**Государственное бюджетное профессиональное образовательное учреждение**

**«Нижегородский Губернский колледж»**

Методическая комиссия «Информатика и вычислительная техника»

Допущен к защите: \_\_\_\_\_\_\_\_

преподаватель\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_ 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.

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Трянина М.А. 21.03.2025 г.

Специальность, группа: 09.02.07, 43П

Нижний Новгород

2025 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc192926186)

[1. РАЗРАБОТКА, АДМИНИСТРИРОВАНИЕ И ЗАЩИТА БАЗ ДАННЫХ 5](#_Toc192926187)

[1.1 Проектирование базы данных 5](#_Toc192926188)

[1.2 Разработка базы данных 6](#_Toc192926189)

[2. РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ 11](#_Toc192926190)

[2.1 Основа приложения 11](#_Toc192926191)

[2.2 Подключение к базе данных 12](#_Toc192926192)

[2.3 Обеспечение навигации в приложении 13](#_Toc192926193)

[2.4 Авторизация в системе 14](#_Toc192926194)

[2.5 Работа с товарами и информацией о них 15](#_Toc192926195)

[3. СОПРОВОЖДЕНИЕ И ОБСЛУЖИВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ КОМПЬЮТЕРНЫХ СИСТЕМ 18](#_Toc192926196)

[3.1 Формирование заказа 18](#_Toc192926197)

[3.2 Просмотр и управление списком заказов 20](#_Toc192926198)

[3.3 Тестирование процедуры добавления товара 22](#_Toc192926199)

[4. ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ 28](#_Toc192926200)

[4.1 Разработка DLL-библиотеки 28](#_Toc192926201)

[4.2 Разработка Unit-тестов к библиотеке 28](#_Toc192926202)

[ЗАКЛЮЧЕНИЕ 30](#_Toc192926203)

[ПРИЛОЖЕНИЯ 31](#_Toc192926204)

[ПРИЛОЖЕНИЕ А 31](#_Toc192926205)

ВВЕДЕНИЕ

Учебная практика по ПМ.01 Разработка модулей программного обеспечения для компьютерных систем проходила в ГБПОУ Нижегородский Губернский колледж в период с 01.03.2025 по 21.03.2025.

Актуальность прохождения практики обусловлена высоким спросом на квалифицированных программистов в условиях цифровизации и роста IT-отрасли. Разработка программных модулей является ключевым этапом создания современных информационных систем, включая клиент-серверные приложения и мобильные платформы

Целью данной учебной практики является приобретение теоретических и практических навыков разработки модулей программного обеспечения для компьютерных систем. Это включает создание алгоритмов, написание кода, тестирование, отладку и оформление документации на программные продукты.

Для достижения этой цели необходимо добиться выполнения ряда задач:

* осуществление разработки кода программного модуля на языках низкого и высокого уровней;
* создание программ по разработанному алгоритму как отдельный модуль;
* выполнение отладки и тестирования программ на уровне модуля;
* осуществление разработки кода программного модуля на современных языках программирования;
* выполнение оптимизации и рефакторинга программного кода;
* оформление документации на программные средства.

Результатом прохождения практики должно стать приобретение опыта по:

* разработке кода программного продукта на основе готовой спецификации на уровне модуля;
* использованию инструментальных средств на этапе отладки программного продукта;
* проведению тестирования программного модуля по определенному сценарию;
* разработке мобильных приложений.

1. РАЗРАБОТКА, АДМИНИСТРИРОВАНИЕ И ЗАЩИТА БАЗ ДАННЫХ
   1. Проектирование базы данных

В результате анализа предложенной в задании предметной области можно представить список некоторых сущностей БД, отвечающих за основную логику работы программы и хранение данных предприятия Пиши-стирай. Ниже представлены краткие описания каждой из них.

1. Пользователи – все субъекты, как-либо взаимодействующие с данным программным обеспечением и базой данных, в частности, а также некоторая информация о них: фамилия, имя, отчество, логин, пароль, роль в системе.
2. Роли – разнообразие имеющихся в системе наборов прав на взаимодействие с ней. Обязательно присваивается субъекту при работе с программным обеспечением, вне зависимости от того, авторизован он или нет.
3. Продукты – сущность, содержащая основную информацию о продукции предприятия. В частности, это значения артикул, наименование, описание, категорию товара, изображение, производителя, стоимость, размер скидки, количество на складе, статус продукта, максимальный объем скидки, единицы измерения и поставщик.
4. Категории продукции – классификация продукции в соответствии с заранее установленными в предметной области типами.
5. Производители – сущность, хранящая в себе некоторую информацию о производителях товаров организации.
6. Единицы измерения – единицы, используемые для отражения количества номенклатуры в разрезе, например, штук или упаковок.
7. Поставщики – подобно производителям, Пиши-стирай видит необходимость в хранении некоторой информации о поставщиках своих товаров.
8. Пункты выдачи – работая с Пиши-стирай, покупатели имеют возможность выбрать удобный для них адрес выдачи заказа из предложенных в данной сущности.
9. Статусы заказов – установленные маркеры выполнения заказов, упрощающие контроль за их выполнением.
10. Заказы – одна из важнейших сущностей, содержащая в себе всю необходимую информацию по текущему заказу пользователя, в том числе статус заказа, дату доставки, выбранный адрес пункта выдачи, дату оформления заказа, оформившего заказ клиента и код для получения заказа.
11. Продукты в заказах – информация о составе заказов, т.е конкретных продуктах, его наполняющих, и их количестве.

В результате вышеуказанного анализа предметной области была составлена ER-диаграмма, отражающая все имеющиеся сущности и связи между ними. Ознакомиться с ней можно на рисунке 1.

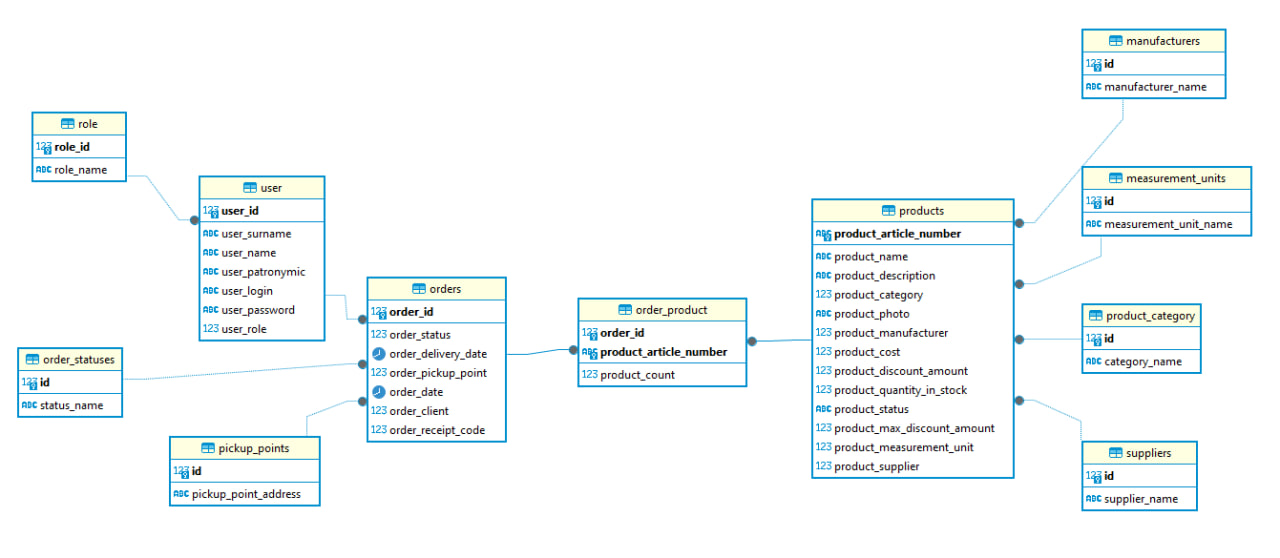


Рисунок 1 – ER-диаграмма предметной области

* 1. Разработка базы данных

Подготовленная выше ER-диаграмма стала основой для восстановления из предложенного в задании скрипта средствами языка PostgreSQL базы данных, листинг которой можно найти в Приложении А таблице А.1.

Далее, база данных была заполнена значениями из ресурсов, приложенных к заданию, в результате чего таблицы приняли вид, отображенный на рисунках 2-7.

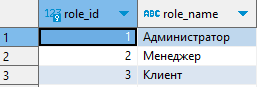


Рисунок 2 – Результат заполнения таблицы role

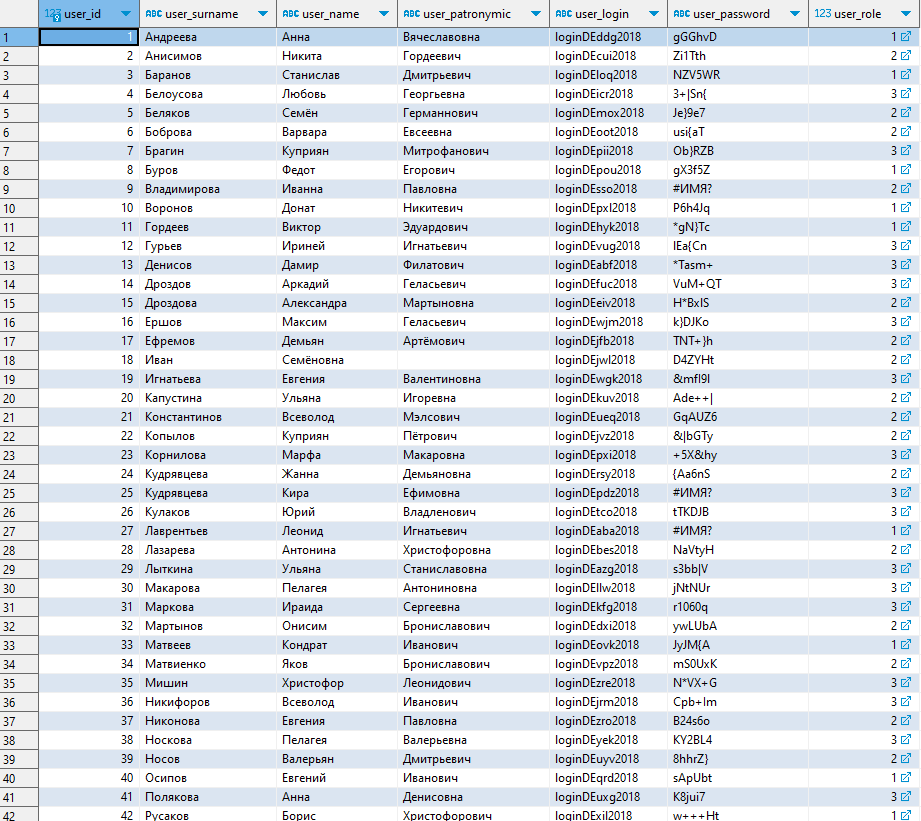


Рисунок 3 – Результат заполнения таблицы user

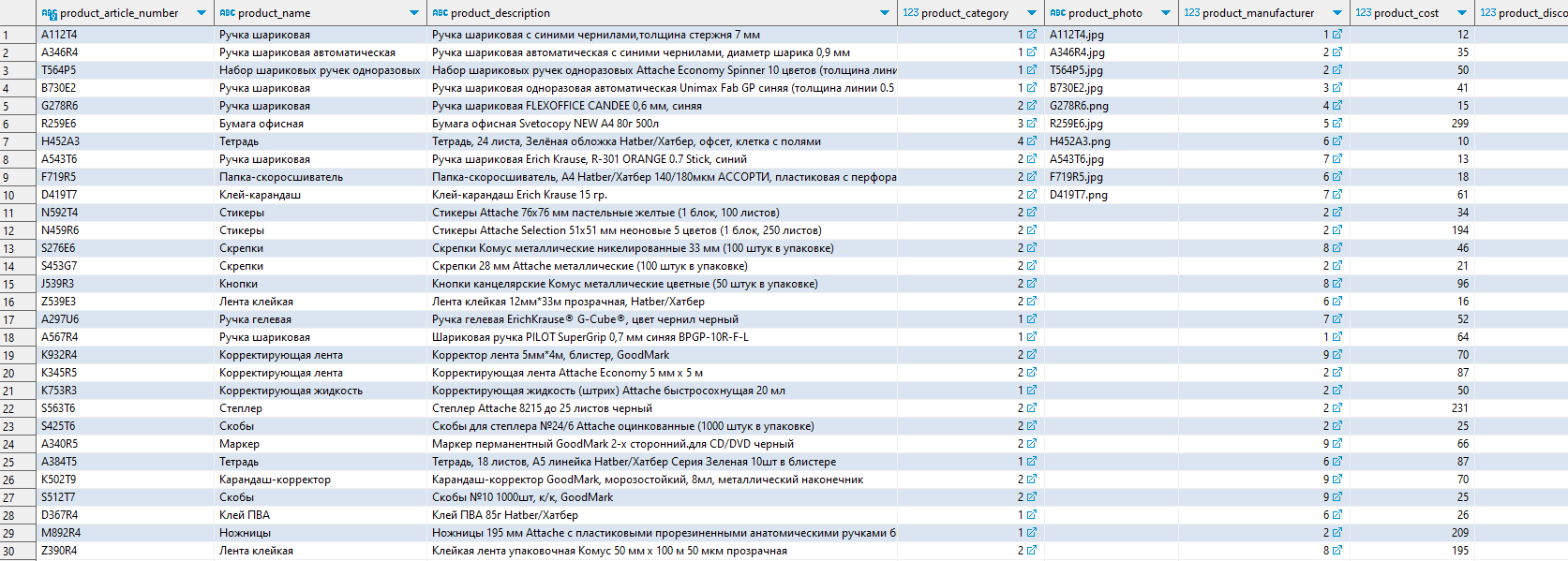


Рисунок 4 – Результат заполнения таблицы products

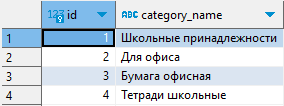


Рисунок 5 – Результат заполнения таблицы product\_category

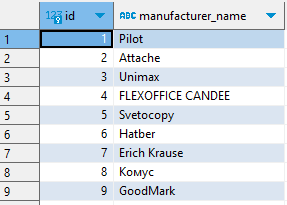


Рисунок 6 – Результат заполнения таблицы manufacturers

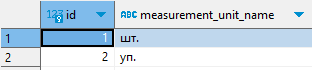


Рисунок 7 – Результат заполнения таблицы measurement\_units



Рисунок 8 – Результат заполнения таблицы suppliers



Рисунок 9 – Результат заполнения таблицы pickup\_points



Рисунок 10 – Результат заполнения таблицы order\_statuses

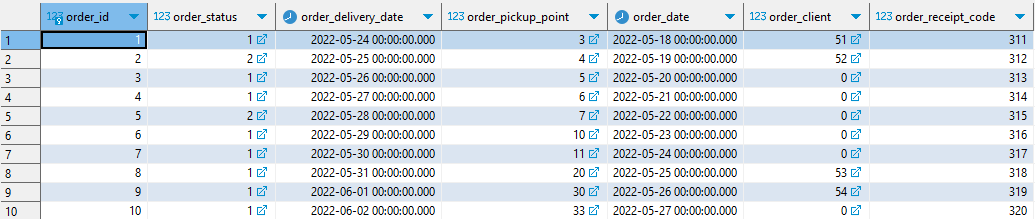


Рисунок 11 – Результат заполнения таблицы orders

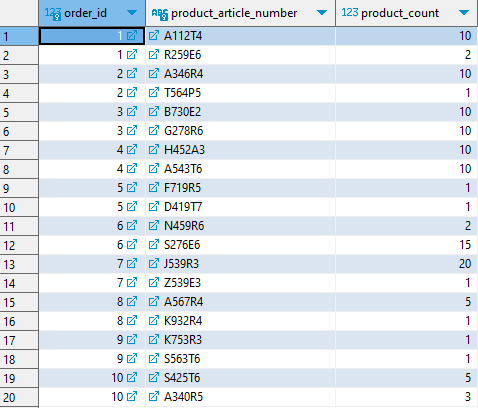


Рисунок 12 – Результат заполнения таблицы order\_product

1. РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ
   1. Основа приложения

Своеобразным «фундаментом» разрабатываемого приложения является паттерн разработки, определяющий, в общем случае, логику взаимодействия компонентов внутри программы. В нашем случае за основу берется MVVM – паттерн разработки, позволяющий разделить приложение на три функциональные части: Model – основная логика программы (работа с данными, вычисления, запросы и так далее); View – вид или представление (пользовательский интерфейс); ViewModel – модель представления, которая служит прослойкой между View и Model.

Таким образом, принимая во внимание предполагаемую многостраничность разрабатываемого приложения, в дальнейшем каждый из экранов будет представлять собой отдельный View (Avalonia UserControl, в частности) с привязанным контекстом (DataContext) данных, содержащих в себе всю логику, реализуемую через пользовательский интерфейс, сформированный XAML-разметкой, предлагаемой фреймворком Avalonia. Контекст у каждой страницы (ViewModel) предполагается свой собственный, осуществлённый при помощи классов языка C#, наследуемых от MVVM-фреймворка Reactive UI и содержащий в себе некоторую базовую логику, необходимую для реализации пользовательских функций, а также обеспечения взаимодействия с базой данных.

Последний уровень приложения – Model – совокупность классов, используемых совместно всеми ViewModel приложения, отражающих и переносящих структуру сущностей базы данных в приложении типа Avalonia .NET MVVM App.

* 1. Подключение к базе данных

Отдельно стоит рассмотреть подключение приложения к базе данных и методику дальнейшего с ней взаимодействия.

Так, для реализации соединения с БД PostgreSQL в первую очередь необходима установка двух NuGet пакетов, представленных на рисунках 8-9.

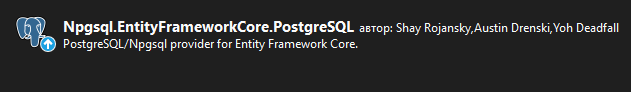


Рисунок 8 – Пакет Npgsql.EntityFrameworkCore.PostgreSQL

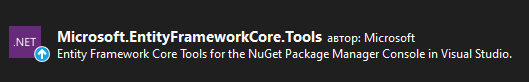


Рисунок 9 – Пакет Microsoft.EntityFrameworkCore.Tools

Далее, воспользовавшись консолью диспетчера пакетов, необходимо ввести следующую команду:

Scaffold-DbContext "Host=edu.pg.ngknn.local;Port=5432;Database=CftBase;Username=43P;Password=444444" Npgsql.EntityFrameworkCore.PostgreSQL -o Models

Первое выражение – строка подключения к базе данных, которая была заполнена реальными значениями созданной БД.

Второе выражение – провайдер, обеспечивающий соединение.

Третье и четвертое выражение – указание папки, в которую будут помещены автоматически генерируемые модели БД.

После выполнения команд в проекте будут созданы классы, соответствующие таблицам из БД, а также еще один класс, который и будет отвечать за подключение БД (так называемый контекст или Context – в этом классе находится в т.ч. и строки подключения к базе данных).

Дальнейшее обращение к БД и ее объектам будет возможно посредством создания объекта класса подключения. В нашем случае такой объект создан в классе MainWindowViewModel и доступен для обращения из любой другой ViewModel. Код создания объекта приведен на рисунке 10.

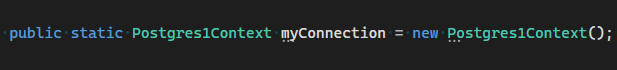


Рисунок 10 – Создание объекта подключения БД

Напрямую через этот объект в дальнейшем и будет организовано взаимодействие с базой данных – с моделями ее таблиц, на практике.

* 1. Обеспечение навигации в приложении

Стоит упомянуть, что как таковые окна приложения не сменяются – меняется лишь наполнение основной, поэтому навигация между ними имеет свою специфику. Рассмотрим подробнее.

Так, главной страницей приложения является MainWindow, содержащая в себе UserControl с привязанным (свойство Binding) к нему объектом соответствующего типа из MainWindowViewModel, значение которого мы меняем в нужное нам время, тем самым сменяя и страницы. Так, создание всех необходимых объектов для последующего к ним обращения с целью навигации можно увидеть на рисунке 11. На нем также видно, что начальным значением UserControl – запущенной вместе с приложением страницей – будет AuthView, т.е. страница с авторизацией.

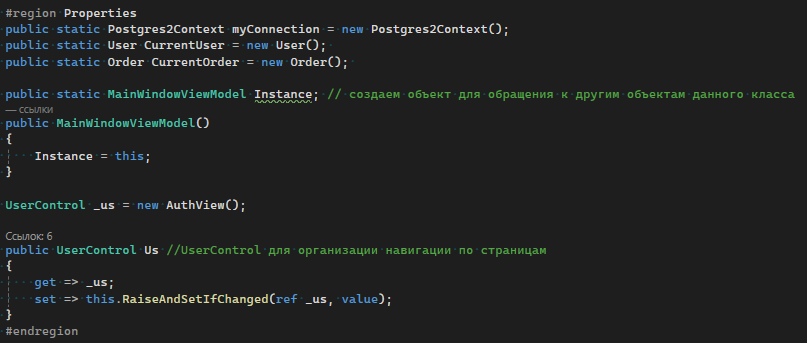


Рисунок 11 – Создание объектов для навигации по страницам

* 1. Авторизация в системе

Как было упомянуто выше, первой загружаемой страницей в приложении является окно авторизации. На нем пользователю предлагается ввести логин и пароль или авторизоваться с ролью гостя. На этом этапе данные о текущем пользователе (при успешной авторизации, естественно – в обратном случае пользователю сообщат об ошибке) сохраняются в главном классе программы и впоследствии будут использованы оттуда.

Окно авторизации представлено на рисунке 12.

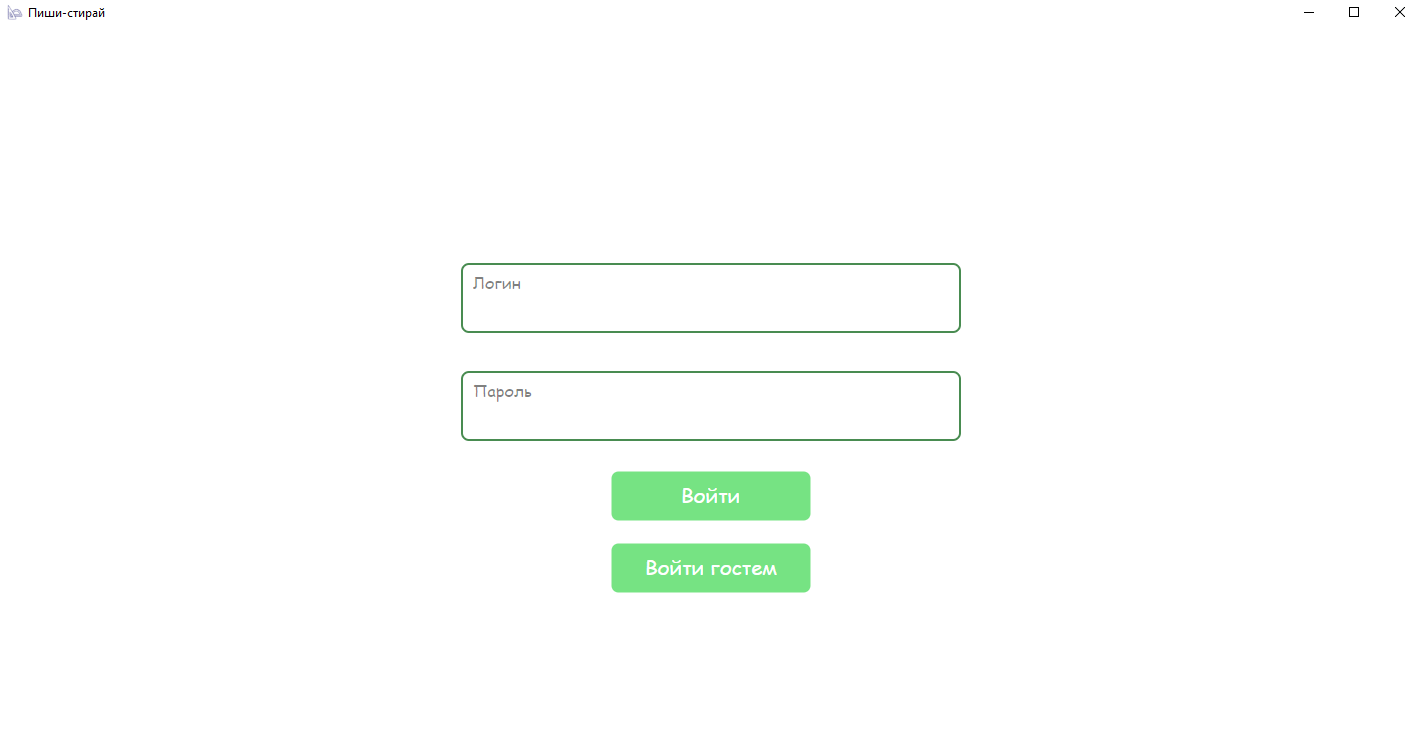


Рисунок 12 – Окно авторизации

* 1. Работа с товарами и информацией о них

Для обеспечения учета товаров используется список с информацией по каждому из них, представленный на странице ProductsView.

Вывод данных о товарах организуется при помощи элемента пользовательского интерфейса ListBox, а источником данных для него – лист типа данных Products, инициализированный в конструкторе соответствующей странице ViewModel, что представлено на рисунке 12.

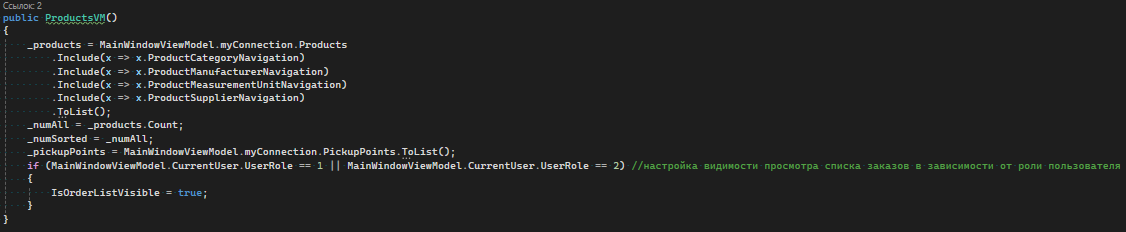


Рисунок 13 – Получение данных о партнерах в лист

Полученные из соединения с БД данные структурируются также при помощи различных элементов UI и привязываются к ним; добавляется заголовок страницы и логотип предприятия Пиши-стирай.

Так как среди информации о товаре у некоторых из них присутствует картинка, был также реализован конвертер изображений из их формы в базе данных к форме, требуемой платформой. Метод также обрабатывает отсутствие картинки у товара и заменяет ее стандартной иконкой товара. Увидеть реализованный конвертер изображений можно на рисунке 14.

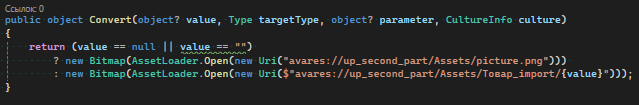


Рисунок 14 – Конвертер изображений

Помимо вышеуказанного, в приложении производится расчет стоимости товара с учетом скидки (помимо имеющейся стандартной цены) посредством создания частичного класса Product и свойства (представленного на рисунке 14), производящего расчет на основании условий из задания.



Рисунок 15 – Сниженная стоимость товара с учетом скидки

Также, согласно условию задания, если скидка товара составляет более 15%, то она подсвечивается указанным цветом. За цвет также отвечает свойство все в том же частичном классе, которое можно увидеть на рисунке 16.

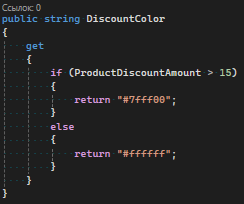


Рисунок 16 – Свойство изменения цвета скидки по условию

В верхнем правом углу экрана располагается надпись с инициалами пользователя, по нажатию на которую открывается меню с возможностью выхода из профиля.

Также в верхней части экрана располагается информация о количестве товаров в базе и отображенных в данный момент на экране.

Для обеспечения комфортной работы с ассортиментом товаров программа также предоставляет возможность поиска товаров по наименованию, фильтрации по диапазону скидок и сортировки по стоимости товара. Метод, реализующий все эти возможности, представлен в приложении А в таблице А.2.

В результате вышеперечисленных действий страница приобретает вид, представленный на рисунке 17.

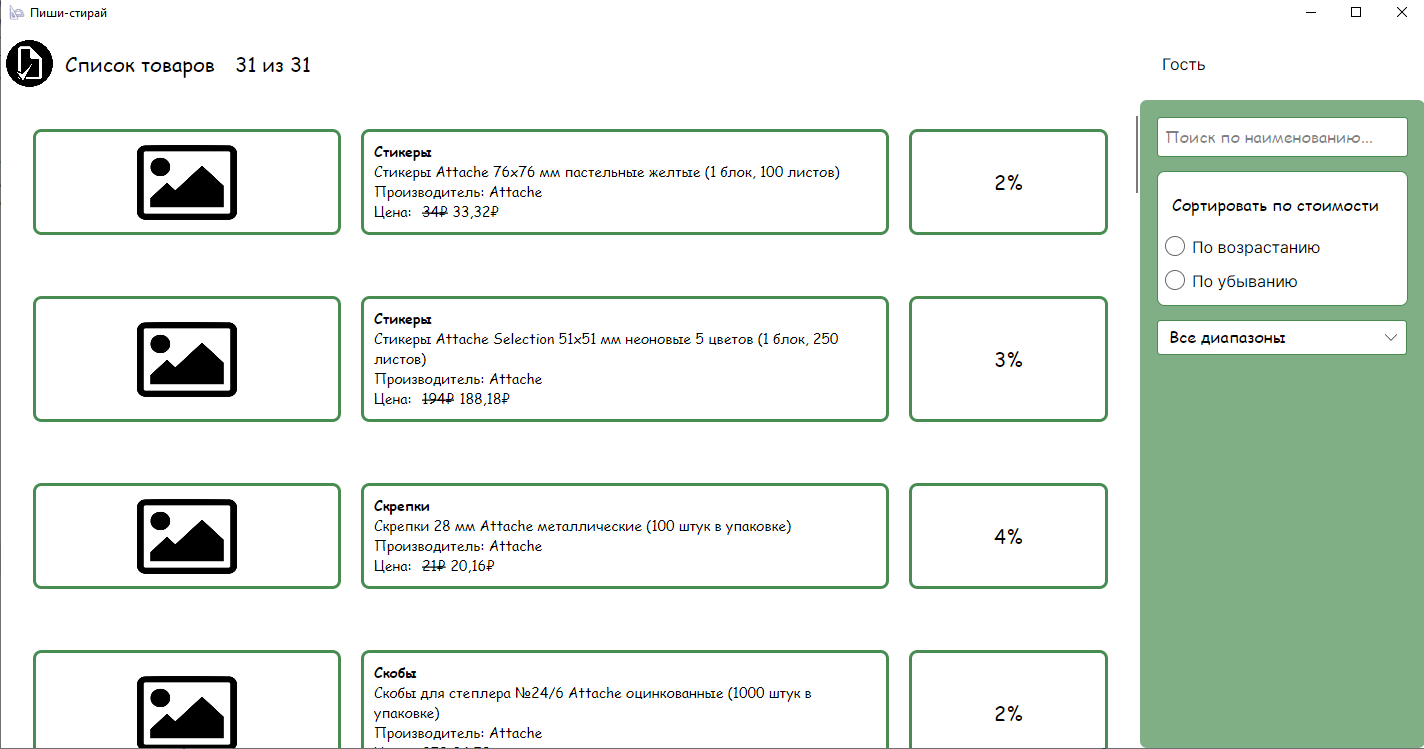


Рисунок 17 – Страница просмотра ассортимента товаров

1. СОПРОВОЖДЕНИЕ И ОБСЛУЖИВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ КОМПЬЮТЕРНЫХ СИСТЕМ
   1. Формирование заказа

Для осуществления функционала формирования заказа первым делом была создана соответствующая страница (OrderView), к которой был привязан предыдущий класс с логикой. Так как по условию задания страница с формированием заказа должна открываться в отдельном модальном окне, был разработан специальный метод (рисунок 18), реализующий эту возможность. Кнопка, вызывающая метод ниже, появляется только при наличии хотя бы 1 товара в заказе благодаря отслеживающему состояние ее видимости свойству.

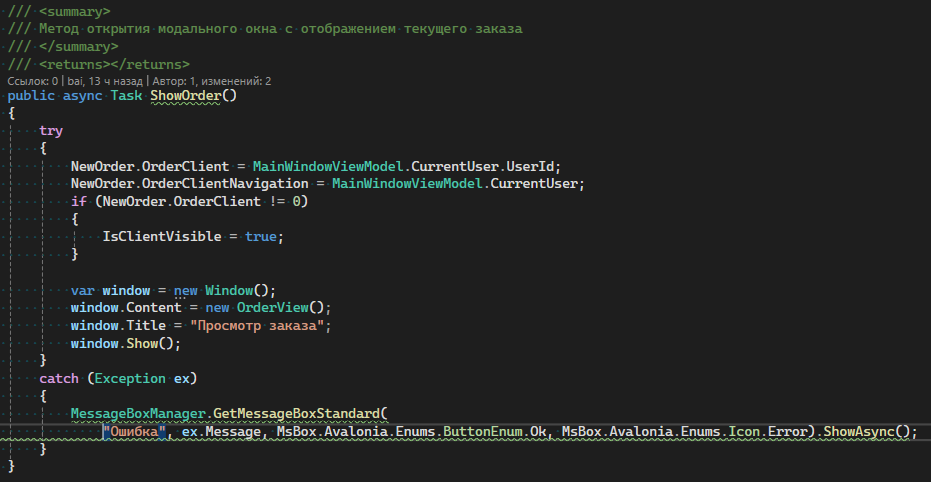


Рисунок 18 – Метод открытия модального окна с формированием заказа

Кроме того, были созданы два метода (рисунки 18-19), соответственно добавляющие и удаляющие товар в текущий заказ по нажатию на правую кнопку мыши и выбора соответствующего пункта в меню.

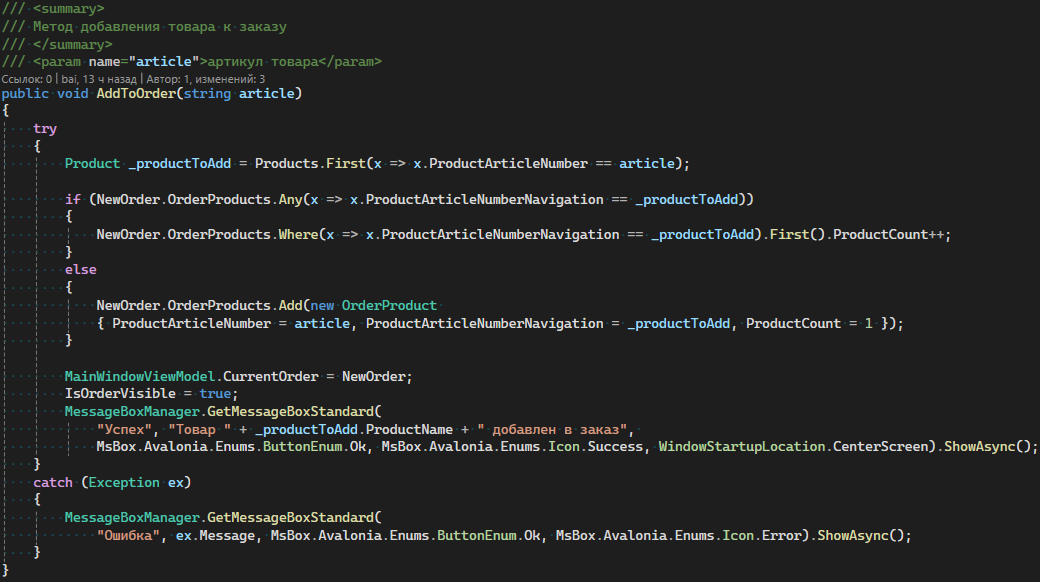


Рисунок 18 – Метод добавления товара в заказ

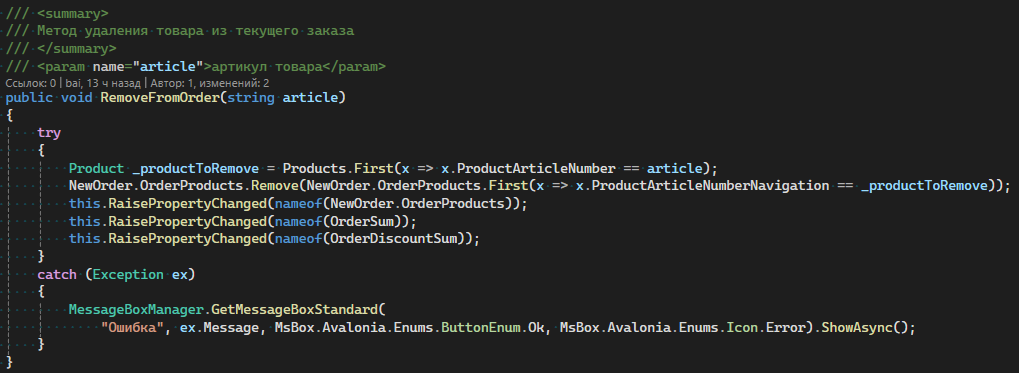


Рисунок 19 – Метод удаления товара из заказа

На этапе просмотра заказа пользователь может выбрать удобный для него пункт выдачи, а также оценить общую сумму, общую скидку заказа и изменить количество конкретного товара в заказе.

На дальнейшем этапе сохранения для заказа формируются дата доставки в зависимости от количества товара на складе, генерируется номер заказа и код получения. Код метода сохранения находится в приложении А в таблице А.3.

После сохранения заказа пользователю предлагается сохранить талон для получения в формате PDF и закрыть окно формирования заказа. Реализующий сохранение в PDF-файл метод можно найти в приложении А в таблице А.4. Пример сохраненного талона представлен ниже на рисунке 20.

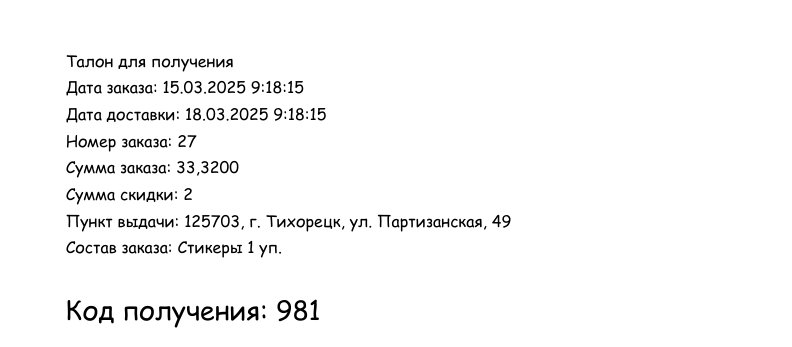


Рисунок 20 – Пример талона на получение заказа

После завершения заказа кнопка с просмотром его содержания пропадает до следующего добавления хотя бы одного товара в заказ.

Примерный внешний вид страницы формирования заказа можно увидеть на рисунке 21.



Рисунок 21 – Страница формирования заказа

* 1. Просмотр и управление списком заказов

При авторизации пользователя с ролью администратора или менеджера приложение отображает еще одну кнопку в верхней части экрана просмотра товаров – Посмотреть список заказов.

По нажатию на нее пользователь переходит на соответствующую страницу, где отображается информация о всех существующих на данный момент заказах: номер заказа, ФИО заказавшего клиента (если это авторизованный пользователь), дата заказа и изменяемая дата доставки, общая скидка и сумма заказа, состав заказа (вся информация о товарах в заказе), а также изменяемый статус заказа. Присутствует также кнопка Сохранить для фиксации в базе данных изменений в заказе.

Для правильного отображения в элементах управления и обмена с БД даты заказа и доставки конвертируются между временными типами данных благодаря написанному (и отображенному на рисунке 22) конвертеру дат.

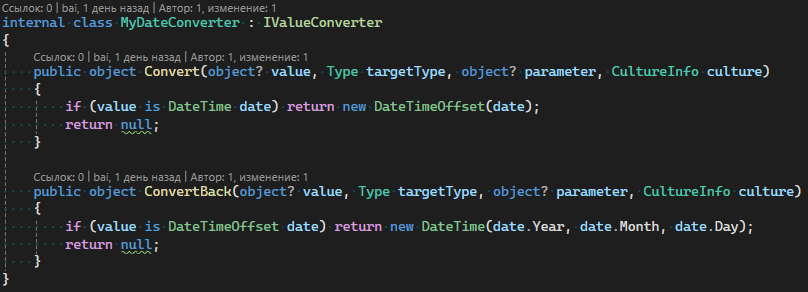


Рисунок 22 – Конвертер дат

Помимо этого, согласно условию задания строка с заказом выделяется особым цветом в зависимости от наличия (и количества) на складе товаров, составляющих заказ. Это обеспечивается благодаря свойству в частичном классе Order, проверяющем количество на складе каждого товара. Увидеть это свойство можно на рисунке 23.

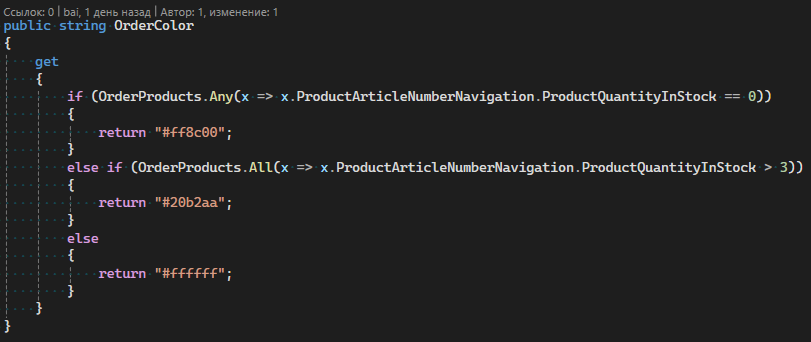


Рисунок 23 – Свойство выделения заказа цветом

Для обеспечения более комфортной работы с большим массивом заказов также обеспечены сортировка по сумме заказа и фильтрация по предложенным в заказе диапазонам цены. Метод сортировки и фильтрации идентичен тому, что был на странице со списком продуктов.

По кнопке Назад пользователь также имеет возможность вернуться на предыдущую страницу.

Итоговый внешний вид страницы просмотра списка заказов можно увидеть на рисунке 24.

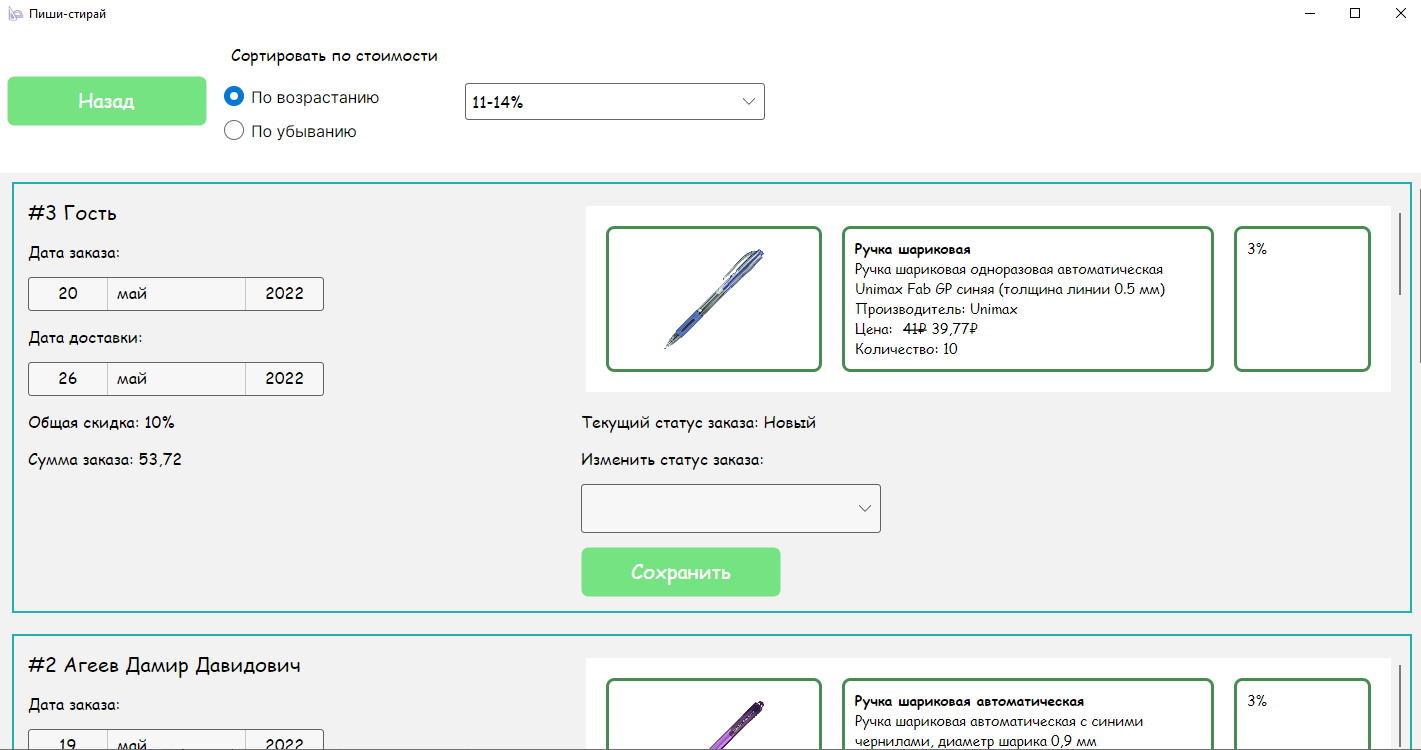


Рисунок 24 – Страница просмотра списка заказов

* 1. Тестирование процедуры добавления товара

Также, согласно условию задания, необходимо было провести ручное тестирование процедуры добавления товара администратором. Составленные тест-кейсы представлены в таблицах 1-5.

Таблица 1 – Тестовый пример №1

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_FUN\_1 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Успешное добавление товара с валидными данными в приложении Пиши-стирай |

Продолжение таблицы 1

|  |  |
| --- | --- |
| **Краткое изложение теста** | Проверка добавления товара с корректными обязательными и необязательными полями администратором в приложении Пиши-стирай |
| **Этапы теста** | 1. Авторизоваться как администратор 2. Перейти в раздел «Добавить товар» 3. Ввести валидные данные: артикул, наименование, описание, категорию товара, изображение, производителя, стоимость, размер скидки, количество на складе, статус продукта, максимальный объем скидки, единицы измерения и поставщика 4. Нажать «Сохранить» |
| **Тестовые данные** | Артикул = «АРТРУЧ1», наименование = «Ручка синяя гелевая», описание = «Ручка синяя гелевая стирающаяся с колпачком и запасными стержнями», категория товара = 1, изображение = «», производителя = 1, стоимость = 70, размер скидки = 5, количество на складе = 30, статус продукта = «», максимальный объем скидки = 20, единицы измерения = 1, поставщик = 1 |
| **Ожидаемый результат** | Товар добавлен, отображается в списке товаров с корректными данными. Сообщение: «Товар успешно сохранен». |
| **Фактический результат** | Товар добавлен, отображается в списке товаров с корректными данными. Сообщение: «Товар успешно сохранен». |
| **Статус** | Зачет |
| **Предварительное условие** | Программа запущена |
| **Постусловие** | Программа продолжает работу, открыта страница со списком товаров |
| **Примечания/комментарии** | - |

Таблица 2 – Тестовый пример №2

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_FUN\_2 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Добавление товара с пустым обязательным полем Наименование в приложении Пиши-стирай |
| **Краткое изложение теста** | Проверка обработки наличия пустого поля Наименование при добавлении товара администратором в приложении Пиши-стирай |

Продолжение таблицы 2

|  |  |
| --- | --- |
| **Этапы теста** | 1. Авторизоваться как администратор 2. Перейти в раздел «Добавить товар» 3. Ввести все валидные данные, кроме поля Наименование 4. Нажать «Сохранить» |
| **Тестовые данные** | Артикул = «АРТКАР2», наименование = «», описание = «Карандаш простой стирающийся с колпачком и запасными стержнями», категория товара = 1, изображение = «», производителя = 1, стоимость = 40, размер скидки = 3, количество на складе = 30, статус продукта = «», максимальный объем скидки = 10, единицы измерения = 1, поставщик = 2 |
| **Ожидаемый результат** | Программа выдает ошибку: «Поле «Наименование» обязательно для заполнения». Товар не сохранен |
| **Фактический результат** | Программа выдает ошибку: «Поле «Наименование» обязательно для заполнения». Товар не сохранен |
| **Статус** | Зачет |
| **Предварительное условие** | Программа запущена |
| **Постусловие** | Программа продолжает работу, открыта страница с добавлением товара |
| **Примечания/комментарии** | - |

Таблица 3 – Тестовый пример №3

|  |  |
| --- | --- |
| Тестовый пример **#** | TC\_FUN\_3 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Добавление товара с отрицательным значением поля Стоимость в приложении Пиши-стирай |
| **Краткое изложение теста** | Проверка обработки наличия отрицательного значения в поле Стоимость при добавлении товара администратором в приложении Пиши-стирай |
| **Этапы теста** | 1. Авторизоваться как администратор 2. Перейти в раздел «Добавить товар» 3. Ввести все валидные данные, кроме поля Стоимость 4. Нажать «Сохранить» |

Продолжение таблицы 3

|  |  |
| --- | --- |
| **Тестовые данные** | Артикул = «АРТРУЧ3», наименование = «Ручка черная шариковая», описание = «Ручка шариковая черная с колпачком и запасными стержнями», категория товара = 1, изображение = «», производителя = 2, стоимость = -1, размер скидки = 7, количество на складе = 45, статус продукта = «», максимальный объем скидки = 15, единицы измерения = 1, поставщик = 1 |
| **Ожидаемый результат** | Программа выдает ошибку: «Поле «Стоимость» не может быть отрицательным». Товар не сохранен |
| **Фактический результат** | Программа выдает ошибку: «Поле «Стоимость» не может быть отрицательным». Товар не сохранен |
| **Статус** | Зачет |
| **Предварительное условие** | Программа запущена |
| **Постусловие** | Программа продолжает работу, открыта страница с добавлением товара |
| **Примечания/комментарии** | - |

Таблица 4 – Тестовый пример №4

|  |  |
| --- | --- |
| Тестовый пример **#** | TC\_FUN\_4 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Добавление товара с уже имеющимся в базе данных значением поля Артикул в приложении Пиши-стирай |
| **Краткое изложение теста** | Проверка обработки наличия уже имеющегося в базе данных значения поля Артикул при добавлении товара администратором в приложении Пиши-стирай |
| **Этапы теста** | 1. Авторизоваться как администратор 2. Перейти в раздел «Добавить товар» 3. Ввести все валидные данные, кроме поля Артикул 4. Нажать «Сохранить» |

Продолжение таблицы 4

|  |  |
| --- | --- |
| **Тестовые данные** | Артикул = «АРТРУЧ3», наименование = «Ручка красная шариковая», описание = «Ручка гелевая красная с колпачком и запасными стержнями», категория товара = 1, изображение = «», производителя = 2, стоимость = 30, размер скидки = 9, количество на складе = 6, статус продукта = «», максимальный объем скидки = 18, единицы измерения = 1, поставщик = 2 |
| **Ожидаемый результат** | Программа выдает ошибку: «Данный артикул уже используется». Товар не сохранен |
| **Фактический результат** | Программа выдает ошибку: «Данный артикул уже используется». Товар не сохранен |
| **Статус** | Зачет |
| **Предварительное условие** | Программа запущена |
| **Постусловие** | Программа продолжает работу, открыта страница с добавлением товара |
| **Примечания/комментарии** | - |

Таблица 5 – Тестовый пример №5

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_FUN\_5 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Добавление товара с граничным значением поля Стоимость в приложении Пиши-стирай |
| **Краткое изложение теста** | Проверка обработки наличия максимальным значением поля Стоимость при добавлении товара администратором в приложении Пиши-стирай |
| **Этапы теста** | 1. Авторизоваться как администратор 2. Перейти в раздел «Добавить товар» 3. Ввести все валидные данные, кроме поля Стоимость 4. Нажать «Сохранить» |
| **Тестовые данные** | Артикул = «АРТРУЧ5», наименование = «Ручка синяя шариковая», описание = «Ручка синяя шариковая с колпачком», категория товара = 1, изображение = «», производителя = 1, стоимость = 9999999999999999.9999, размер скидки = 1, количество на складе = 6, статус продукта = «», максимальный объем скидки = 3, единицы измерения = 2, поставщик = 1 |

Продолжение таблицы 5

|  |  |
| --- | --- |
| **Ожидаемый результат** | Товар добавлен, отображается в списке товаров с корректной стоимостью. Сообщение: «Товар успешно сохранен». |
| **Фактический результат** | Товар добавлен, отображается в списке товаров с корректной стоимостью. Сообщение: «Товар успешно сохранен». |
| **Статус** | Зачет |
| **Предварительное условие** | Программа запущена |
| **Постусловие** | Программа продолжает работу, открыта страница с добавлением товара |
| **Примечания/комментарии** | - |

1. ОСУЩЕСТВЛЕНИЕ ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ
   1. Разработка DLL-библиотеки

С целью обеспечения оптимального графика работы сотрудников методом получения списка свободных временных интервалов в графике сотрудника, был разработан метод в формате DLL, принимающий список занятых промежутков времени, минимальное необходимое время для работы менеджера, рабочий день сотрудника, и возвращающий список подходящих свободных временных промежутков – массив строк.

В методе занятые интервалы сортируются по времени начала, чтобы алгоритм корректно обрабатывал их в хронологическом порядке, после чего для каждого свободного промежутка между текущим временем и началом следующего события генерируются все возможные интервалы заданной длины. Если текущее время попадает в занятый интервал, оно автоматически сдвигается на его конец. После обработки всех событий проверяется оставшееся время до конца рабочего дня.

Метод также осуществляет проверку введенных значений на корректность и соответствие предметной области и обрабатывает возможные ошибки, вызывая в случае неблагоприятного исхода информативное исключение.

Код самого метода приведен в приложении А в таблице А.5.

* 1. Разработка Unit-тестов к библиотеке

Для определения правильности работы метода из разработанной выше DLL было проведено автоматизированное тестирование посредством создания Unit-тестов.

1. 7 простых тестов, включающих следующие методы класса Assert (основного метода тестирования в проекте MS Test): AreEqual (тестовые данные идентичны), IsTrue (тестируемое выражение истинно), IsFalse (тестируемое выражение ложно). Данные тесты охватывают проверку действия основного функционала метода на корректных значениях.
2. 3 сложных теста, включающих метод ThrowsException (в результате применения тестовых данных срабатывает исключение). Данные тесты призваны оценить защищенность метода и корректность обработки ошибок им в случае непредвиденных ситуаций, таких как, например, переполнение типа или нулевой параметр.

Код всех тестов можно увидеть в приложении А в таблице А.6.

ЗАКЛЮЧЕНИЕ

В ходе учебной практики по ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем» были приобретены важные теоретические знания и практические навыки, необходимые для разработки программных модулей. В процессе выполнения практических заданий освоены ключевые этапы создания программного обеспечения, включая проектирование алгоритмов, написание кода на современных языках программирования, тестирование, отладку, оптимизацию и документирование программных продуктов.

Практика позволила закрепить знания об основных этапах разработки программного обеспечения, основных принципах технологии структурного и объектно-ориентированного программирования, способах оптимизации и приемах рефакторинга, основных принципах отладки и тестирования программных продуктов.

Полученные знания и опыт будут полезны для дальнейшего обучения и профессионального развития в сфере разработки программного обеспечения.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Таблица А.1 – Код создания базы данных

|  |
| --- |
| **CREATE** **TABLE** **role** (  role\_id SERIAL **PRIMARY** **KEY**,  role\_name **VARCHAR**(100) **NOT** **NULL**  );  **CREATE** **TABLE** **"user"** (  user\_id SERIAL **PRIMARY** **KEY**,  user\_surname **VARCHAR**(100) **NOT** **NULL**,  user\_name **VARCHAR**(100) **NOT** **NULL**,  user\_patronymic **VARCHAR**(100) **NOT** **NULL**,  user\_login **TEXT** **NOT** **NULL**,  user\_password **TEXT** **NOT** **NULL**,  user\_role **INT** **NOT** **NULL** **REFERENCES** **role**(role\_id)  );  **CREATE** **TABLE** public.product\_category (  id serial4 **NOT** **NULL**,  category\_name **text** **NOT** **NULL**,  **CONSTRAINT** product\_category\_pk **PRIMARY** **KEY** (id)  );  **CREATE** **TABLE** public.measurement\_units (  id serial4 **NOT** **NULL**,  measurement\_unit\_name **text** **NOT** **NULL**,  **CONSTRAINT** measurement\_units\_pk **PRIMARY** **KEY** (id)  );  **CREATE** **TABLE** public.suppliers (  id serial4 **NOT** **NULL**,  supplier\_name **text** **NOT** **NULL**,  **CONSTRAINT** suppliers\_pk **PRIMARY** **KEY** (id)  );  **CREATE** **TABLE** public.manufacturers (  id serial4 **NOT** **NULL**,  manufacturer\_name **text** **NOT** **NULL**,  **CONSTRAINT** manufacturers\_pk **PRIMARY** **KEY** (id)  );  **CREATE** **TABLE** public.products (  product\_article\_number **varchar**(100) **NOT** **NULL**,  product\_name **text** **NOT** **NULL**,  product\_description **text** **NOT** **NULL**,  product\_category serial4 **NOT** **NULL**,  product\_photo **varchar**(100) **NULL**,  product\_manufacturer serial4 **NOT** **NULL**,  product\_cost **decimal**(19, 4) **NOT** **NULL**,  product\_discount\_amount **float4** **NOT** **NULL**,  product\_quantity\_in\_stock **int** **NOT** **NULL**,  product\_status **text** **NULL**,  product\_max\_discount\_amount **float4** **NOT** **NULL**,  product\_measurement\_unit serial4 **NOT** **NULL**,  product\_supplier serial4 **NOT** **NULL**,  **CONSTRAINT** products\_pk **PRIMARY** **KEY** (product\_article\_number),  **CONSTRAINT** products\_manufacturers\_fk **FOREIGN** **KEY** (product\_manufacturer) **REFERENCES** public.manufacturers(id),  **CONSTRAINT** products\_suppliers\_fk **FOREIGN** **KEY** (product\_supplier) **REFERENCES** public.suppliers(id),  **CONSTRAINT** products\_product\_category\_fk **FOREIGN** **KEY** (product\_category) **REFERENCES** public.product\_category(id), |

Продолжение таблицы А.1

|  |
| --- |
| **CONSTRAINT** products\_measurement\_units\_fk **FOREIGN** **KEY** (product\_measurement\_unit) **REFERENCES** public.measurement\_units(id)  );  **CREATE** **TABLE** public.order\_statuses (  id serial4 **NOT** **NULL**,  status\_name **text** **NOT** **NULL**,  **CONSTRAINT** order\_statuses\_pk **PRIMARY** **KEY** (id)  );  **CREATE** **TABLE** public.pickup\_points (  id serial4 **NOT** **NULL**,  pickup\_point\_address **text** **NOT** **NULL**,  **CONSTRAINT** pickup\_points\_pk **PRIMARY** **KEY** (id)  );  **CREATE** **TABLE** public.orders (  order\_id serial4 **NOT** **NULL**,  order\_status serial4 **NOT** **NULL**,  order\_delivery\_date **timestamp** **NOT** **NULL**,  order\_pickup\_point serial4 **NOT** **NULL**,  order\_date **timestamp** **NOT** **NULL**,  order\_client **int4** **DEFAULT** 0 **NOT** **NULL**,  order\_receipt\_code **int2** **NOT** **NULL**,  **CONSTRAINT** newtable\_pk **PRIMARY** **KEY** (order\_id),  **CONSTRAINT** orders\_order\_statuses\_fk **FOREIGN** **KEY** (order\_status) **REFERENCES** public.order\_statuses(id),  **CONSTRAINT** orders\_pickup\_points\_fk **FOREIGN** **KEY** (order\_pickup\_point) **REFERENCES** public.pickup\_points(id),  **CONSTRAINT** orders\_user\_fk **FOREIGN** **KEY** (order\_client) **REFERENCES** public."user"(user\_id)  );  **CREATE** **TABLE** order\_product (  order\_id **INT** **NOT** **NULL** **REFERENCES** **"orders"**(order\_id),  product\_article\_number **VARCHAR**(100) **NOT** **NULL** **REFERENCES** products(product\_article\_number),  product\_count **INT** **NOT** **null**,  **PRIMARY** **KEY** (order\_id, product\_article\_number)  ); |

Таблица А.2 – Метод поиска, сортировки и фильтрации

|  |
| --- |
| /// <summary>  /// Метод фильтрации товаров  /// </summary>  void DoFilter()  {  NotFoundProducts = false;  Products = MainWindowViewModel.myConnection.Products  .Include(x => x.ProductCategoryNavigation)  .Include(x => x.ProductManufacturerNavigation)  .Include(x => x.ProductMeasurementUnitNavigation)  .Include(x => x.ProductSupplierNavigation)  .ToList();  NumSorted = Products.Count;  //поиск по наименованию товара  if (!string.IsNullOrWhiteSpace(\_searchStr))  {  Products = Products.Where(x => x.ProductName.ToLower().Contains(\_searchStr.ToLower())).ToList();  NumSorted = Products.Count();  } |

Продолжение таблицы А.2

|  |
| --- |
| //фильтрация по скидке  if (\_selectedDiscount != null)  {  if (\_selectedDiscount == \_discounts[0])  {  Products = Products.Where(x => x.ProductDiscountAmount < 10 || (x.ProductDiscountAmount < 15 && x.ProductDiscountAmount >= 10) || x.ProductDiscountAmount >= 15).ToList();  NumSorted = Products.Count();  }  else if (\_selectedDiscount == \_discounts[1])  {  Products = Products.Where(x => x.ProductDiscountAmount < 10).ToList();  NumSorted = Products.Count();  }  else if (\_selectedDiscount == \_discounts[2])  {  Products = Products.Where(x => x.ProductDiscountAmount < 15 && x.ProductDiscountAmount >= 10).ToList();  NumSorted = Products.Count();  }  else  {  Products = Products.Where(x => x.ProductDiscountAmount >= 15).ToList();  NumSorted = Products.Count();  }  }  //сортировка по стоимости  if (\_sortUpCost)  {  \_sortDownCost = false;  Products = Products.OrderBy(x => x.ProductCost).ToList();  NumSorted = Products.Count();  }  else if (\_sortDownCost)  {  \_sortUpCost = false;  Products = Products.OrderByDescending(x => x.ProductCost).ToList();  NumSorted = Products.Count();  }  NotFoundProducts = NumSorted.Equals(0);  } |

Таблица А.3 – Метод сохранения заказа

|  |
| --- |
| /// <summary>  /// Метод для добавления заказа в базу данных  /// </summary>  public void SaveChangesInOrder() |

Продолжение таблицы А.3

|  |
| --- |
| {  try  {  if (NewOrder.OrderPickupPointNavigation is null)  {  MessageBoxManager.GetMessageBoxStandard("Внимание", "Выберите пункт выдачи заказа", MsBox.Avalonia.Enums.ButtonEnum.Ok, MsBox.Avalonia.Enums.Icon.Warning, WindowStartupLocation.CenterScreen).ShowAsync();  return;  }  Random \_rnd = new Random();  NewOrder.OrderDate = DateTime.Now;  NewOrder.OrderStatus = 1;  NewOrder.OrderReceiptCode = (short)\_rnd.Next(100, 1000);  NewOrder.OrderPickupPoint = NewOrder.OrderPickupPointNavigation.Id;  if (NewOrder.OrderProducts.Any(x => x.ProductArticleNumberNavigation.ProductQuantityInStock < 3))  {  NewOrder.OrderDeliveryDate = DateTime.Now.AddDays(6);  }  else  {  NewOrder.OrderDeliveryDate = DateTime.Now.AddDays(3);  } |

Продолжение таблицы А.3

|  |
| --- |
| if (NewOrder.OrderId == 0)  {  MainWindowViewModel.myConnection.Orders.Add(NewOrder);  MainWindowViewModel.myConnection.OrderProducts.AddRange(NewOrder.OrderProducts);  }  MainWindowViewModel.myConnection.SaveChanges();  NewOrder = new Order();  IsOrderVisible = false;  MainWindowViewModel.Instance.Us = new ProductsView();  IsPdfEnable = true;  MessageBoxManager.GetMessageBoxStandard("Успех", "Заказ успешно оформлен! Вы можете получить талон в формате PDF и закрыть окно", MsBox.Avalonia.Enums.ButtonEnum.Ok, MsBox.Avalonia.Enums.Icon.Success, WindowStartupLocation.CenterScreen).ShowAsync();  }  catch (Exception ex)  {  MessageBoxManager.GetMessageBoxStandard("Ошибка", ex.Message, MsBox.Avalonia.Enums.ButtonEnum.Ok, MsBox.Avalonia.Enums.Icon.Error).ShowAsync();  }  } |

Таблица А.4 – Метод генерации талона в формате PDF

|  |
| --- |
| private async void GeneratePdf(object sender, Avalonia.Interactivity.RoutedEventArgs e)  {  try  {  Order order = MainWindowViewModel.myConnection.Orders.Where(x => x.OrderId == MainWindowViewModel.myConnection.Orders.Max(x => x.OrderId)).First();  var parentWindow = this.VisualRoot as Window;  // Создаем PDF-документ |

Продолжение таблицы А.4

|  |
| --- |
| PdfSharpCore.Pdf.PdfDocument document = new PdfSharpCore.Pdf.PdfDocument();  PdfSharpCore.Pdf.PdfPage page = document.AddPage();  XGraphics gfx = XGraphics.FromPdfPage(page);  XFont font = new XFont("Comic Sans MS", 12);  XFont font1 = new XFont("Comic Sans MS", 20);  // Добавляем данные  string products = string.Join(", ", order.OrderProducts.Select(w => w.ProductArticleNumberNavigation.ProductName + " " + w.ProductCount.ToString() + " " + w.ProductArticleNumberNavigation.ProductMeasurementUnitNavigation.MeasurementUnitName));  gfx.DrawString("Талон для получения", font, XBrushes.Black, new XPoint(50, 50));  gfx.DrawString($"Дата заказа: {order.OrderDate}", font, XBrushes.Black, new XPoint(50, 70));  gfx.DrawString($"Дата доставки: {order.OrderDeliveryDate}", font, XBrushes.Black, new XPoint(50, 90));  gfx.DrawString($"Номер заказа: {order.OrderId}", font, XBrushes.Black, new XPoint(50, 110));  gfx.DrawString($"Сумма заказа: {order.OrderSum}", font, XBrushes.Black, new XPoint(50, 130));  gfx.DrawString($"Сумма скидки: {order.OrderDiscountSum}", font, XBrushes.Black, new XPoint(50, 150));  gfx.DrawString($"Пункт выдачи: {order.OrderPickupPointNavigation.PickupPointAddress}", font, XBrushes.Black, new XPoint(50, 170));  gfx.DrawString($"Состав заказа: {products}", font, XBrushes.Black, new XPoint(50, 190));  gfx.DrawString($"Код получения: {order.OrderReceiptCode}", font1, XBrushes.Black, new XPoint(50, 240));  // Диалог сохранения файла  var saveDialog = new SaveFileDialog  {  Title = "Сохранить PDF",  DefaultExtension = "pdf",  Filters = { new FileDialogFilter { Name = "PDF-файлы", Extensions = { "pdf" } } },  InitialFileName = "Талон.pdf"  };    var filePath = await saveDialog.ShowAsync(parentWindow);  if (filePath != null)  {  document.Save(filePath);  document.Close();  }  }  catch (Exception ex)  {  MessageBoxManager.GetMessageBoxStandard("Ошибка", ex.Message, MsBox.Avalonia.Enums.ButtonEnum.Ok, MsBox.Avalonia.Enums.Icon.Error).ShowAsync();  }  } |

Таблица А.5 – Метод расчета списка подходящих свободных временных промежутков в графике работника

|  |
| --- |
| /// <summary>  /// Метод расчета списка свободных временных интервалов в графике сотрудника  /// </summary>  /// <param name="startTimes">массив времени начала рабочего промежутка</param>  /// <param name="durations">массив длительности рабочих промежутков</param>  /// <param name="beginWorkingTime">время начала рабочего дня</param>  /// <param name="endWorkingTime">время окончания рабочего дня</param>  /// <param name="consultationTime">минимальное необходимое время для работы менеджера</param>  /// <returns>список (массив строк) подходящих свободных временных промежутков</returns>  /// <exception cref="Exception"></exception>  public string[] AvailablePeriods(TimeSpan[] startTimes, int[] durations, TimeSpan beginWorkingTime, TimeSpan endWorkingTime, int consultationTime)  {  if (startTimes.Length != durations.Length)  throw new Exception("Каждому элементу массива startTimes должен соответствовать элемент массива durations");  if (consultationTime < 1)  throw new Exception("Время консультации должно быть натуральным числом");  List<string> \_freePeriods = new();  DateTime \_timeNow = DateTime.Today.Add(beginWorkingTime);  DateTime \_endWorkingTime = DateTime.Today.Add(endWorkingTime);  Array.Sort(startTimes, durations, Comparer<TimeSpan>.Create((a, b) => a.CompareTo(b)));  int \_counter = 0;  while (\_timeNow <= \_endWorkingTime.AddMinutes(-consultationTime))  {  DateTime \_startOfWork;  DateTime \_endOfWork;  if (\_counter < startTimes.Length)  {  \_startOfWork = DateTime.Today.Add(startTimes[\_counter]);  \_endOfWork = \_startOfWork.AddMinutes(durations[\_counter]);  }  else  {  \_startOfWork = \_endWorkingTime;  \_endOfWork = \_endWorkingTime;  }  DateTime \_freeTimeEnd = \_startOfWork;  while (\_timeNow.AddMinutes(consultationTime) <= \_freeTimeEnd)  {  \_freePeriods.Add($"{\_timeNow:HH:mm}-{\_timeNow.AddMinutes(consultationTime):HH:mm}");  \_timeNow = \_timeNow.AddMinutes(consultationTime);  }  if (\_counter < startTimes.Length)  {  if (\_timeNow < \_endOfWork)  \_timeNow = \_endOfWork;  \_counter++;  }  }  return \_freePeriods.ToArray();  } |

Таблица А.6 – Unit-тесты к библиотеке SF2022User{NN}Lib

|  |
| --- |
| using SF2022User\_NN\_Lib;  namespace TestAvailablePeriods  {  [TestClass]  public class UnitTestAvailablePeriods  {  Calculations \_calculations = new Calculations();  [TestMethod]  public void Test\_NoBusyIntervals()  {  TimeSpan[] startTimes = Array.Empty<TimeSpan>();  int[] durations = Array.Empty<int>();  TimeSpan beginWorkingTime = new TimeSpan(8, 0, 0);  TimeSpan endWorkingTime = new TimeSpan(18, 0, 0);  int consultationTime = 30;  string[] result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  Assert.IsTrue(result.Contains("08:00-08:30"));  }  [TestMethod]  public void Test\_FreeIntervalAtEndOfDay()  {  TimeSpan[] startTimes = { new TimeSpan(8, 0, 0) };  int[] durations = { 9 \* 60 };  TimeSpan beginWorkingTime = new TimeSpan(8, 0, 0);  TimeSpan endWorkingTime = new TimeSpan(18, 0, 0);  int consultationTime = 30;  string[] result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  Assert.AreEqual(2, result.Length);  }  [TestMethod]  public void Test\_NoFreeIntervalsBetweenEvents()  {  TimeSpan[] startTimes = { new TimeSpan(10, 0, 0), new TimeSpan(10, 30, 0) };  int[] durations = { 30, 30 };  TimeSpan beginWorkingTime = new TimeSpan(8, 0, 0);  TimeSpan endWorkingTime = new TimeSpan(18, 0, 0);  int consultationTime = 30;  string[] result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  Assert.IsFalse(result.Contains("10:00-10:30"));  }  [TestMethod]  public void Test\_AllDayBusy()  {  TimeSpan[] startTimes = { new TimeSpan(8, 0, 0) }; |

Продолжение таблицы А.6

|  |
| --- |
| int[] durations = { 10 \* 60 };  TimeSpan beginWorkingTime = new TimeSpan(8, 0, 0);  TimeSpan endWorkingTime = new TimeSpan(18, 0, 0);  int consultationTime = 30;  string[] result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  Assert.AreEqual(0, result.Length);  }  [TestMethod]  public void Test\_ShortConsultationTime()  {  TimeSpan[] startTimes = { new TimeSpan(10, 0, 0), new TimeSpan(15, 0, 0) };  int[] durations = { 90, 10 };  TimeSpan beginWorkingTime = new TimeSpan(8, 0, 0);  TimeSpan endWorkingTime = new TimeSpan(18, 0, 0);  int consultationTime = 1;  string[] result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  Assert.IsTrue(result.Contains("08:00-08:01"));  }  [TestMethod]  public void Test\_LongConsultationTime()  {  TimeSpan[] startTimes = { new TimeSpan(10, 0, 0) };  int[] durations = { 30 };  TimeSpan beginWorkingTime = new TimeSpan(8, 0, 0);  TimeSpan endWorkingTime = new TimeSpan(18, 0, 0);  int consultationTime = 540;  string[] result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  Assert.AreEqual(0, result.Length);  }  [TestMethod]  public void Test\_OverlappingEvents()  {  TimeSpan[] startTimes = { new TimeSpan(10, 0, 0), new TimeSpan(10, 15, 0) };  int[] durations = { 30, 30 };  TimeSpan beginWorkingTime = new TimeSpan(8, 0, 0);  TimeSpan endWorkingTime = new TimeSpan(18, 0, 0);  int consultationTime = 15;  string[] result = \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime);  Assert.IsFalse(result.Contains("10:00-10:15"));  }  [TestMethod] |

Продолжение таблицы А.6

|  |
| --- |
| public void Test\_NegativeConsultationTime()  {  TimeSpan[] startTimes = { new TimeSpan(10, 0, 0) };  int[] durations = { 30 };  TimeSpan beginWorkingTime = new TimeSpan(8, 0, 0);  TimeSpan endWorkingTime = new TimeSpan(18, 0, 0);  int consultationTime = -10;  Assert.ThrowsException<Exception>(() => \_calculations.AvailablePeriods(startTimes, durations, beginWorkingTime, endWorkingTime, consultationTime));  }  [TestMethod]  public void Test\_InvalidDurationArray()  {  TimeSpan[] startTimes = { new TimeSpan(10, 0, 0) };  int[] durations = { };  Assert.ThrowsException<Exception>(() => \_calculations.AvailablePeriods(startTimes, durations, new TimeSpan(8, 0, 0), new TimeSpan(18, 0, 0), 30));  }  [TestMethod]  public void Test\_InvalidStartTimesArray()  {  TimeSpan[] startTimes = { };  int[] durations = { 30 };  Assert.ThrowsException<Exception>(() => \_calculations.AvailablePeriods(startTimes, durations, new TimeSpan(8, 0, 0), new TimeSpan(18, 0, 0), 30));  }  }  } |