки.

Border: Обрамляет содержимое формы, имеет серую рамку и зеленый фон. Это создает визуальный акцент на области авторизации.

StackPanel: Упорядочивает дочерние элементы вертикально и центрирует их.

Элементы ввода

<TextBox x:Name="log" Text="{Binding UserName}" TextWrapping="Wrap" Margin="0,5,0,0" Background="White" Width="200"/>

<TextBox x:Name="par" Text="{Binding Password}" TextWrapping="Wrap" Margin="0,5,0,0" Background="White" Width="200"/>

TextBox для логина (log): Пользователь вводит свое имя пользователя. Связь с User Name из ViewModel позволяет обновлять значение в реальном времени.

TextBox для пароля (par): Пользователь вводит свой пароль. Связь с Password из ViewModel. Для повышения безопасности можно настроить его как PasswordBox.

Кнопки

<Button Command="{Binding ExecuteLogin}" IsEnabled="{Binding EnterButtonEnabled}" Background="RGB(73, 140, 81)" Foreground="White" Width="200" HorizontalContentAlignment="Center">Вход</Button>

<Button Command="{Binding ToShowCommand}" Background="RGB(73, 140, 81)" Foreground="White" Width="200" HorizontalContentAlignment="Center">Войти как гость</Button>

Кнопка "Вход": Выполняет команду ExecuteLogin. Она активна только тогда, когда свойство EnterButtonEnabled равно true.

Кнопка "Войти как гость": Выполняет команду ToShowCommand, позволяя пользователю зайти без авторизации.

Капча

<TextBlock Text="{Binding captcha}" IsVisible="{Binding EnterButtonVisible}" FontWeight="Bold" HorizontalAlignment="Center" Padding="10" />

<TextBox Text="{Binding captchaText}" Watermark="Введите капчу" IsVisible="{Binding EnterButtonVisible}" Background="#E6E6FA" BorderBrush="#9370DB" BorderThickness="2"/>

TextBlock: Отображает текст капчи, который генерируется в ViewModel. Видим только если EnterButtonVisible равно true.

TextBox: Поле для ввода капчи. Пользователь должен ввести текст, чтобы пройти авторизацию. Также видимо только при необходимости.

Логика работы

При загрузке экрана авторизации пользователь видит форму для ввода логина и пароля, а также капчу, если это необходимо.

После ввода данных и нажатия на кнопку "Вход" выполняется проверка введенных данных. Если данные верны и капча введена правильно, происходит переход на следующий экран (например, ShowProduct).

Если авторизация не удалась, отображается сообщение об ошибке, и пользователю предлагается ввести капчу заново.

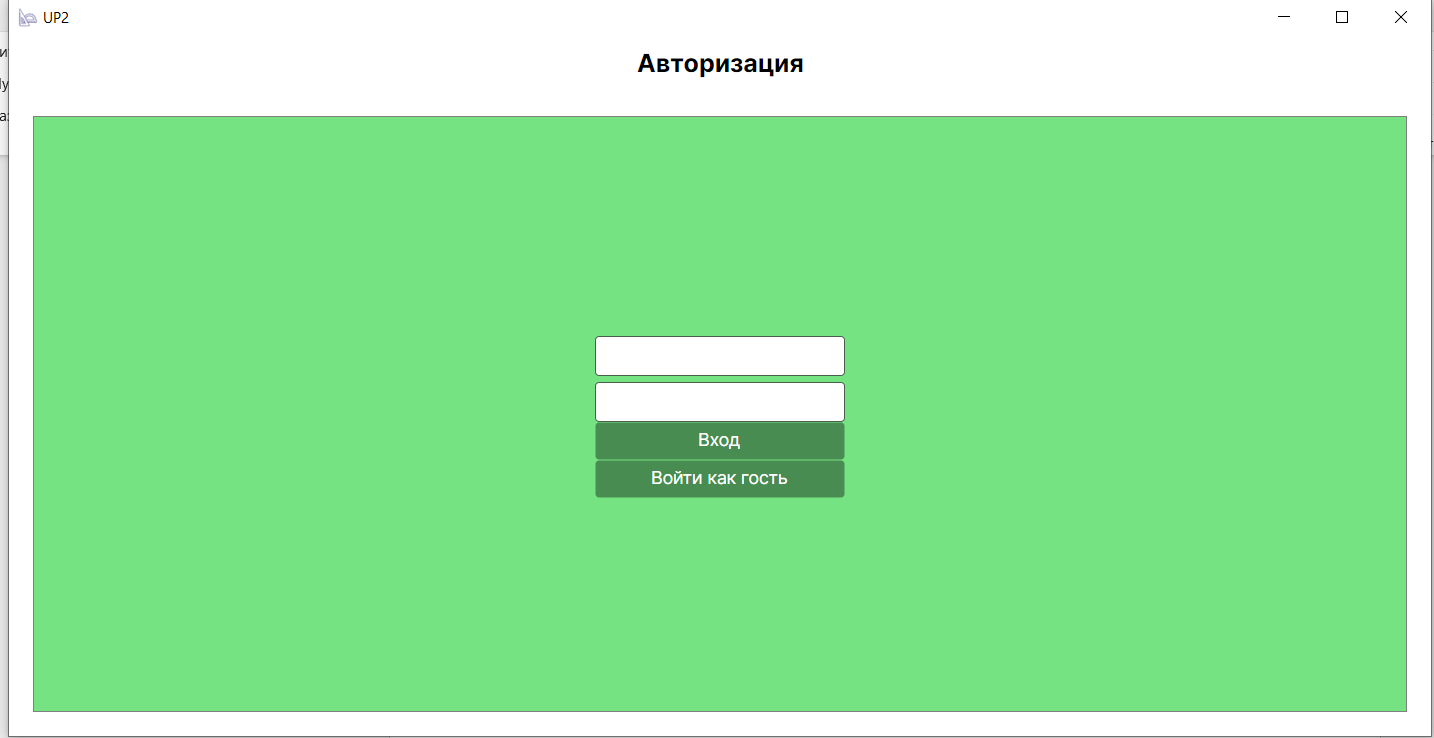


Рисунок №13 – Страница авторизации

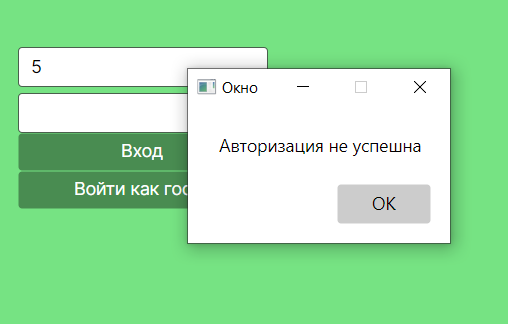


Рисунок №14 – Не успешная авторизация

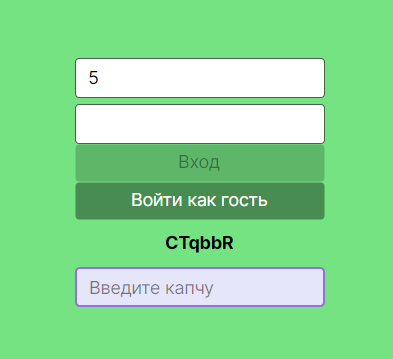


Рисунок №15 – Капча

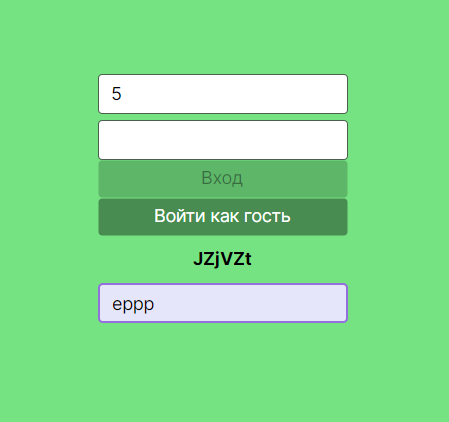


Рисунок №16 – Блокировка кнопки

* 1. Создание страницы вывода продуктов

Общая структура

Страница состоит из следующих основных частей:

Заголовок: Отображает текст "Товары".

Фильтры и управление: Элементы управления для фильтрации и сортировки списка товаров.

Список товаров: Отображает товары в виде списка с возможностью добавления их в корзину.

Элементы управления

Заголовок

<TextBlock Text="Товары" HorizontalAlignment="Center" FontSize="24" FontWeight="Bold" Margin="10,0,0,20"/>

TextBlock: Отображает заголовок "Товары" в центре верхней части страницы. У него задан размер шрифта 24 и жирное начертание, что делает его заметным.

Фильтры и управление

<Grid Margin="20">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

...

</Grid>

Grid: Используется для организации элементов управления в строки и столбцы. Это позволяет удобно располагать фильтры и другую информацию.

Элементы управления в Grid

ComboBox для выбора диапазона скидок:

<ComboBox ItemsSource="{Binding DiscountRanges}"

SelectedItem="{Binding SelectedDiscountRange}"

Width="200" Margin="0,0,20,0" Grid.Column="0" Grid.Row="0">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding}"/>

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

Отображает список диапазонов скидок и позволяет пользователю выбрать один из них.

TextBox для поиска по имени товара:

<TextBox Watermark="Введите наименование товара" Text="{Binding Search}" Height="50" VerticalAlignment="Top" Margin="0,0,20,0" Grid.Column="1" Grid.Row="0"/>

Позволяет пользователю вводить текст для поиска товаров по имени.

RadioButton для сортировки:

<StackPanel Orientation="Horizontal" Grid.Column="2" Grid.Row="0" Margin="0,0,20,0">

<RadioButton Content="По возрастанию" IsChecked="{Binding SortUp}" Margin="0,0,10,0"/>

<RadioButton Content="По убыванию" IsChecked="{Binding SortDown}"/>

</StackPanel>

Два переключателя, которые позволяют пользователю выбрать способ сортировки товаров (по возрастанию или убыванию цены).

Информация о пользователе и количестве товаров:

<StackPanel Orientation="Horizontal" Grid.Column="3" Grid.Row="1" Margin="0,10,0,0">

<TextBlock Text="Пользователь: " />

<TextBlock Text="{Binding CurrentUser .FullName}" Margin="5,0,5,0"/>

<TextBlock Text="Роль: " />

<TextBlock Text="{Binding CurrentUser .Role}" />

</StackPanel>

Отображает имя текущего пользователя и его роль.

Кнопки для навигации:

<StackPanel Orientation="Horizontal" Grid.Column="4" Grid.Row="1" Margin="0,30,0,0">

<Button Command="{Binding ToLast}">Назад </Button>

<Button Command="{Binding ToOrder}">Заказ </Button>

</StackPanel>

Позволяют пользователю вернуться на предыдущий экран или перейти к экрану заказов.

Список товаров

Объяснение элементов списка товаров

ListBox: Основной элемент управления для отображения списка товаров. Он связывается с коллекцией ProductList из ViewModel, что позволяет динамически отображать данные.

DataTemplate: Определяет, как каждый элемент списка будет выглядеть. Внутри DataTemplate мы используем Grid, чтобы организовать элементы по колонкам.

Элементы внутри DataTemplate

Изображение товара:

<Image Height="100" Width="100" Source="{Binding Image, Converter={StaticResource ImageConverter}}" Margin="5" Grid.Column="0" />

Отображает изображение товара. Связывается с полем Image модели товара и использует ImageConverter для преобразования данных изображения.

Информация о товаре:

<StackPanel Grid.Column="1" Margin="5">

<TextBlock Text="{Binding Name}" TextWrapping="Wrap" Margin="0,5,0,0" FontSize="16" FontWeight="Bold"/>

<TextBlock Text="{Binding Description}" TextWrapping="Wrap" Margin="0,5,0,0"/>

<TextBlock>

<Run Text="Производитель:" />

<Run Text="{Binding ProviderNavigation.Name, FallbackValue='Без имени'}"/>

</TextBlock>

<TextBlock>

<Run Text="Цена:" />

<Run Text="{Binding Cost, StringFormat={}{0:F1}}"/>

</TextBlock>

</StackPanel>

Этот StackPanel содержит текстовые блоки, которые отображают название товара, его описание, имя производителя и цену. Используется FallbackValue для отображения текста "Без имени", если имя производителя отсутствует.

Кнопка добавления в корзину:

<Button Command="{Binding $parent[User Control].((vm:ShowProductVM)DataContext).AddZakaz}" CommandParameter="{Binding}">Добавить в корзину</Button>

Кнопка, которая позволяет пользователю добавить товар в корзину. Использует команду AddZakaz из ViewModel, передавая текущий товар как параметр.

Скидка на товар:

<StackPanel Grid.Column="2" Margin="5" VerticalAlignment="Center">

<TextBlock Text="{Binding CurrentDiscount, StringFormat={}{0:F1}}}" TextWrapping="Wrap" Margin="0,5,0,0"/>

</StackPanel>

Отображает текущую скидку на товар. Скидка форматируется с одним десятичным знаком.

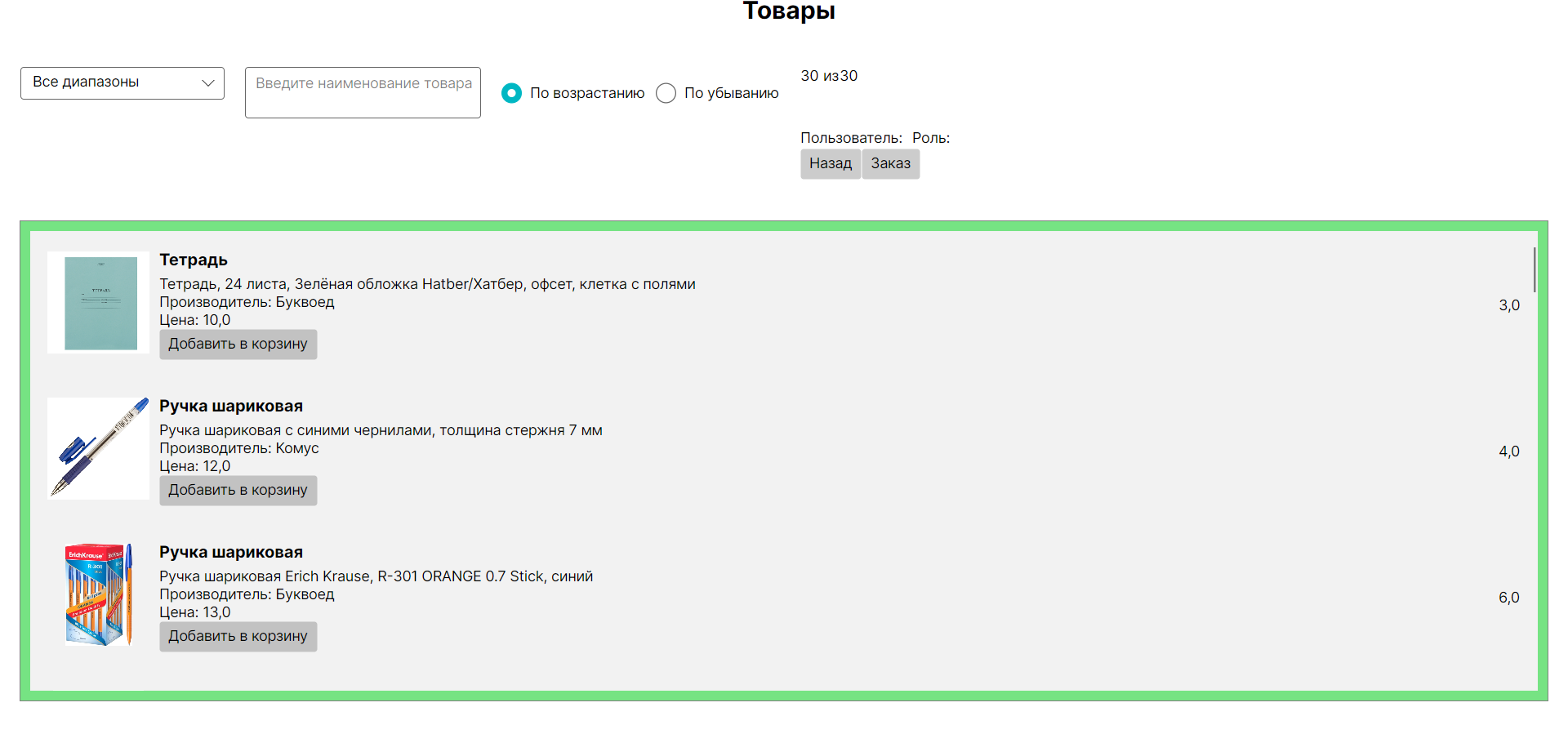


Рисунок №17 – станица Товары

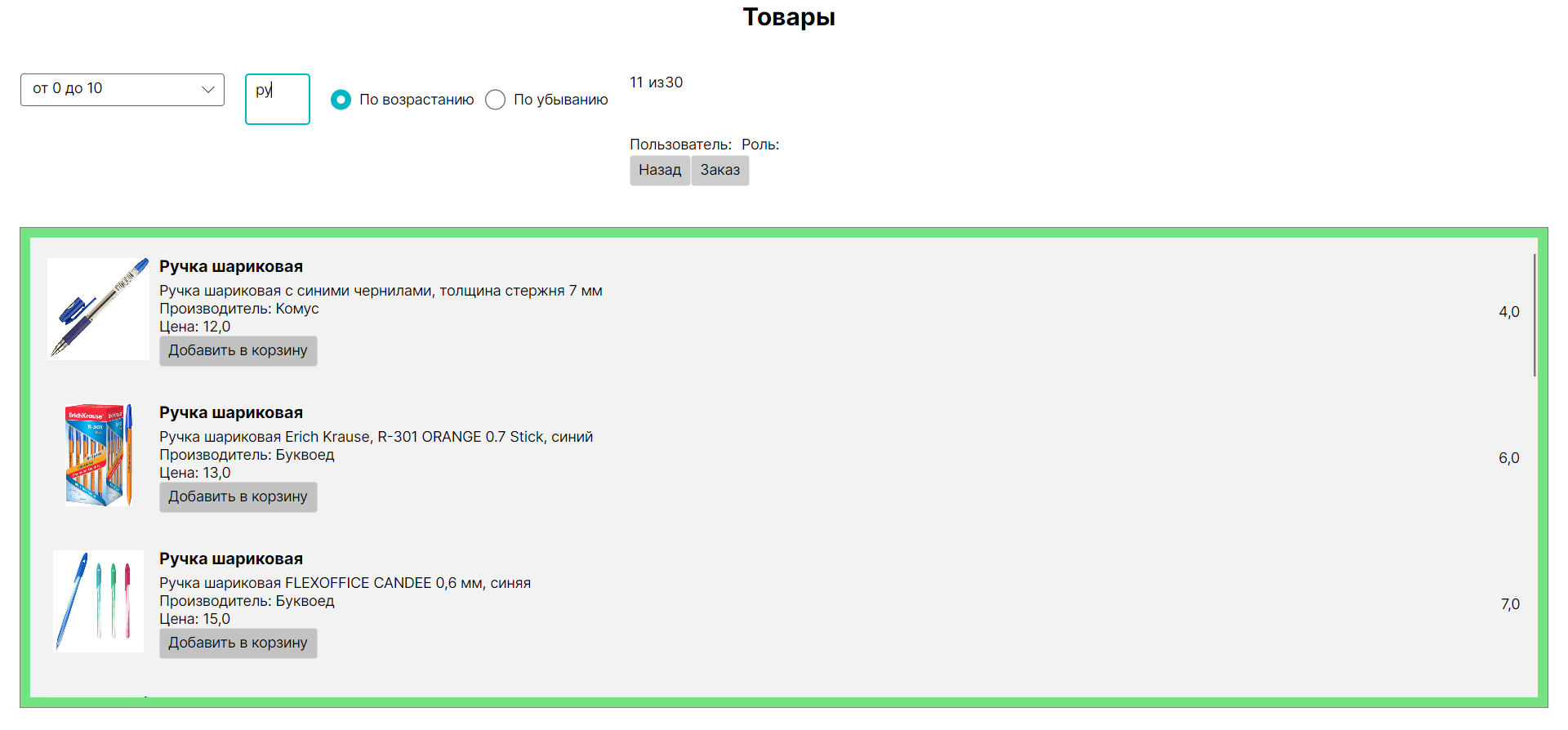


Рисунок №18 – фильтрация, сортировка, поиск одновременно

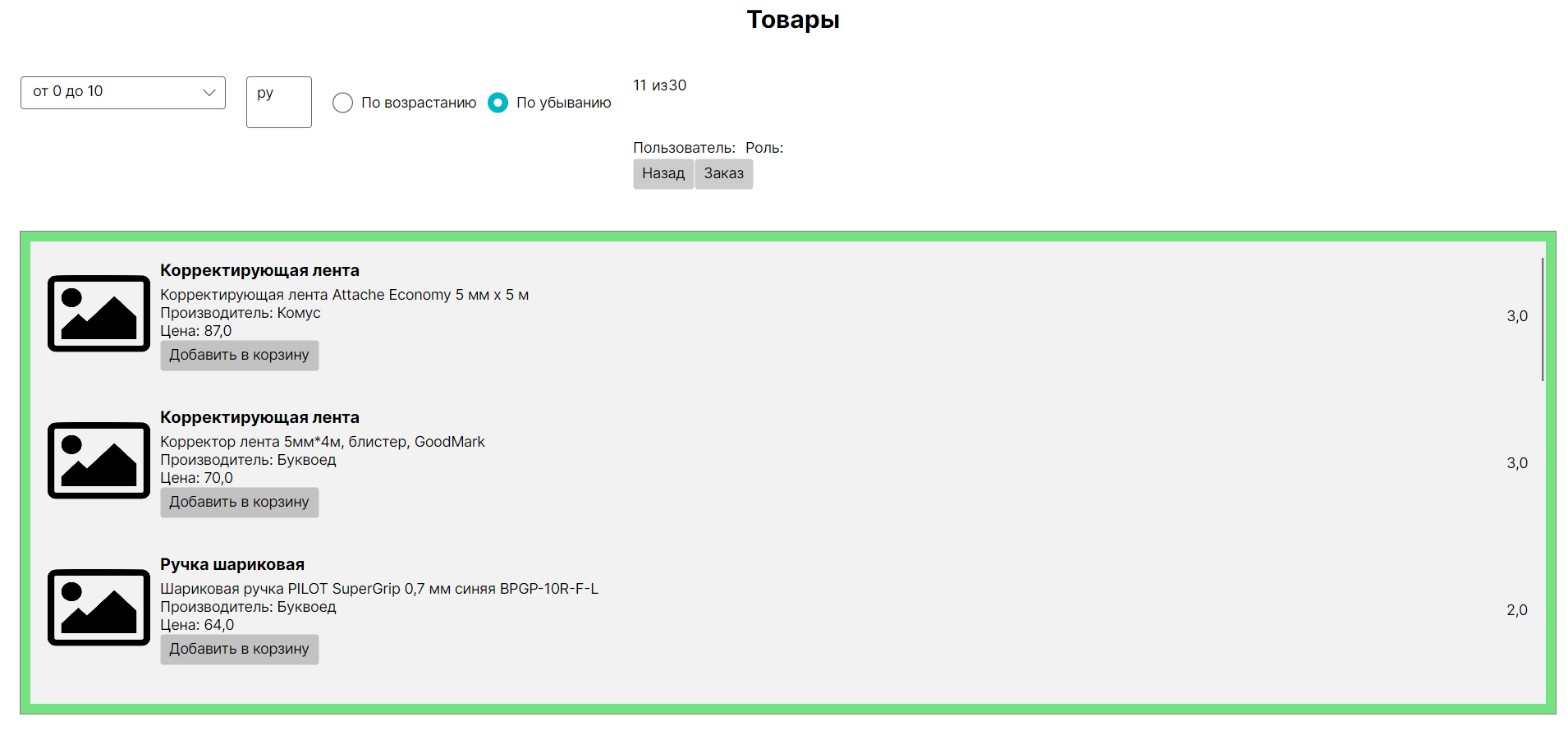


Рисунок №19 – фильтрация, сортировка, поиск заглушки на фото

* 1. Создание страницы Корзина

Страница ShowOrder

Страница ShowOrder представляет собой пользовательский интерфейс для отображения списка заказов. Она включает в себя следующие ключевые элементы:

Верхняя панель управления:

Кнопка "Назад": Позволяет пользователю вернуться на предыдущую страницу.

Информация о сумме заказа: Отображает общую стоимость всех товаров в заказе с возможностью форматирования.

Фильтры: Пользователь может выбрать фильтры для отображения товаров с различными уровнями скидок, такими как "Все", "0-10%", "11-14%", "15% и более".

Кнопки сортировки: Позволяют пользователю сортировать товары по цене в порядке возрастания или убывания.

Список заказов:

Содержит ListBox, который отображает все товары в заказе с их артикулом, количеством, изображением, названием, описанием и ценой.

Каждое нажатие на кнопку "Удалить" позволяет пользователю удалить товар из заказа после подтверждения.

ViewModel ShowOrderVM

ShowOrderVM управляет логикой взаимодействия с данными на странице ShowOrder. Он включает в себя:

Свойства:

FilteredOrderList: Содержит список товаров, отображаемых на странице, который может изменяться в зависимости от выбранного фильтра.

SelectedFilter: Хранит текущий выбранный фильтр, который влияет на отображаемые товары.

TotalOrderCost: Вычисляет общую стоимость всех товаров в заказе.

Команды:

SortByAscendingCommand и SortByDescendingCommand: Позволяют сортировать список товаров по цене.

Delete: Удаляет товар из заказа после подтверждения от пользователя.

Функции фильтрации:

Метод FilterOrders фильтрует список товаров в зависимости от выбранного фильтра, обновляя FilteredOrderList.

Обработка удаления товара:

Метод Delete запрашивает подтверждение у пользователя перед удалением товара из базы данных.

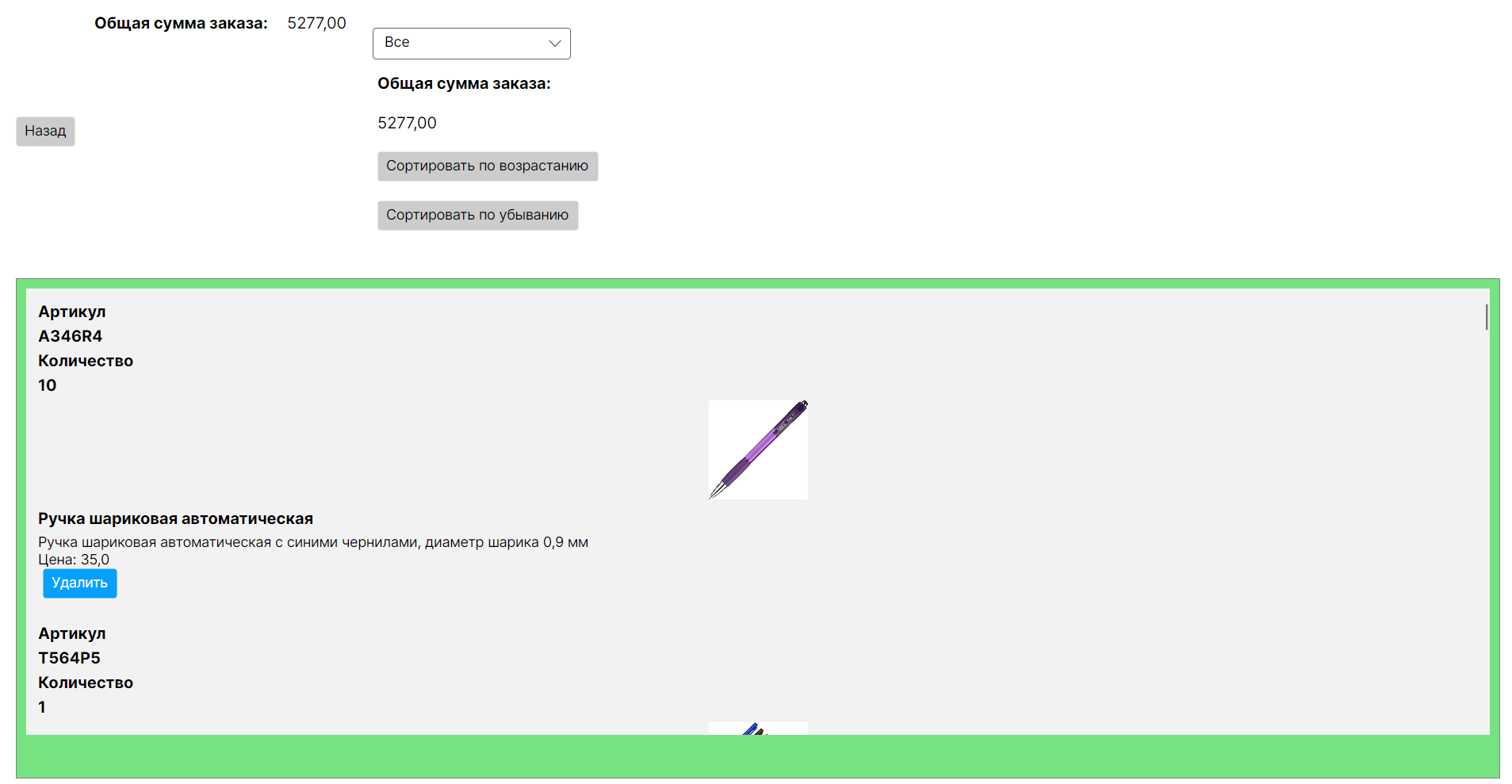


Рисунок №20 – Вывод корзины



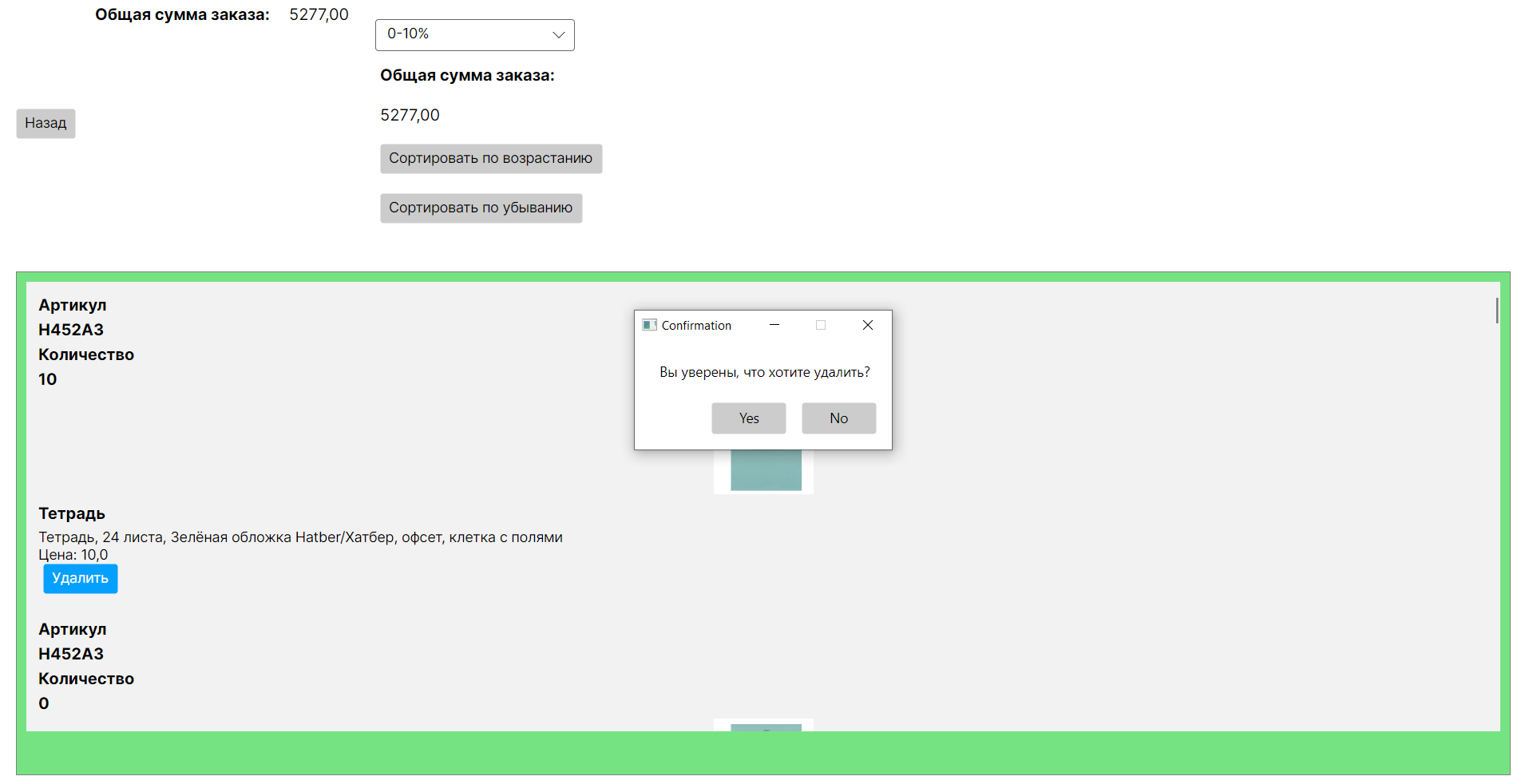
Рисунок №21 – Сортировка 

Рисунок №22 – Удаление



Рисунок №23 – Результат удаления

1. Разработка и тестирование dll библиотеки
   1. Разработка библиотеки

Структура класса

Класс Calculations содержит один публичный метод:

AvailablePeriods: Этот метод принимает несколько параметров и возвращает массив строк, представляющих доступные временные интервалы для консультаций.

Параметры метода AvailablePeriods

Метод AvailablePeriods принимает следующие параметры:

TimeSpan[] startTimes: Массив, содержащий время начала занятых периодов. Каждый элемент представляет собой TimeSpan, который указывает, когда начинается занятие.

int[] durations: Массив целых чисел, представляющих продолжительность каждого занятия в минутах. Длительность каждого элемента соответствует времени начала из массива startTimes.

TimeSpan beginWorkingTime: Время начала рабочего дня. Это значение указывает, когда начинается доступное время для консультаций.

TimeSpan endWorkingTime: Время окончания рабочего дня. Это значение указывает, когда заканчивается доступное время для консультаций.

int consultationTime: Продолжительность консультации в минутах. Это значение определяет, сколько времени нужно для каждой консультации.

Логика метода

Теперь давайте разберем логику метода AvailablePeriods шаг за шагом:

Создание списка занятых периодов:

Инициализируется список busyPeriods, который будет хранить занятые временные интервалы.

В цикле for добавляются в busyPeriods время начала каждого занятия и время его окончания (начало + продолжительность).

for (int i = 0; i < startTimes.Length; i++)

{

busyPeriods.Add(startTimes[i]);

busyPeriods.Add(startTimes[i].Add(TimeSpan.FromMinutes(durations[i])));

}

Создание списка доступных периодов:

Инициализируется список availablePeriods, который будет хранить доступные временные интервалы для консультаций.

Переменная currentStart устанавливается на beginWorkingTime, чтобы отслеживать текущее доступное время для консультаций.

Сортировка занятых периодов:

Занятые интервалы сортируются для корректной обработки. Это важно, чтобы правильно определить свободное время между занятиями.

busyPeriods.Sort();

Определение доступных периодов:

Цикл for обрабатывает занятые интервалы по парам (начало и конец).

Внутри цикла проверяется, есть ли свободное время между currentStart и busyStart (началом занятого интервала).

Если свободное время достаточно для консультации (т.е. currentStart + consultationTime меньше или равно busyStart), то добавляется новый доступный интервал в availablePeriods, и currentStart увеличивается на 30 минут.

while (currentStart + TimeSpan.FromMinutes(consultationTime) <= busyStart)

{

availablePeriods.Add($"{currentStart:hh\\:mm}-{currentStart.Add(TimeSpan.FromMinutes(consultationTime)):hh\\:mm}");

currentStart = currentStart.Add(TimeSpan.FromMinutes(30));

}

Перемещение текущего времени:

После обработки каждого занятого интервала currentStart перемещается на конец этого интервала, чтобы продолжить проверку доступных периодов.

currentStart = busyEnd > currentStart ? busyEnd : currentStart;

Проверка свободного времени после последнего занятого интервала:

После завершения цикла по занятым периодам метод проверяет, есть ли свободное время между последним занятым интервалом и концом рабочего дня.

Если есть, то добавляются доступные интервалы в availablePeriods.

while (currentStart + TimeSpan.FromMinutes(consultationTime) <= endWorkingTime)

{

availablePeriods.Add($"{currentStart:hh\\:mm}-{currentStart.Add(TimeSpan.FromMinutes(consultationTime)):hh\\:mm}");

currentStart = currentStart.Add(TimeSpan.FromMinutes(30));

}

Возврат результата:

Метод возвращает массив строк, представляющих доступные временные интервалы для консультаций.

После того как все доступные временные интервалы для консультаций были определены, метод AvailablePeriods завершает свою работу следующим образом:

Форматирование результата:

Доступные временные интервалы, которые были добавлены в список availablePeriods, форматируются в виде строк. Каждая строка представляет собой интервал времени в формате "HH:mm-HH:mm".

Возврат массива строк:

В конце метода происходит возврат массива строк, представляющих все доступные временные интервалы для консультаций. Это позволяет пользователю или вызывающему коду легко видеть, в какие моменты времени можно записаться на консультацию.

return availablePeriods.ToArray();

Пример использования

Теперь, когда мы разобрали логику метода AvailablePeriods, давайте рассмотрим пример его использования:

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0), new TimeSpan(10, 30, 0) }; // Занятия начинаются в 9:00 и 10:30

int[] durations = { 60, 30 }; // Первое занятие длится 60 минут, второе - 30 минут

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0); // Рабочий день начинается в 9:00

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0); // Рабочий день заканчивается в 17:00

int consultationTime = 30; // Продолжительность консультации - 30 минут

Calculations calculations = new Calculations();

string[] availablePeriods = calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

foreach (var period in availablePeriods)

{

Console.WriteLine(period); // Вывод доступных временных интервалов

}

Ожидаемый вывод

В зависимости от занятых периодов и рабочего времени, ожидаемый вывод может выглядеть следующим образом:

09:00-09:30

09:30-10:00

10:00-10:30

10:30-11:00

11:00-11:30

...

16:30-17:00

* 1. Написание Юнит тестов

Test\_All\_Free\_Periods

Цель: Проверить, что метод возвращает все возможные временные интервалы, когда нет занятых периодов.

Входные данные:

startTimes: пустой массив (нет занятых периодов).

durations: пустой массив.

beginWorkingTime: 09:00.

endWorkingTime: 17:00.

consultationTime: 30 минут.

Ожидаемый результат: Массив строк с временными интервалами от 09:00 до 17:00 с шагом 30 минут.

Test\_Partial\_Busy\_Periods

Цель: Проверить, что метод корректно обрабатывает частично занятые периоды.

Входные данные:

startTimes: 09:00 и 10:00 (два занятых периода).

durations: 30 минут для каждого.

beginWorkingTime: 09:00.

endWorkingTime: 12:00.

consultationTime: 30 минут.

Ожидаемый результат: Доступные интервалы: "09:30-10:00", "10:30-11:00", "11:00-11:30", "11:30-12:00".

Test\_Multiple\_Consultations

Цель: Проверить, что метод правильно обрабатывает несколько консультаций.

Входные данные:

startTimes: 09:00, 10:00, 11:00.

durations: 30 минут для каждого.

beginWorkingTime: 09:00.

endWorkingTime: 17:00.

consultationTime: 30 минут.

Ожидаемый результат: Доступные интервалы от 09:30 до 17:00 с шагом 30 минут.

Test\_Fully\_Busy\_Day

Цель: Проверить, что метод возвращает пустой массив, если весь день занят.

Входные данные:

startTimes: 09:00 (один занятой период).

durations: 480 минут (8 часов).

beginWorkingTime: 09:00.

endWorkingTime: 17:00.

consultationTime: 30 минут.

Ожидаемый результат: Пустой массив, так как весь день занят.

Test\_No\_Busy\_Periods

Цель: Проверить, что метод возвращает все временные интервалы, если нет занятых периодов.

Входные данные: Аналогично Test\_All\_Free\_Periods, с теми же параметрами.

Ожидаемый результат: То же, что и в предыдущем тесте — все интервалы от 09:00 до 17:00 с шагом 30 минут.

Test\_ConsultationTime\_At\_Start\_Of\_Workday

Цель: Проверить, что метод правильно обрабатывает ситуацию, когда консультация начинается в начале рабочего дня.

Входные данные:

startTimes: 09:00.

durations: 60 минут.

beginWorkingTime: 09:00.

endWorkingTime: 17:00.

consultationTime: 30 минут.

Ожидаемый результат: Доступные интервалы от 10:00 до 17:00 с шагом 30 минут.

Test\_ConsultationTime\_Exactly\_Busy\_Periods

Цель: Проверить, что метод не возвращает интервалы, совпадающие с занятыми периодами.

Входные данные:

startTimes: 10:00, 11:00.

durations: 30 минут для каждого.

beginWorkingTime: 09:00.

endWorkingTime: 17:00.

consultationTime: 30 минут.

Ожидаемый результат: Доступные интервалы должны быть:

"09:00-09:30"

"09:30-10:00"

"10:30-11:00"

"11:30-12:00"

"12:00-12:30"

"12:30-13:00"

"13:00-13:30"

"13:30-14:00"

"14:00-14:30"

"14:30-15:00"

"15:00-15:30"

"15:30-16:00"

"16:00-16:30"

"16:30-17:00"

Test\_ConsultationTime\_Larger\_Than\_Free\_Slots

Цель: Проверить, что метод правильно работает, когда время консультации больше, чем доступные интервалы.

Входные данные:

startTimes: 10:00 и 11:00.

durations: 30 минут для каждого.

beginWorkingTime: 09:00.

endWorkingTime: 17:00.

consultationTime: 120 минут (2 часа).

Ожидаемый результат: Метод должен вернуть 8 доступных интервалов, так как с учетом занятых периодов остаются временные слоты, которые могут вместить 2-часовую консультацию.

Test\_ConsultationTime\_Available\_At\_Multiple\_Intervals

Цель: Проверить, что метод возвращает доступные интервалы, когда есть несколько свободных промежутков.

Входные данные:

startTimes: 10:00, 12:00 и 14:00.

durations: 30 минут для каждого.

beginWorkingTime: 09:00.

endWorkingTime: 17:00.

consultationTime: 30 минут.

Ожидаемый результат: Доступные интервалы должны включать все временные промежутки от 09:00 до 17:00, за исключением занятых, что дает:

"09:00-09:30"

"09:30-10:00"

"10:30-11:00"

"11:00-11:30"

"11:30-12:00"

"12:30-13:00"

"13:00-13:30"

"13:30-14:00"

"14:30-15:00"

"15:00-15:30"

"15:30-16:00"

"16:00-16:30"

"16:30-17:00"

Test\_Single\_Busy\_Period\_At\_Start

Цель: Проверить, что метод правильно возвращает доступные интервалы, когда есть единственный занятый период в начале рабочего дня.

Входные данные:

startTimes: 09:00.

durations: 60 минут.

beginWorkingTime: 09:00.

endWorkingTime: 17:00.

consultationTime: 30 минут.

Ожидаемый результат: Доступные интервалы должны быть от 10:00 до 17:00 с шагом 30 минут:

"10:00-10:30"

"10:30-11:00"

"11:00-11:30"

"11:30-12:00"

"12:00-12:30"

"12:30-13:00"

"13:00-13:30"

"13:30-14:00"

"14:00-14:30"

"14:30-15:00"

"15:00-15:30"

"15:30-16:00"

"16:00-16:30"

"16:30-17:00"

AvailablePeriods\_NoAppointments\_CorrectlyIdentifiesAllSlots

Цель: Проверить, что метод правильно определяет все временные слоты, когда нет занятых периодов.

Входные данные:

startTimes: пустой массив.

durations: пустой массив.

beginWorkingTime: 09:

Ожидаемый результат: Доступные интервалы должны быть:

"09:00-09:30"

"09:30-10:00"

"10:30-11:00"

"11:30-12:00"

"12:00-12:30"

"12:30-13:00"

"13:00-13:30"

"13:30-14:00"

"15:00-15:30"

"15:30-16:00"

"16:00-16:30"

"16:30-17:00"

AvailablePeriods\_OverlappingAppointments\_DoesNotReturnOverlappingSlots

Цель: Проверить, что метод не возвращает временные интервалы, которые перекрываются с уже занятыми.

Входные данные:

Время начала: 09:00 и 09:30.

Длительность: 45 и 30 минут соответственно.

Рабочее время: с 09:00 до 10:00.

Время консультации: 30 минут.

Ожидаемый результат: Метод не должен возвращать интервал "09:45-10:15", так как он перекрывается с первой занятой консультацией (09:00-09:45).

Assert.IsFalse(actual.Contains("09:45-10:15"));

2. AvailablePeriods\_NoFreeSlots\_ReturnsEmptyArray

Цель: Проверить, что метод возвращает пустой массив, если все доступные временные слоты заняты.

Входные данные:

Время начала: 09:00.

Длительность: 600 минут (10 часов).

Рабочее время: с 09:00 до 10:00.

Время консультации: 30 минут.

Ожидаемый результат: Метод должен вернуть пустой массив, так как весь день занят одной встречей, превышающей рабочее время.

Assert.AreEqual(0, actual.Length);

AvailablePeriods\_IncorrectConsultationTime\_DoesNotReturnInvalidSlots

Цель: Проверить, что метод не возвращает временные интервалы, если запрашиваемое время консультации превышает доступные слоты.

Входные данные:

Время начала: 09:00.

Длительность: 60 минут.

Рабочее время: с 09:00 до 10:00.

Время консультации: 75 минут.

Ожидаемый результат: Метод не должен возвращать ни одного интервала, так как запрашиваемая длительность консультации превышает доступное время.

Assert.AreNotEqual(1, actual.Length); // Не должно быть ни одного интервала

AvailablePeriods\_EdgeCase\_DoesNotReturnIncorrectSlots

Цель: Проверить, что метод корректно обрабатывает крайние случаи, когда начало рабочего времени совпадает с началом встречи.

Входные данные:

Время начала: 09:00.

Длительность: 30 минут.

Рабочее время: с 09:00 до 12:00.

Время консультации: 30 минут.

Ожидаемый результат: Метод не должен возвращать интервал "09:00-09:30", так как этот интервал полностью занят первой консультацией.

Assert.IsFalse(actual.Contains("09:00-09:30")); // Не должно быть интервала 9:00-9:30

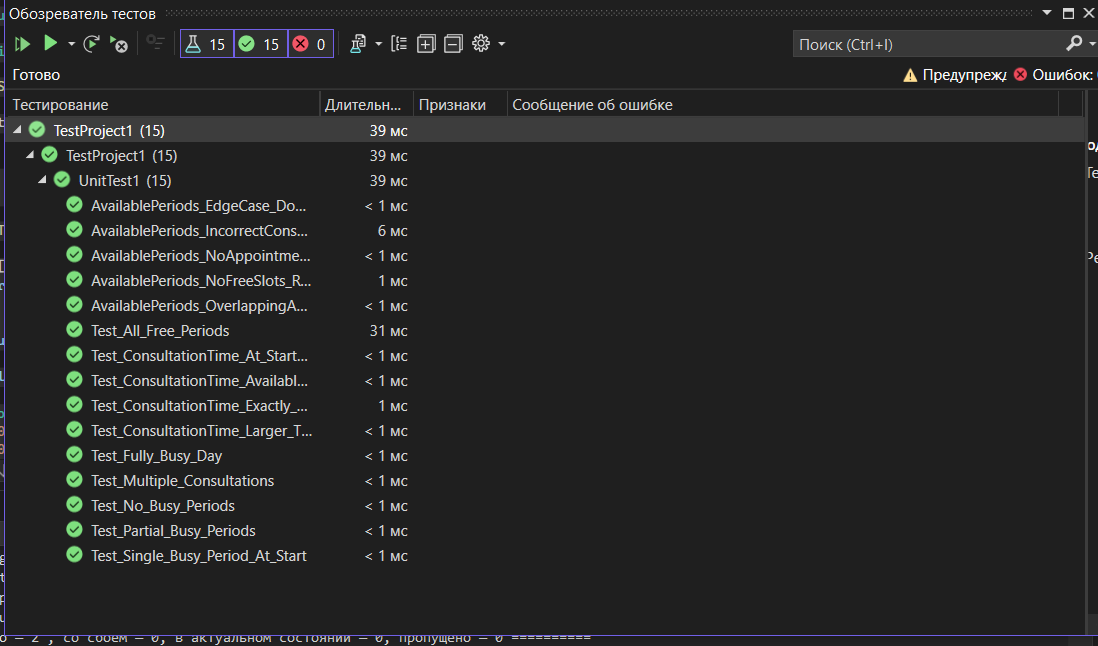


Рисунок №24 – Юнит тесты

Заключение

В ходе прохождения учебной практики по разработке программных модулей существенно расширены знания и навыки в области программирования. Успешно выполненные задачи по изучению технического задания, разработке базы данных и пользовательского интерфейса, а также реализация функционала приложения позволили понять весь процесс создания программного обеспечения от начала до конца.

Особое внимание было уделено отладке и тестированию, что подтвердило важность этих этапов в разработке. Рефакторинг и оптимизация кода стали важными шагами для улучшения качества и производительности приложения.

Практика способствовала не только профессиональному росту, но и углублению теоретических знаний, что в дальнейшем поможет в реализации более сложных проектов. Полученные навыки и опыт окажут положительное влияние на карьеру в сфере информационных технологий.

Приложения

Приложение А

**CREATE** **TABLE** Kategory (

id SERIAL **PRIMARY** **KEY**,

name **VARCHAR**(100) **NOT** **NULL**

);

**CREATE TABLE Orders (**

**id SERIAL PRIMARY KEY,**

**date\_order DATE NOT NULL,**

**date\_delivery DATE NOT NULL,**

**pick\_up INT NOT NULL,**

**FIO VARCHAR(200),**

**code INT NOT NULL,**

**status INT NOT NULL,**

**FOREIGN KEY (pick\_up) REFERENCES Pick\_up\_point(id),**

**FOREIGN KEY (status) REFERENCES OrderStatus(StatusID)**

**);**

**CREATE** **TABLE** Product (

id SERIAL **PRIMARY** **KEY**,

article\_number **VARCHAR**(100) **NOT** **NULL**,

name **VARCHAR**(255) **NOT** **NULL**,

unit **VARCHAR**(50) **NOT** **NULL**,

cost **DECIMAL**(19, 4) **NOT** **NULL**,

max\_discount\_amount **INT**,

production **INT** **NOT** **NULL**,

provider **INT** **NOT** **NULL**,

kategory **INT** **NOT** **NULL**,

current\_discount **DECIMAL**(19, 4),

quantity\_in\_stock **INT** **NOT** **NULL**,

description **TEXT**,

image **VARCHAR**(100),

**FOREIGN** **KEY** (provider) **REFERENCES** Provider(id),

**FOREIGN** **KEY** (kategory) **REFERENCES** Kategory(id),

**FOREIGN** **KEY** (production) **REFERENCES** Production(id)

);

**CREATE** **TABLE** Production (

id SERIAL **PRIMARY** **KEY**,

name **VARCHAR**(100) **NOT** **NULL**

);

**CREATE** **TABLE** Provider (

id SERIAL **PRIMARY** **KEY**,

name **VARCHAR**(100) **NOT** **NULL**

);

**CREATE** **TABLE** Role (

RoleID SERIAL **PRIMARY** **KEY**,

RoleName **VARCHAR**(100) **NOT** **NULL**

);

**create** **table** Users

(

UserID **int** **primary** **key**,

UserSurname **varchar**(100) **not** **null**,

UserName **varchar**(100) **not** **null**,

UserPatronymic **varchar**(100) **not** **null**,

UserLogin **text** **not** **null**,

UserPassword **text** **not** **null**,

UserRole **int** **not** **null**,

**foreign** **key** (UserRole) **references** Role(RoleID)

);

**CREATE** **TABLE** OrderProduct (

id SERIAL **PRIMARY** **KEY**,

OrderID **INT** **NOT** **NULL**,

ProductArticleNumber **VARCHAR**(100) **NOT** **NULL**,

Quantity **INT** **NOT** **NULL**,

**FOREIGN** **KEY** (OrderID) **REFERENCES** Orders(OrderID) **ON** **DELETE** **CASCADE**,

**FOREIGN** **KEY** (ProductArticleNumber) **REFERENCES** Product(article\_number) **ON** **DELETE** **CASCADE**

);

Приложение Б

private Calculations \_calculations;

[TestInitialize]

public void Setup()

{

\_calculations = new Calculations();

}

// Проверяет ситуацию, когда все время свободно

[TestMethod]

public void Test\_All\_Free\_Periods()

{

TimeSpan[] startTimes = { }; // Нет занятых периодов

int[] durations = { }; // Нет длительностей

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0);

int consultationTime = 30;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(new string[] {

"09:00-09:30", "09:30-10:00", "10:00-10:30", "10:30-11:00",

"11:00-11:30", "11:30-12:00", "12:00-12:30", "12:30-13:00",

"13:00-13:30", "13:30-14:00", "14:00-14:30", "14:30-15:00",

"15:00-15:30", "15:30-16:00", "16:00-16:30", "16:30-17:00"

}, result);

}

// Проверяет доступные интервалы при частично занятых периодах

[TestMethod]

public void Test\_Partial\_Busy\_Periods()

{

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0), new TimeSpan(10, 0, 0) };

int[] durations = { 30, 30 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(12, 0, 0);

int consultationTime = 30;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(new string[] {

"09:30-10:00", "10:30-11:00", "11:00-11:30", "11:30-12:00"

}, result);

}

// Проверяет доступные интервалы при нескольких консультациях

[TestMethod]

public void Test\_Multiple\_Consultations()

{

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0), new TimeSpan(10, 0, 0), new TimeSpan(11, 0, 0) };

int[] durations = { 30, 30, 30 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0);

int consultationTime = 30;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(new string[] {

"09:30-10:00", "10:30-11:00", "11:30-12:00", "12:00-12:30",

"12:30-13:00", "13:00-13:30", "13:30-14:00", "14:00-14:30",

"14:30-15:00", "15:00-15:30", "15:30-16:00", "16:00-16:30",

"16:30-17:00"

}, result);

}

// Проверяет, что доступные интервалы пусты, если весь день занят

[TestMethod]

public void Test\_Fully\_Busy\_Day()

{

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0) };

int[] durations = { 480 }; // 8 часов

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0);

int consultationTime = 30;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(new string[] { }, result); // Ожидаем пустой список

}

// Проверяет доступные интервалы при отсутствии занятых периодов

[TestMethod]

public void Test\_No\_Busy\_Periods()

{

TimeSpan[] startTimes = { };

int[] durations = { };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0);

int consultationTime = 30;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(new string[] {

"09:00-09:30", "09:30-10:00", "10:00-10:30", "10:30-11:00",

"11:00-11:30", "11:30-12:00", "12:00-12:30", "12:30-13:00",

"13:00-13:30", "13:30-14:00", "14:00-14:30", "14:30-15:00",

"15:00-15:30", "15:30-16:00", "16:00-16:30", "16:30-17:00"

}, result);

}

// Проверяет доступные интервалы, когда консультация начинается в начале рабочего дня

[TestMethod]

public void Test\_ConsultationTime\_At\_Start\_Of\_Workday()

{

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0) };

int[] durations = { 60 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0);

int consultationTime = 30;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(new string[] {

"10:00-10:30", "10:30-11:00", "11:00-11:30", "11:30-12:00",

"12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00",

"14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00",

"16:00-16:30", "16:30-17:00"

}, result);

}

// Проверяет доступные интервалы, когда консультация совпадает с занятыми периодами

[TestMethod]

public void Test\_ConsultationTime\_Exactly\_Busy\_Periods()

{

TimeSpan[] startTimes = { new TimeSpan(10, 0, 0), new TimeSpan(11, 0, 0) };

int[] durations = { 30, 30 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0);

int consultationTime = 30;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(new string[] {

"09:00-09:30", "09:30-10:00", "10:30-11:00", "11:30-12:00",

"12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00",

"14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00",

"16:00-16:30", "16:30-17:00"

}, result);

}

// Проверяет, что время консультации больше, чем свободные интервалы

[TestMethod]

public void Test\_ConsultationTime\_Larger\_Than\_Free\_Slots()

{

TimeSpan[] startTimes = { new TimeSpan(10, 0, 0), new TimeSpan(11, 0, 0) };

int[] durations = { 30, 30 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0);

int consultationTime = 120;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

Assert.AreEqual(8, result.Length);

}

// Проверяет доступные интервалы при множественных свободных промежутках

[TestMethod]

public void Test\_ConsultationTime\_Available\_At\_Multiple\_Intervals()

{

TimeSpan[] startTimes = { new TimeSpan(10, 0, 0), new TimeSpan(12, 0, 0), new TimeSpan(14, 0, 0) };

int[] durations = { 30, 30, 30 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0);

int consultationTime = 30;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(new string[] {

"09:00-09:30", "09:30-10:00", "10:30-11:00", "11:00-11:30",

"11:30-12:00", "12:30-13:00", "13:00-13:30", "13:30-14:00",

"14:30-15:00", "15:00-15:30", "15:30-16:00", "16:00-16:30",

"16:30-17:00"

}, result);

}

// Проверяет доступные интервалы при единственном занятом периоде в начале рабочего дня

[TestMethod]

public void Test\_Single\_Busy\_Period\_At\_Start()

{

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0) };

int[] durations = { 60 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(17, 0, 0);

int consultationTime = 30;

var result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(new string[] {

"10:00-10:30", "10:30-11:00", "11:00-11:30", "11:30-12:00",

"12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00",

"14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00",

"16:00-16:30", "16:30-17:00"

}, result);

}

[TestMethod]

public void AvailablePeriods\_NoAppointments\_CorrectlyIdentifiesAllSlots()

{

// Нет занятых интервалов

TimeSpan[] startTimes = { };

int[] durations = { };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(12, 0, 0);

int consultationTime = 30;

List<string> expected = new List<string>();

for (TimeSpan time = beginWorkingTime; time < endWorkingTime; time = time.Add(TimeSpan.FromMinutes(30)))

{

expected.Add($"{time:hh\\:mm}-{time.Add(TimeSpan.FromMinutes(consultationTime)):hh\\:mm}");

}

string[] actual = new Calculations().AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

CollectionAssert.AreEqual(expected.ToArray(), actual);

}

[TestMethod]

public void AvailablePeriods\_OverlappingAppointments\_DoesNotReturnOverlappingSlots()

{

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0), new TimeSpan(9, 30, 0) };

int[] durations = { 45, 30 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(10, 0, 0);

int consultationTime = 30;

string[] actual = new Calculations().AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

// Проверяем, что функция не вернула интервал 9:45-10:15, который перекрывается с занятыми

Assert.IsFalse(actual.Contains("09:45-10:15"));

}

[TestMethod]

public void AvailablePeriods\_NoFreeSlots\_ReturnsEmptyArray()

{

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0) };

int[] durations = { 600 }; // 10 часов

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(10, 0, 0);

int consultationTime = 30;

string[] actual = new Calculations().AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

Assert.AreEqual(0, actual.Length); // Должен вернуть пустой массив

}

[TestMethod]

public void AvailablePeriods\_IncorrectConsultationTime\_DoesNotReturnInvalidSlots()

{

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0) };

int[] durations = { 60 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(10, 0, 0);

int consultationTime = 75; // Запрос на 75 минут, а слоты только по 30

string[] actual = new Calculations().AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

Assert.AreNotEqual(1, actual.Length); // Не должно быть ни одного интервала

}

[TestMethod]

public void AvailablePeriods\_EdgeCase\_DoesNotReturnIncorrectSlots()

{

//Краевой случай: начало рабочего дня совпадает с началом встречи

TimeSpan[] startTimes = { new TimeSpan(9, 0, 0) };

int[] durations = { 30 };

TimeSpan beginWorkingTime = new TimeSpan(9, 0, 0);

TimeSpan endWorkingTime = new TimeSpan(12, 0, 0);

int consultationTime = 30;

string[] actual = new Calculations().AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);

Assert.IsFalse(actual.Contains("09:00-09:30")); // Не должно быть интервала 9:00-9:30

}