**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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гончарова Н. Е. 21.03.2025 г.

Специальность, группа: 09.02.07, 43П

Нижний Новгород

2025 г.

**Содержание**

[**ВВЕДЕНИЕ** 3](#_Toc192236257)

[**РАЗРАБОТКА НАСТОЛЬНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** 5](#_Toc192236258)

[**1.1.** **Разработка, администрирование и защита баз данных** 5](#_Toc192236259)

[**1.2.** **Разработка модулей программного обеспечения для компьютерных систем** 9](#_Toc192236260)

[**1.3.** **Сопровождение и обслуживание программного обеспечения компьютерных** 14](#_Toc192236261)

[**1.4.** **Осуществление интеграции программных модулей** 17](#_Toc192236262)

[**1.5.** **Ручное тестирование** 19](#_Toc192236263)

[**1.6.** **Проектирование UML–диаграмм** 23](#_Toc192236264)

[**ЗАКЛЮЧЕНИЕ** 25](#_Toc192236265)

[**ПРИЛОЖЕНИЯ** 26](#_Toc192236266)

[**Приложение А** 26](#_Toc192236267)

# **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 г.

В современном мире информационные технологии играют ключевую роль во всех сферах жизни: от бизнеса и промышленности до образования и медицины. Разработка программного обеспечения (ПО) для компьютерных систем является неотъемлемой частью цифровизации и автоматизации процессов. Учебная практика по данной дисциплине актуальна по следующим причинам:

1. Высокий спрос на разработчиков ПО. Современные компании нуждаются в квалифицированных специалистах, способных разрабатывать и поддерживать программные решения для различных компьютерных систем. Обучение практическим навыкам разработки модулей ПО позволяет студентам подготовиться к реальной работе в IT–сфере.
2. Развитие модульного подхода в программировании. В настоящее время разработка ПО строится на принципах модульности, что упрощает создание, тестирование и поддержку программных продуктов. Владение методами модульного программирования является важным навыком для будущих специалистов.
3. Актуальность использования современных технологий. Практика охватывает изучение языков программирования, инструментов разработки, методов тестирования и отладки программного кода. Это позволяет студентам освоить востребованные технологии, используемые в реальных проектах.

Таким образом, учебная практика «Разработка модулей программного обеспечения для компьютерных систем» является актуальной и востребованной, так как способствует подготовке квалифицированных специалистов, способных разрабатывать современные программные продукты и решать актуальные задачи IT–индустрии.

Целью данной учебной практики является овладение практических навыков разработки, тестирования и отладки модулей программного обеспечения, а также освоение современных методов и инструментов программирования для создания эффективной и надёжной компьютерной системы.

Для достижения данной цели поставлены следующие задачи:

1. Изучить принципы модульного программирования и ознакомиться с архитектурой программных систем и принципами их построения;
2. Ознакомиться с принципами объектно–ориентированного программирования.
3. Разработать программные модули на языке программирования C# на кроссплатформенном XAML фреймворке для платформы .NET – Avalonia;
4. Использовать современные инструменты и среды разработки. Реализовать взаимодействия между модулями ПО;
5. Применить модульное и интеграционное тестирование. Использовать системы контроля версий (Git).

Практическим результатом прохождения учебной практики является отчёт по выполнению заданий с представленными в нём скриншотами, программными кодами и диаграммами.

# **РАЗРАБОТКА НАСТОЛЬНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

* 1. **Разработка, администрирование и защита баз данных**

В рамках учебной практики была предоставлена предметная область, которая описывала магазин ООО «Пиши–стирай» занимающийся продажей канцелярских товаров.

Для данной предметной области была разработана база данных с использованием PostgreSQL – одной из самых надежных и мощных реляционных систем управления базами данных (СУБД). При проектировании базы данных была соблюдена третья нормальная форма, что позволило:

* Исключить избыточность данных.
* Минимизировать вероятность аномалий при добавлении, изменении и удалении записей.
* Обеспечить логичную структуру данных и удобство работы с ними.

Для предотвращения несогласованности данных в базе реализованы механизмы ссылочной целостности. Это означает, что:

* Все связи между таблицами построены с использованием внешних ключей (FOREIGN KEY), что предотвращает удаление или изменение записей, которые связаны с другими таблицами.
* Используются ограничения целостности (CONSTRAINTS), обеспечивающие корректность вводимых данных, например:
* Уникальные значения для идентификаторов (PRIMARY KEY).
* Ограничения на допустимые диапазоны значений.
* Контроль каскадного удаления или обновления данных.

Скрип со всеми созданными таблицами представлен в Приложении А в таблице А1. База данных включает в себя сущности, отражающие бизнес–процессы магазина «Пиши–стирай»:

1. Продукты (product)
2. Производители продуктов (productmanufacturer)
3. Производители (manufacturers)
4. Поставщики продуктов (productsupplier)
5. Поставщики (suppliers)
6. Единицы измерения (unitofmeasurement)
7. Категории (categories)
8. Заказанные продукты (orderproduct)
9. Заказы (orders)
10. Статусы (status)
11. Пункты выдачи (pickuppoint)
12. Пользователи (users)
13. Роли (role).

Весь список сущностей отображён в таблице №1.

Таблица №1 – Список сущностей

|  |  |  |
| --- | --- | --- |
| **№** | **Название** | **Назначение** |
| 1 | product | Содержит информацию о продуктах, их характеристиках, стоимости, наличии и статусе. |
| 2 | productmanufacturer | Связывает продукты с их производителями. |
| 3 | manufacturers | Хранит информацию о производителях продукции. |
| 4 | productsupplier | Связывает продукты с их поставщиками. |
| 5 | suppliers | Содержит информацию о поставщиках. |
| 6 | unitofmeasurement | Хранит данные об единицах измерения продукции. |
| 7 | categories | Содержит информацию о категориях товаров. |
| 8 | orderproduct | Связывает заказы с продуктами, указывая количество каждого товара в заказе. |
| 9 | orders | Хранит информацию о заказах, их статусе, дате, пункте выдачи и коде получения. |
| 10 | status | Содержит статусы заказов. |
| 11 | pickuppoint | Хранит данные о пунктах выдачи заказов. |
| 12 | users | Содержит информацию о пользователях системы, включая их роли. |
| 13 | role | Описывает роли пользователей в системе. |

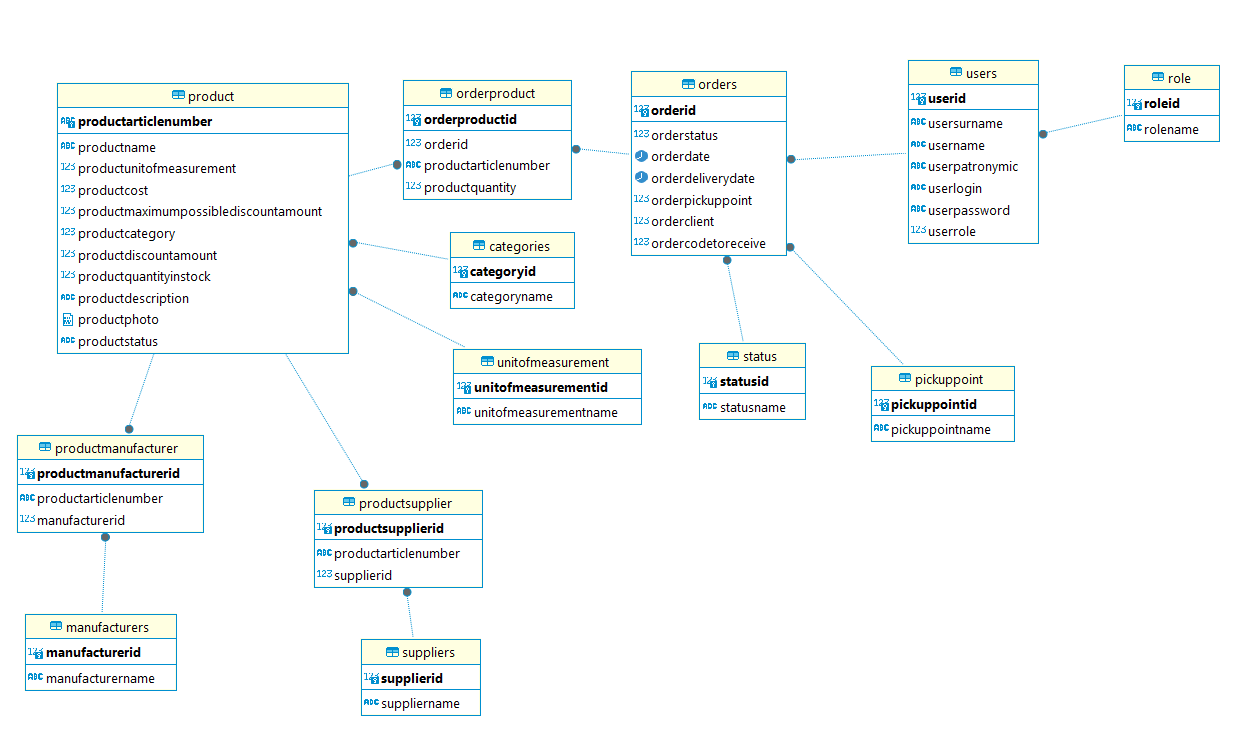
ER–диаграмма – это способ графического представления информационной модели, который позволяет показать все таблицы, их атрибуты и связи между ними. В нашем случае ER–диаграмма представлена на рисунке №1.

Рисунок №1 – ER–диаграмма БД «Пиши–стирай»

В таблице №2 описаны связи между таблицами в базе данных.

Таблица №2 – Список связей

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название связи** | **Сущности, участвующие в связи** | **Назначение** |
| 1 | M:M | product – productsupplier – suppliers | Один продукт может иметь нескольких поставщиков, и один поставщик может поставлять несколько продуктов. |
| 2 | 1:M | manufacturers – productmanufacturer – product | Один производитель может выпускать несколько продуктов. |
| 3 | 1:M | categories – product | Одна категория может включать несколько продуктов. |
| 4 | 1:M | unitofmeasurement – product | Одна единица измерения может относиться к нескольким продуктам. |
| 5 | 1:M | orders – orderproduct – product | Один заказ может включать несколько продуктов. |
| 6 | 1:M | users – orders | Один пользователь может сделать несколько заказов. |
| 7 | 1:M | role – users | Одна роль может быть присвоена нескольким пользователям. |
| 8 | 1:M | pickuppoint – orders | Один пункт выдачи может обслуживать несколько заказов. |
| 9 | 1:M | status – orders | Один статус может быть назначен нескольким заказам. |

* 1. **Разработка модулей программного обеспечения для компьютерных систем**

Разработка приложения для магазина «Пиши–стирай» реализовалась на принципах паттерна MVVM.

MVVM (Model–View–ViewModel) – это архитектурный паттерн, используемый в разработке приложений с разделением логики, представления и данных. Он помогает сделать код более структурированным, удобным для сопровождения и тестирования.

Основные компоненты паттерна MVVM:

1. Model (Модель)

* Отвечает за данные и бизнес–логику приложения.
* Может взаимодействовать с базой данных, веб–API или другими источниками данных.
* Не зависит от пользовательского интерфейса.

1. View (Представление)

* Отображает информацию пользователю.
* Представляет собой визуальный интерфейс приложения (окна, кнопки, списки и т. д.).
* Не содержит логики обработки данных, а только подписывается на изменения в ViewModel.

1. ViewModel (Модель представления)

* Посредник между Model и View.
* Содержит свойства и команды, к которым привязан интерфейс.
* Реализует механизм двустороннего связывания данных (Data Binding), позволяя View автоматически обновляться при изменении данных в ViewModel.

Работа MVVM заключается в следующем:

1. ViewModel получает данные из Model, обрабатывает их и предоставляет в удобном виде для View.
2. View подписывается на изменения в ViewModel и автоматически обновляется при изменении данных.
3. Пользователь взаимодействует с View, вызывая команды ViewModel, которые изменяют Model.
4. Изменения в Model передаются обратно в ViewModel, а затем обновляют View.

Преимущества MVVM:

* Разделение логики и интерфейса – код становится более чистым и структурированным.
* Упрощенное тестирование – можно тестировать Model и ViewModel отдельно от интерфейса.
* Гибкость и повторное использование – ViewModel можно применять с разными представлениями.
* Удобство поддержки и расширения – проще вносить изменения в код.

Паттерн MVVM особенно полезен при разработке UI–приложений, так как позволяет четко разделить ответственность между логикой обработки данных и отображением интерфейса.

Для разработки интерфейсов страниц приложения с авторизацией, выводом списка товаров, просмотром заказов, их редактированием и формированием, а также добавлением, редактированием и удалением товаров для магазина «Пиши–стирай» был выбран кроссплатформенный XAML фреймворк для платформы .NET – Avalonia. Avalonia позволяет создавать интерактивные и интуитивно понятные пользовательские интерфейсы, которые работают на различных операционных системах, включая Windows, macOS и Linux.

Использование Avalonia обеспечивает высокую производительность приложения и возможность удобного масштабирования интерфейсов для различных устройств и разрешений экранов.

* + 1. **Разработка страницы авторизации**

При разработке интерфейса страницы с авторизацией приложения были в основном использованы следующие элементы:

1. Верхняя часть:

* Логотип.
* Заголовок "Авторизация".

1. Форма ввода:

* Поля для логина и пароля с пояснительными текстами.
* Капча и поле для её ввода, отображаются при необходимости.
* Кнопка "Проверить" для капчи.

1. Кнопки действий:

* "Авторизоваться".
* "Просмотреть список товаров" (доступна без авторизации).

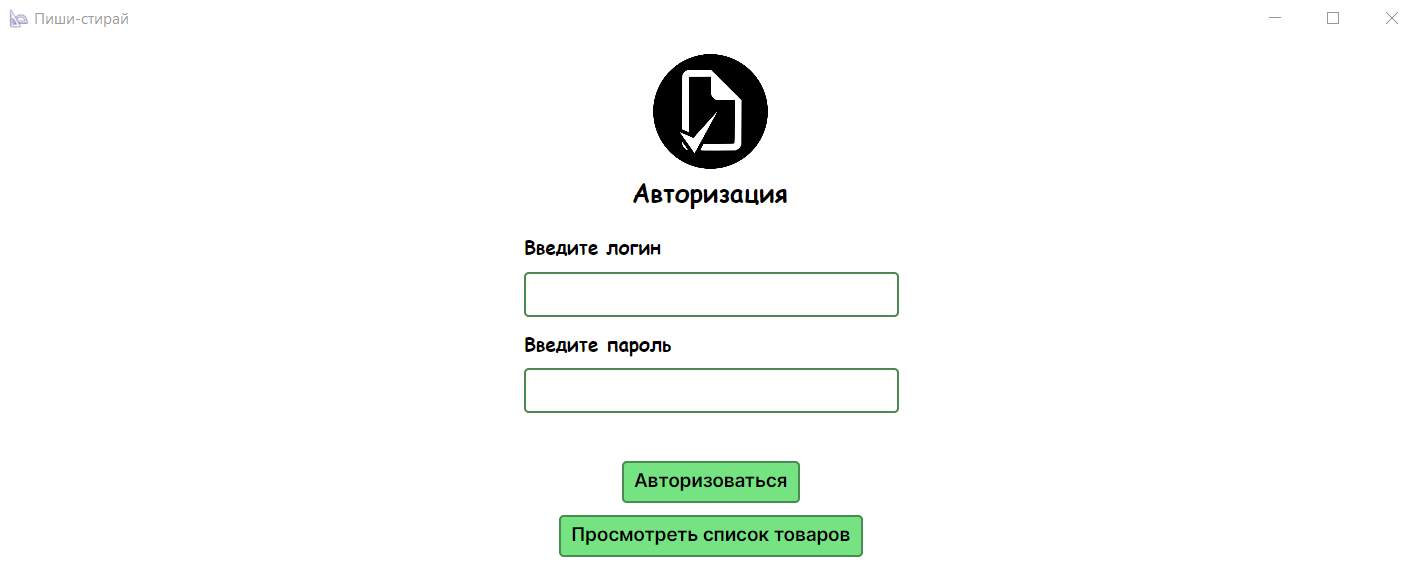
В качестве результата разработки интерфейса страницы с авторизацией в системе для магазина «Пиши–стирай» ниже представлен рисунок №2.

Рисунок №2 – Интерфейс страницы с авторизацией

На данной странице реализуется процесс авторизации пользователей с дополнительной защитой с помощью капчи, а также поддерживает возможность просмотра страницы с продуктами в гостевом режиме.

При вводе логина и пароля данные проверяются по списку пользователей. Если они корректны, происходит переход на страницу с продуктами. Если данные неверные, вызывается функция создания капчи – на экране появляется случайный набор символов и линий, усложняющих распознавание кода.

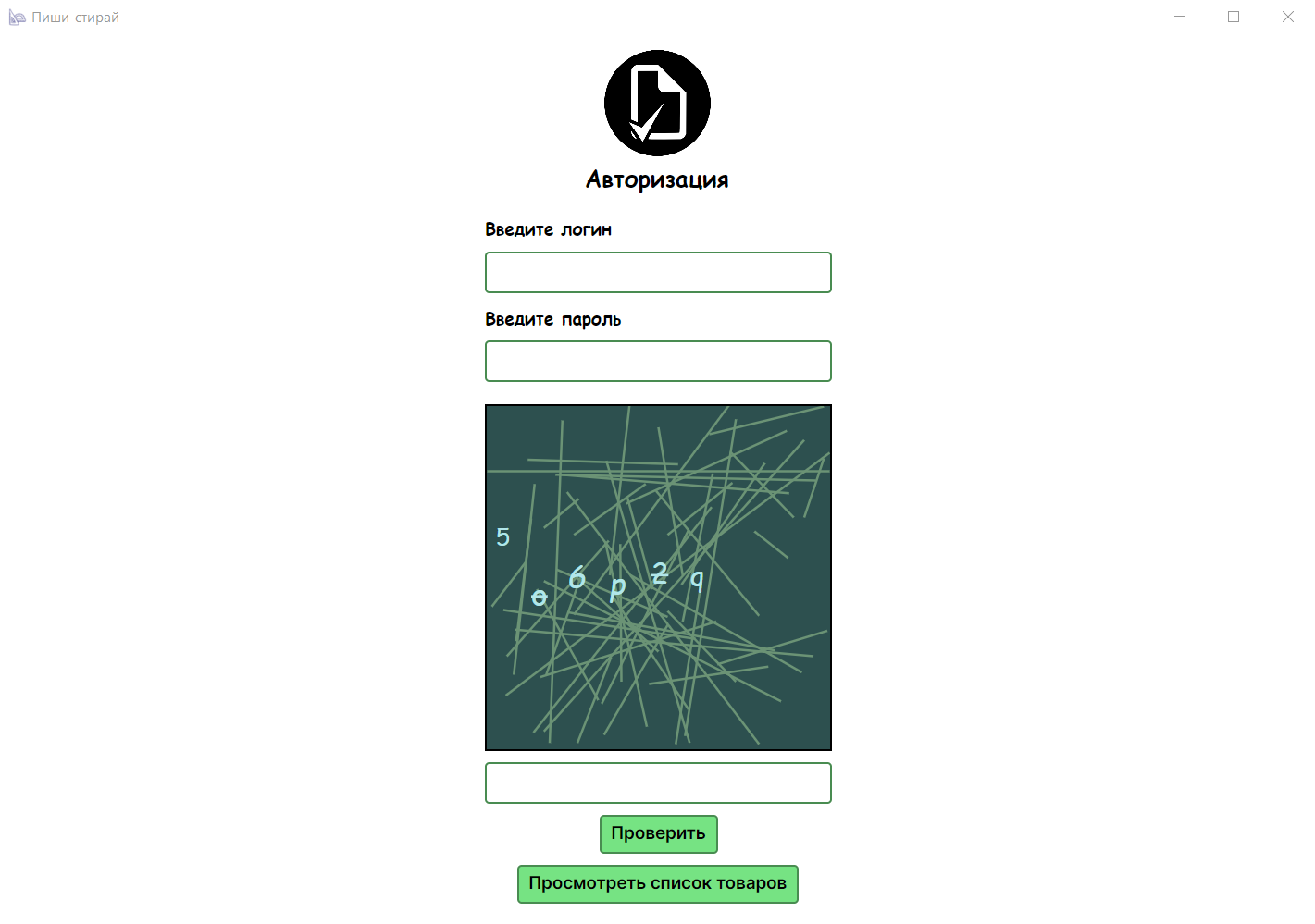
Страница с отображением капчи при некорректной авторизации представлен на рисунке №3.

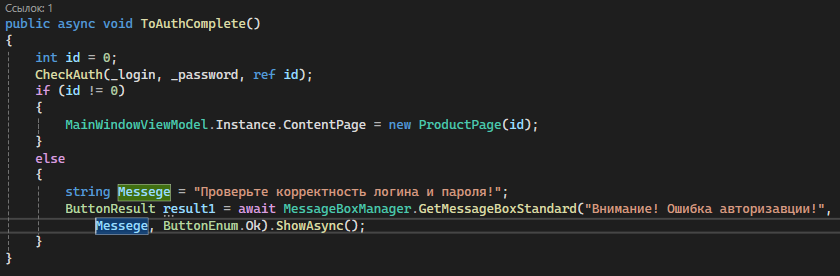
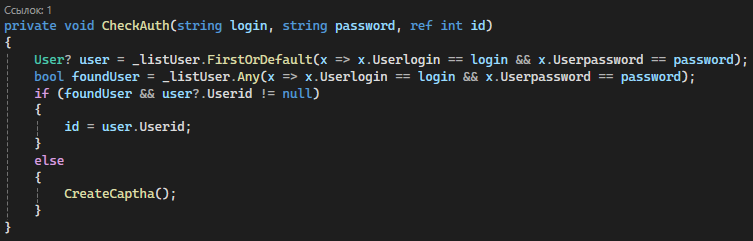
Рисунок №3 – Интерфейс страницы с авторизацией с капчей

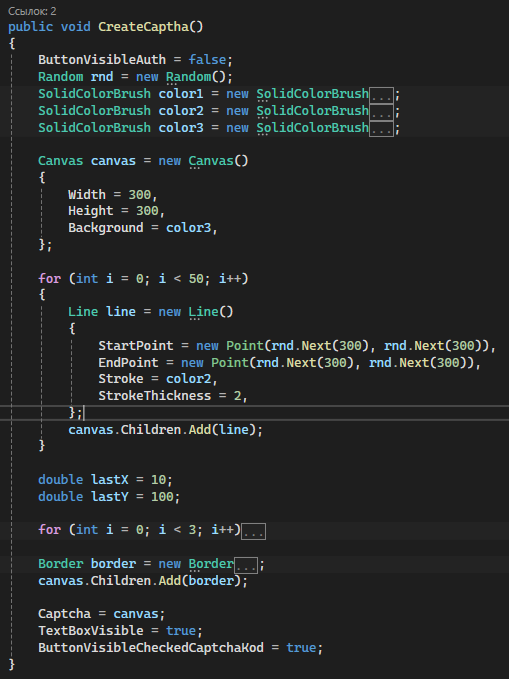
Пользователь должен ввести капчу вместе с логином и паролем. Если ввод верный, авторизация завершается успешно. При повторной ошибке включается таймер блокировки ввода на 10 секунд – поля для ввода отключаются, а после истечения времени появляется новая капча для следующей попытки.

Помимо авторизации, предусмотрен гостевой режим: пользователь может открыть страницу с товарами без ввода данных, переходя на неё с фиксированным идентификатором (гость). Это позволяет просматривать товары, но с ограниченным функционалом по сравнению с авторизованными пользователями.

Таким образом, программа поддерживает два пути доступа к продуктам – через защищённую авторизацию и свободный гостевой просмотр.

Ниже представлены рисунки с №4 по №8 с кодом реализации логики для данной страницы.

Рисунок №4 – Код реализации кнопки «Авторизоваться»

Рисунок №5 – Код реализации проведения авторизации

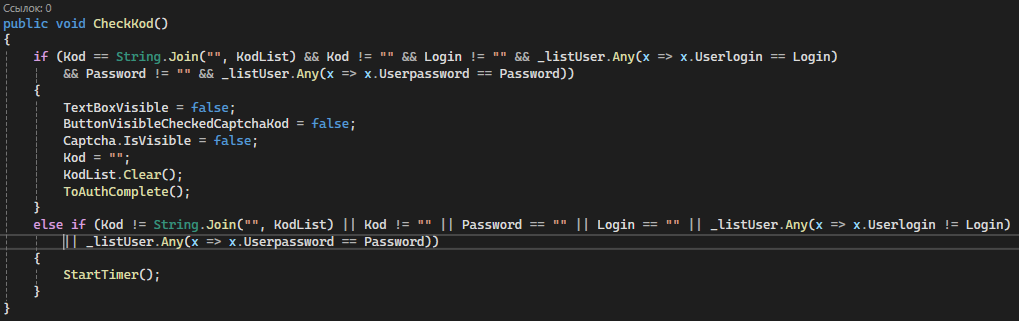
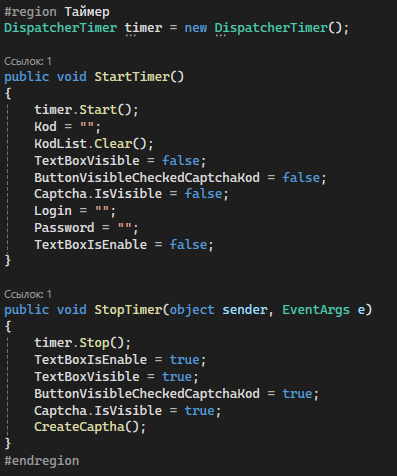
Рисунок №6 – Код реализации создания капчи

Рисунок №7 – Код реализации проверки кода капчи и повторной авторизации

Рисунок №8 – Код реализации таймера при неверном коде капчи

* + 1. **Разработка страницы со списком товаров**

При разработке интерфейса страницы со списком товаров приложения были в основном использованы следующие элементы:

1. Верхняя панель:

* Заголовок "Список товаров" с динамическим количеством товаров.
* Текущий пользователь и кнопка "Выйти из профиля".

1. Фильтрация и поиск:

* Поле поиска.
* Выпадающие списки: "Сортировка", "Фильтр скидок", "Выберите пункт выдачи".

1. Кнопки управления заказами:

* "Просмотреть текущий заказ" – отображается при наличии товаров.
* "Просмотреть все заказы" – доступна для администратора/менеджера.

1. Отображение списка товаров:

* Сообщение "Результаты не найдены", если ничего не найдено.
* Список товаров с элементами, включающими:
* Изображение, название, описание, производителя.
* Цену и зачеркнутую старую цену (при скидке).
* Информацию о скидке.
* Контекстное меню для добавления товара в заказ.

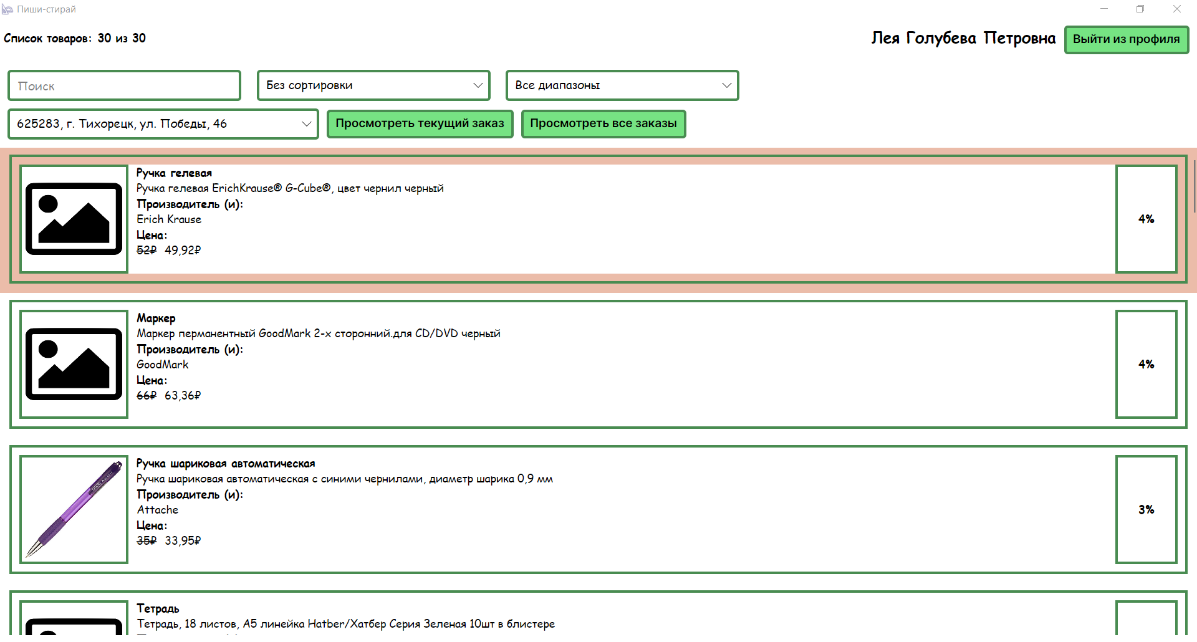
В качестве результата разработки интерфейса страницы со списком товаров в системе для магазина «Пиши–стирай» ниже представлен рисунок №9.

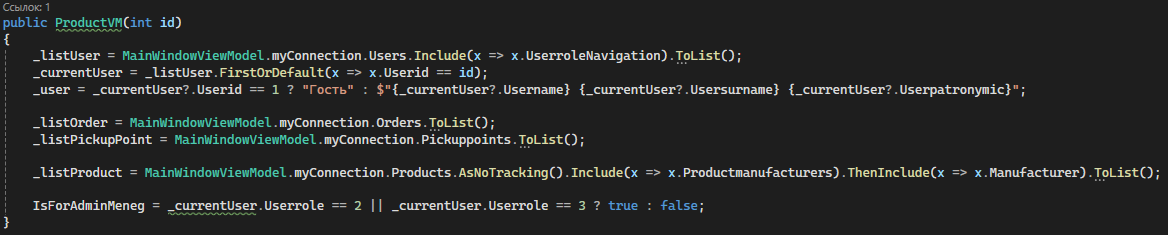
Рисунок №9 – Интерфейс страницы со списком товаров

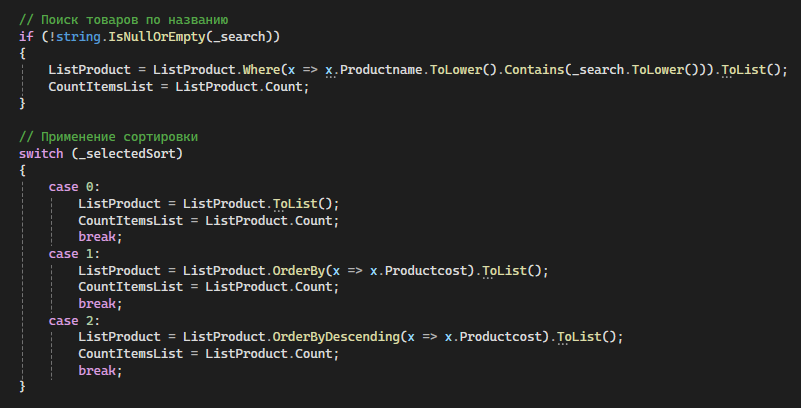
Данная страница реализует логику управления списками товаров, пользователей, пунктов выдачи и заказов в приложении. Когда пользователь заходит в систему, программа определяет, кто он – администратор, менеджер или обычный покупатель, и загружает соответствующие данные. Если пользователь не зарегистрирован, он отображается как "Гость".

Страница загружает список товаров, пунктов выдачи и заказов. Если пользователь – администратор или менеджер, ему открываются дополнительные возможности управления. Основной задачей программы является фильтрация, сортировка и поиск товаров.

Когда пользователь собирает заказ, программа проверяет, был ли уже создан текущий заказ. Если нет, она требует выбрать пункт выдачи, иначе заказ нельзя создать. После выбора пункта выдачи программа создает заказ, запоминает дату оформления и привязывает его к пользователю. Далее программа генерирует уникальный трехзначный код для получения заказа. Затем можно добавлять товары в заказ. Если заказ уже создан, программа просто добавляет в него новые позиции.

Для навигации предусмотрены кнопки перехода: на страницу авторизации, на страницу текущего заказа или на страницу всех заказов, если пользователь администратор или менеджер.

Ниже представлены рисунки с №10 по №15 с кодом реализации основной логики для данной страницы.

Рисунок №10 – Код реализации который загружает данные пользователя, заказы и пункты выдачи

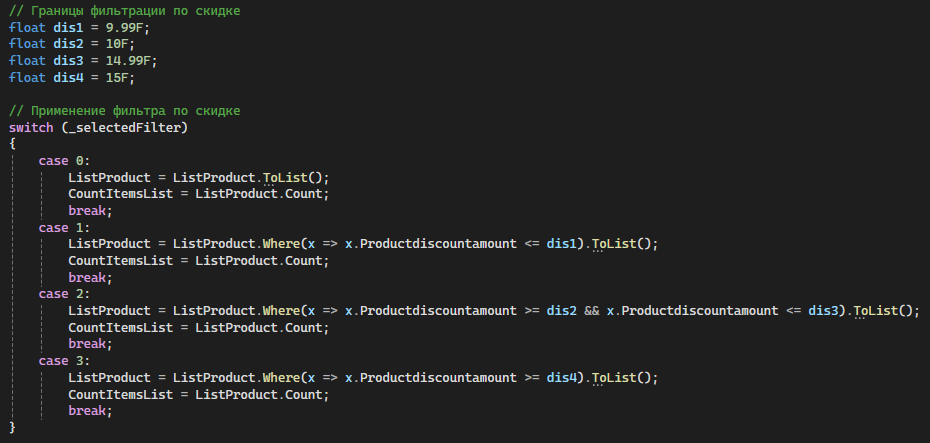
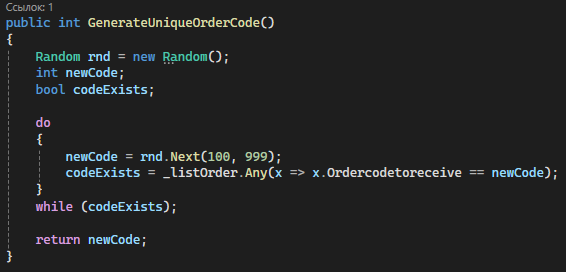
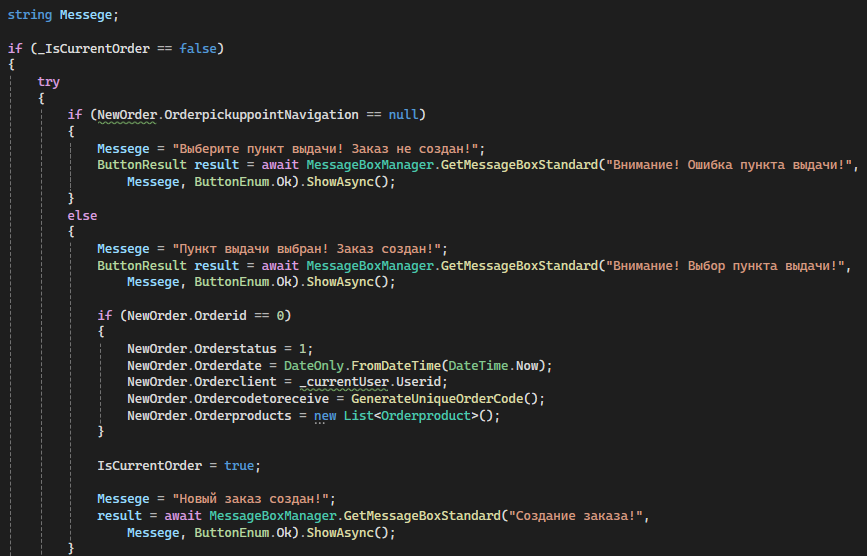
Рисунок №11 – Код реализации с логикой поиска товара и сортировки по стоимости

Рисунок №12 – Код реализации с логикой фильтрации по скидке

Рисунок №13 – Код реализации метода генерации кода для заказа

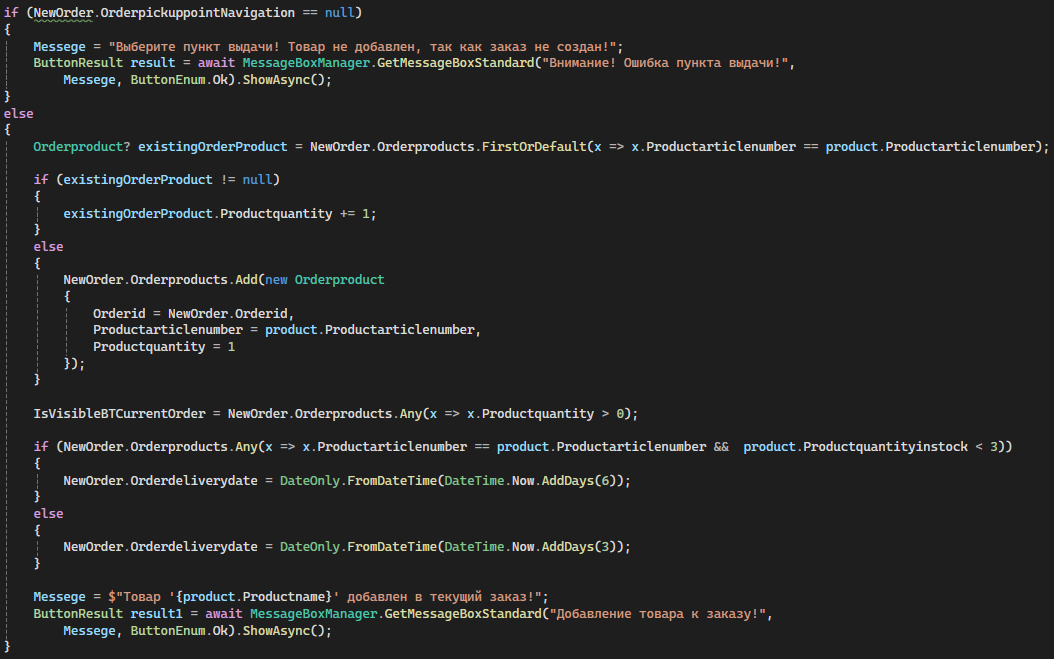
Рисунок №14 – Код реализации с логикой создания заказа

Рисунок №15 – Код реализации с логикой добавления товара в заказ

* + 1. **Разработка страницы с формированием заказа**

При разработке интерфейса страницы с формированием заказа приложения были в основном использованы следующие элементы:

1. Верхняя панель управления:

* Заголовок "Текущий заказ".
* Кнопки: "Сформировать заказ", "Сформировать талон в PDF", "Назад", "Завершить текущий заказ".
* ФИО пользователя или Гость.

1. Информация о заказе:

* Код, дата, время и пункт выдачи заказа.
* Сумма заказа, скидка и итоговая сумма.
* Количество товаров в заказе.

1. Список товаров в заказе:

* Изображение, название, описание и производители.
* Цена (обычная и со скидкой).
* Информация о наличии скидки.

1. Управление товарами:

* Кнопки: "+", "–" (изменение количества), "х" (удаление).

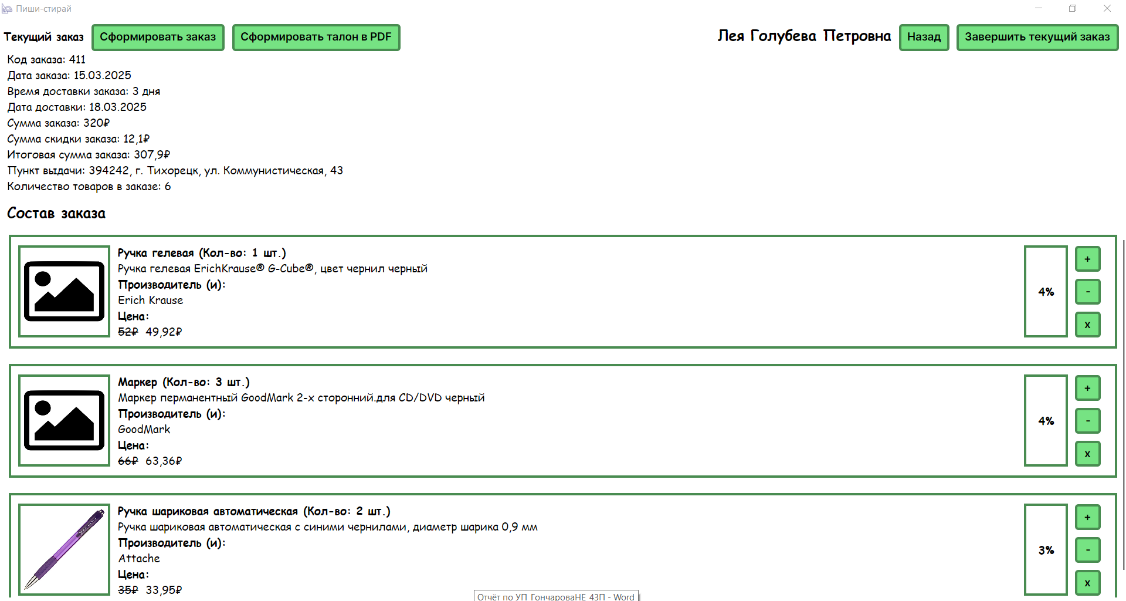
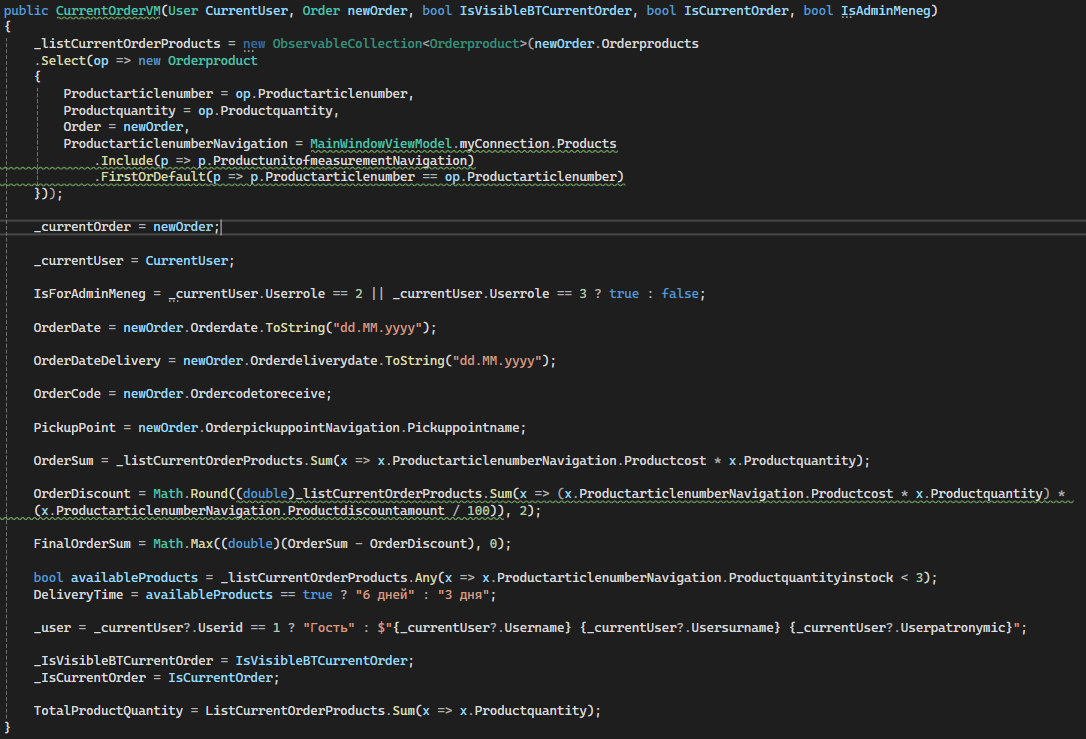
В качестве результата разработки интерфейса страницы с формированием заказа в системе для магазина «Пиши–стирай» ниже представлен рисунок №16.

Рисунок №16 – Интерфейс страницы с формированием заказа

Логика программного кода данной страницы управляет текущим заказом в приложении, предоставляя данные о его содержимом, пользователе и статусе. Он хранит список товаров, дату заказа, код получения, сумму, скидку и итоговую стоимость. Определяет пункт выдачи и срок доставки в зависимости от наличия товаров. Позволяет пользователю вернуться к списку товаров, завершить заказ с предупреждением, сохранить его в базе данных, очистить и сформировать талон в формате PDF. В зависимости от условий выполнения заказа меняется статус, что отражается в интерфейсе.

Ниже представлены рисунки с №17 по № с кодом реализации основной логики для данной страницы.

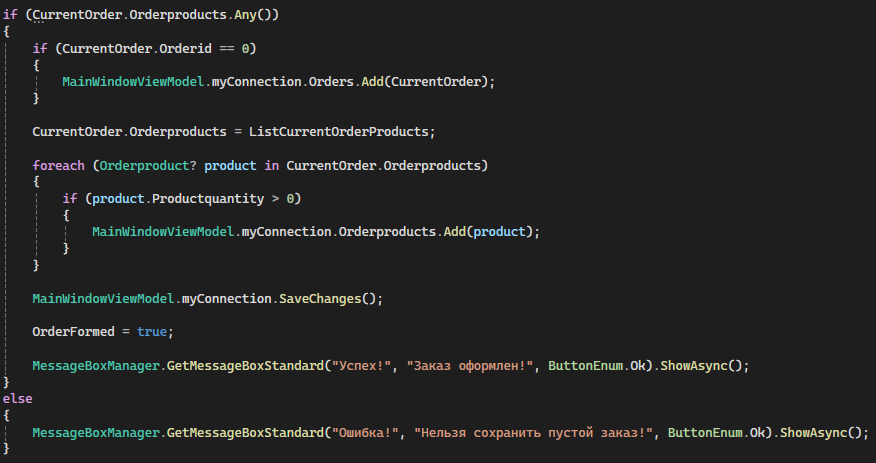
Рисунок №17 – Код реализации для инициализации текущего заказа

Рисунок №18 – Код реализации формирования заказа

* + 1. **Разработка страницы для работы со всеми заказами**

При разработке интерфейса страницы с формированием заказа приложения были в основном использованы следующие элементы:

1. Верхняя панель управления:

* Заголовок "Все заказы".
* Отображение имени пользователя.
* Кнопка "Назад" (возвращение на предыдущую страницу).

1. Панель фильтрации и сортировки заказов: выбор сортировки (по стоимости) и фильтра по скидке.
2. Список заказов:

Каждый заказ представлен в виде карточки с информацией: статус заказа (например, "Новый", "Завершён"), номер заказа, дата заказа, дата доставки, состав заказа (список товаров), общая сумма заказа, сумма заказа со скидкой, общая сумма скидки, общий процент скидки, ФИО клиента или Гость.

1. Каждая карточка заказа выделяется цветом фона в зависимости от количества товара на складе.

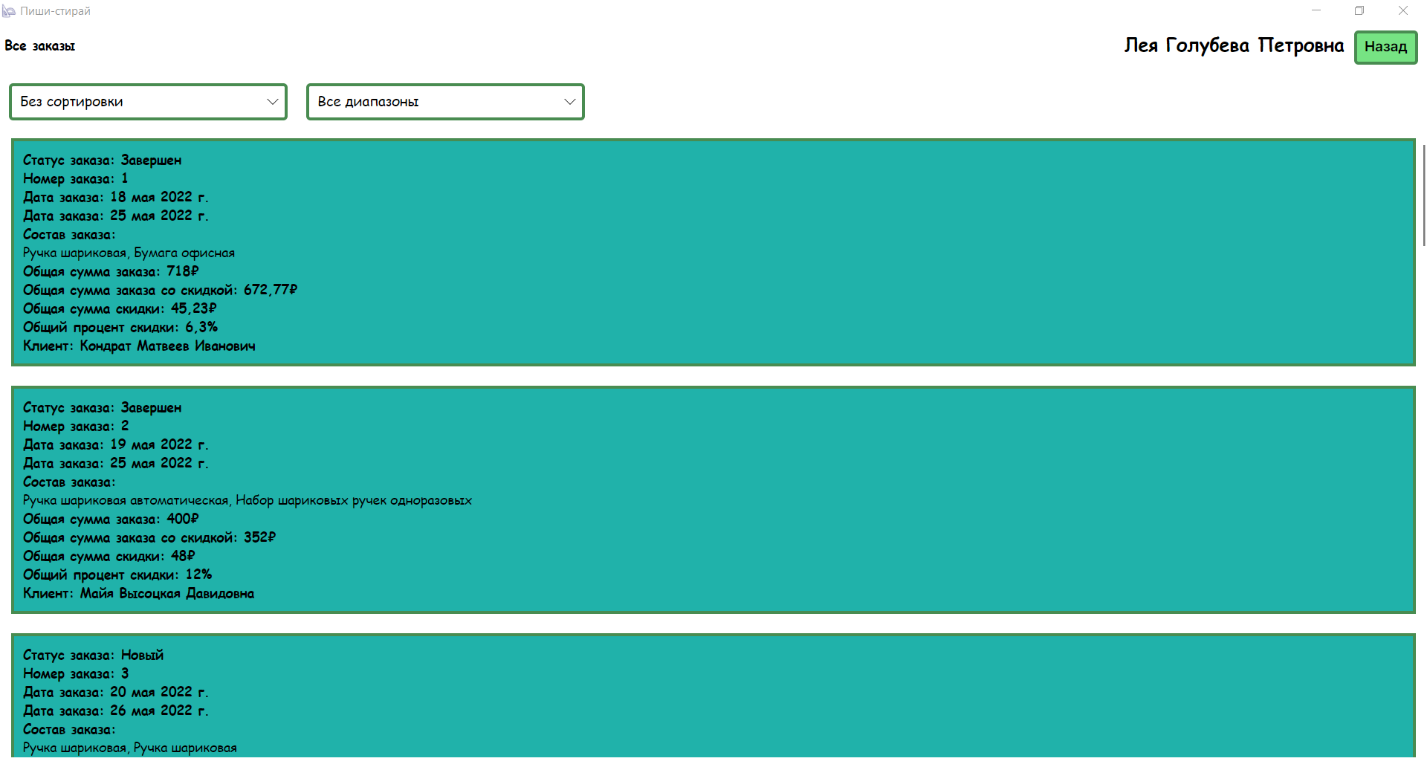
В качестве результата разработки интерфейса страницы со списком заказов в системе для магазина «Пиши–стирай» ниже представлен рисунок №19.

Рисунок №19 – Интерфейс страницы со списком всех заказов

Данная страница позволяет управлять списком заказов. Она хранит информацию о текущем пользователе (ими могут быть только менеджер и администратор), его ФИО и список заказов. При переходе на страницу загружается список всех заказов, у которых есть товары. Реализована возможность обратного перехода на страницу товаров или редактирования заказа. Также доступны сортировка и фильтрация заказов. Сортировка может быть по возрастанию или убыванию суммы заказа, а фильтрация — по величине скидки. При изменении параметров сортировки или фильтрации список заказов обновляется.

Ниже представлены рисунки с №20 по №21 с кодом реализации основной логики для данной страницы.

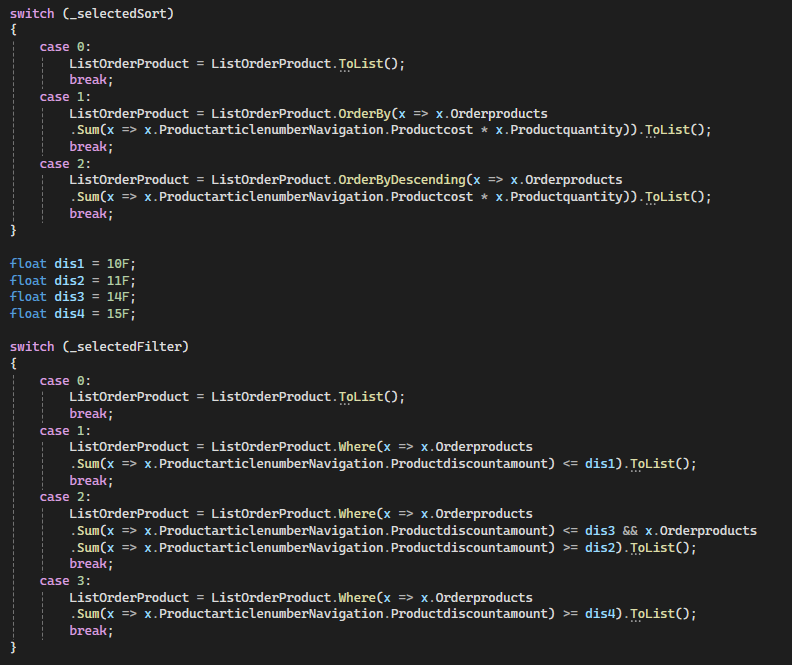
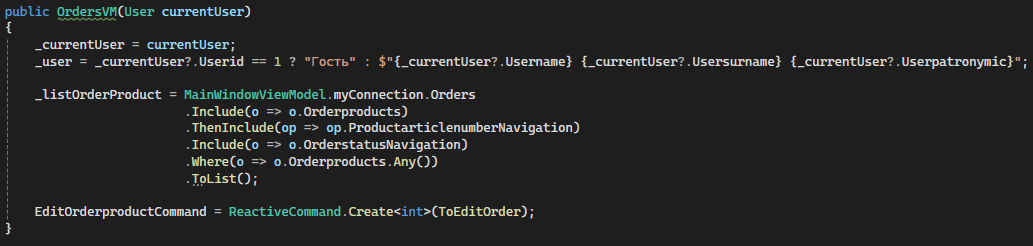
Рисунок №20 – Код реализации для загрузки данных о заказах

Рисунок №21 – Код реализации сортировки и фильтрации заказов

* + 1. **Разработка страницы для редактирования заказа**

При разработке интерфейса страницы с редактированием заказа приложения были в основном использованы следующие элементы:

1. Верхняя панель: заголовок "Редактирование заказа", имя пользователя, кнопка "Назад".
2. Редактирование заказа:

* Выпадающий список для изменения статуса заказа.
* Выбор новой даты доставки через DatePicker.
* Кнопка "Сохранить изменения" для применения изменений.

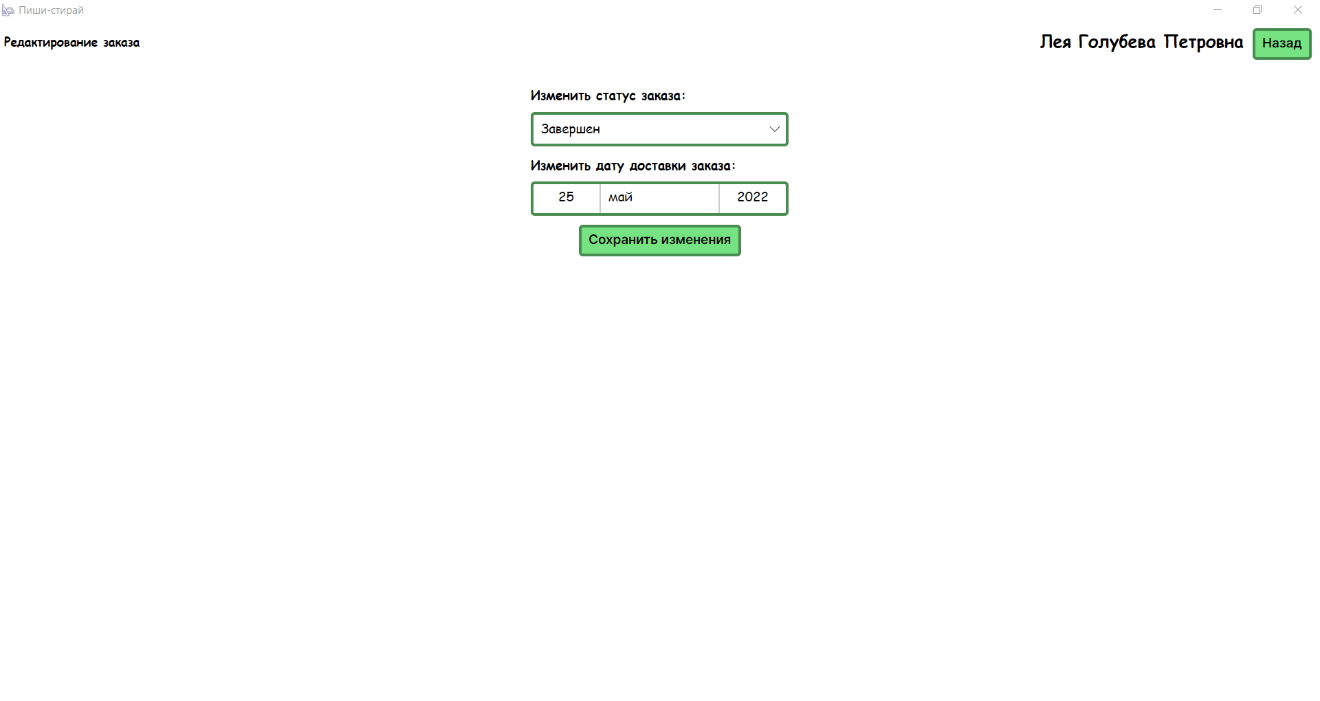
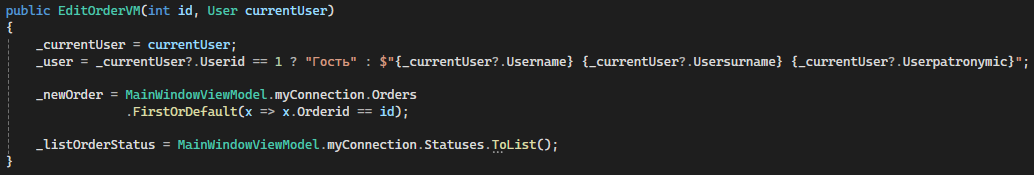
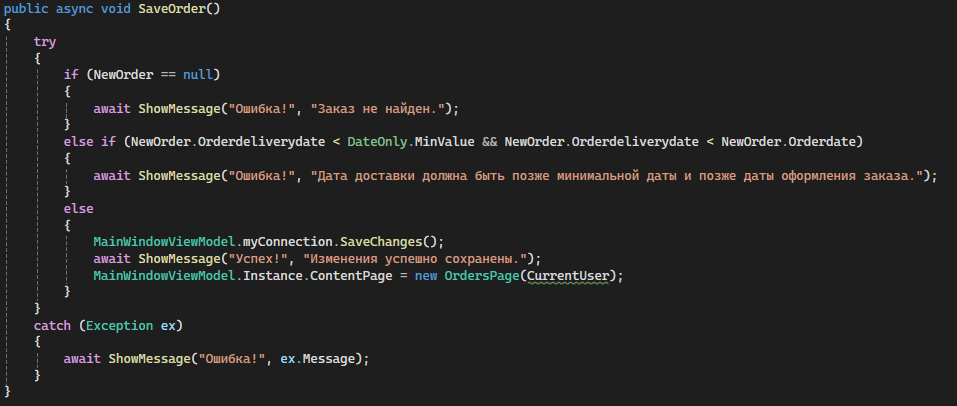
В качестве результата разработки интерфейса страницы с редактированием заказа в системе для магазина «Пиши–стирай» ниже представлен рисунок №22.

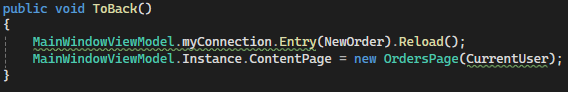
Рисунок №22 – Интерфейс страницы редактирования выбранного заказа

На данной странице реализуется процесс редактирования заказа. Она хранит информацию о текущем пользователе, его ФИО, списке доступных статусов заказов и самом редактируемом заказе.

При открытии страницы загружаются все возможные статусы заказов. Реализованы две основные функции: возврат к списку заказов и сохранение изменений. При возврате заказ не сохранит изменения, и пользователь перенаправляется на страницу заказов. При сохранении проверяется корректность даты доставки — она должна быть позже даты оформления и минимального допустимого значения. Если проверка пройдена, изменения сохраняются в базе, и пользователь получает уведомление об успехе. В случае ошибки выводится сообщение с пояснением.

Ниже представлены рисунки с №23 по №21 с кодом реализации основной логики для данной страницы.

Рисунок №23 – Код реализации загрузки данных о выбранном заказе

Рисунок №24 – Код реализации метода сохранения заказа в базе данных

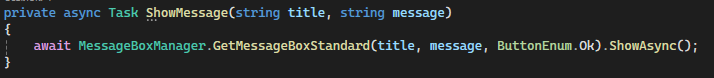
Рисунок №25 – Код реализации метода возврата к списку всех заказов

Рисунок №26 – Код реализации метода вывода сообщений пользователю

* 1. **Осуществление интеграции программных модулей**

Для формирования оптимального графика работы сотрудников была разработана DLL–библиотека, которая будет возвращать после своего выполнения список свободных временных интервалов (заданного размера) в графике сотрудника.

Эта библиотека предоставляет метод для расчета свободных временных промежутков в рабочем графике.

Метод принимает:

* список времен начала уже запланированных записей,
* их длительности,
* границы рабочего дня,
* продолжительность одной консультации.

В процессе работы метод проверяет корректность входных данных, упорядочивает занятые интервалы по времени начала и находит промежутки, достаточные для новой консультации. Эти свободные периоды возвращаются в виде строковых значений времени начала и окончания.

Если входные данные некорректны (например, массивы разной длины или время консультации отрицательное), метод выбрасывает исключение и возвращает пустой массив.

Код DLL-библиотеки представлен в Приложении А в таблице №А2.

Для библиотеки были разработаны модульные тесты, проверяющие разные аспекты работы метода AvailablePeriods:

1. Полный рабочий день свободен – проверялось, что, если нет записей, весь рабочий день доступен для консультаций.
2. Одна запись создает два свободных интервала – убеждались, что, если есть одна запись, оставшееся время корректно разбивается на две части.
3. Нет свободных интервалов, если интервалы слишком короткие – тестировал случай, когда между записями недостаточно времени для новой консультации.
4. Запись в начале рабочего дня – проверялось, что метод не включает время до начала работы.
5. Запись в конце рабочего дня – убеждались, что метод не учитывает время после окончания работы.
6. Записи идут подряд без перерывов – проверялось, что, если все время занято записями, метод не возвращает свободных интервалов.
7. Несколько записей с разрывами – убеждались, что метод корректно находит промежутки между записями.
8. Некорректные входные данные (null) – проверялось, что, если переданы пустые массивы, метод возвращает пустой результат.
9. Несовпадающие по длине массивы – тестировал случай, когда массивы времени начала и длительности записей имеют разную длину.
10. Отрицательное время консультации – убеждались, что метод корректно обрабатывает ситуацию, если длительность консультации указана отрицательной.

Все эти тесты помогли проверить различные сценарии работы метода и убедиться, что он корректно рассчитывает свободные временные промежутки.

Код модульных тестов представлен в Приложении А в таблице №А3.

* 1. **Ручное тестирование**

Было проведено ручное тестирование формы добавления товара администратором. В рамках тестирования было составлено 5 тест-кейсов, направленных на проверку успешности выполнения добавления товара, обязательности полей для ввода, уже существующего артикля у товара, отрицательной цены и некорректного формата изображения для товара. Ниже представлены все 5 тест-кейсов с таблицы №3 по таблицу №7.

Таблица №3 – Тестовый пример #1

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_FUNC\_01 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Успешное добавление нового товара |
| **Краткое изложение теста** | Проверка возможности добавления товара, если все данные введены корректно. |
| **Этапы теста** | 1. Открыть страницу добавления товара в роли администратора. 2. Ввести корректные данные во все обязательные поля. 3. Нажать кнопку «Сохранить». 4. Проверить, что товар добавился в базу данных. |
| **Тестовые данные** | 1. Артикль товара: P12345 2. Название товара: Чай черный 3. Единица измерения товара: 1 (литр) 4. Стоимость товара: 500 5. Категория товара: 2 (Напитки) 6. Количество товара на складе: 100 7. Описание товара: Черный чай высшего качества. 8. Фото товара: Загруженное изображение. 9. Статус товара: Доступен. |
| **Ожидаемый результат** | Товар успешно добавлен, отображается в списке товаров. |
| **Фактический результат** | Товар успешно добавлен, отображается в списке товаров. |
| **Статус** | Зачёт |
| **Предварительное условие** | Доступ к базе данных, права администратора. |
| **Постусловие** | Новый товар добавлен в систему. |
| **Примечания/комментарии** | – |

Таблица №4 – Тестовый пример #2

|  |  |
| --- | --- |
| **Тестовый пример #** | TC\_FUNC\_02 |
| **Приоритет тестирования** | Средний |
| **Заголовок/название теста** | Добавление товара с уже существующим артиклем товара |
| **Краткое изложение теста** | Проверка корректной обработки ошибки дублирования артикула. |

Продолжение таблицы №4 – Тестовый пример #2

|  |  |
| --- | --- |
| **Этапы теста** | 1. Открыть страницу добавления товара в роли администратора. 2. Ввести данные товара, использовав уже существующий артикль товара. 3. Нажать кнопку «Сохранить». |
| **Тестовые данные** | 1. Артикль товара: P10001 (существует в базе) 2. Название товара: Чай зеленый 3. Единица измерения товара: 1 (литр) 4. Стоимость товара: 450 5. Категория товара: 2 (Напитки) 6. Количество товара на складе: 90 7. Описание товара: Зелёный чай среднего качества. 8. Фото товара: Загруженное изображение. 9. Статус товара: Не доступен. |
| **Ожидаемый результат** | Ошибка «Товар с таким артикулом уже существует». |
| **Фактический результат** | Ошибка «Товар с таким артикулом уже существует». |
| **Статус** | Зачёт |
| **Предварительное условие** | В базе уже есть товар с артикулом P10001. |
| **Постусловие** | Новый товар не добавлен. |
| **Примечания/комментарии** | – |

Таблица №5 – Тестовый пример #3

|  |  |
| --- | --- |
| Тестовый пример **#** | TC\_FUNC\_03 |
| **Приоритет тестирования** | Высокий |
| **Заголовок/название теста** | Попытка добавления товара без указания обязательных полей |
| **Краткое изложение теста** | Проверка, что система не позволяет добавлять товар без заполнения всех обязательных полей. |
| **Этапы теста** | 1. Открыть страницу добавления товара в роли администратора. 2. Заполнить только часть обязательных полей. 3. Нажать кнопку «Сохранить». |
| **Тестовые данные** | 1. Артикль товара: P12346 2. Название товара: (пустое) 3. Единица измерения товара: (не указано) 4. Стоимость товара: 600 5. Категория товара: 3 (Продукты) |

Продолжение таблицы №5 – Тестовый пример #3

|  |  |
| --- | --- |
| **Ожидаемый результат** | Ошибка «Обязательные поля не заполнены». |
| **Фактический результат** | Ошибка «Обязательные поля не заполнены». |
| **Статус** | Зачёт |
| **Предварительное условие** | В базе данных обязательные поля не должны содержать пустые значения. |
| **Постусловие** | Товар не добавлен. |
| **Примечания/комментарии** | Проверить валидацию полей. |

Таблица №6 – Тестовый пример #4

|  |  |
| --- | --- |
| Тестовый пример **#** | TC\_FUNC\_04 |
| **Приоритет тестирования** | Средний |
| **Заголовок/название теста** | Добавление товара с отрицательной ценой |
| **Краткое изложение теста** | Проверка, что система не принимает отрицательное значение цены. |
| **Этапы теста** | 1. Открыть страницу добавления товара в роли администратора. 2. Ввести данные товара с отрицательной ценой. 3. Нажать кнопку «Сохранить». |
| **Тестовые данные** | 1. Артикль товара: P56789 2. Название товара: Сок яблочный 3. Единица измерения товара: 1 (литр) 4. Стоимость товара: -300 5. Категория товара: 2 (Напитки) 6. Количество товара на складе: 50 7. Описание товара: Сок яблочный высшего качества. 8. Фото товара: Загруженное изображение. 9. Статус товара: Доступен. |
| **Ожидаемый результат** | Ошибка «Цена не может быть отрицательной». |
| **Фактический результат** | Ошибка «Цена не может быть отрицательной». |
| **Статус** | Зачёт |
| **Предварительное условие** | В базе данных обязательные поля не должны содержать пустые значения. |
| **Постусловие** | Товар не добавлен. |
| **Примечания/комментарии** | Проверить контроль ввода отрицательных значений. |

Таблица №7 – Тестовый пример #5

|  |  |
| --- | --- |
| Тестовый пример **#** | TC\_FUNC\_05 |
| **Приоритет тестирования** | Низкий |
| **Заголовок/название теста** | Добавление товара с некорректным форматом изображения |
| **Краткое изложение теста** | Проверка, что система не принимает неподдерживаемый формат изображения. |
| **Этапы теста** | 1. Открыть страницу добавления товара в роли администратора. 2. Ввести корректные данные о товаре. 3. Загрузить изображение в неподдерживаемом формате (например, .txt). 4. Нажать кнопку «Сохранить». |
| **Тестовые данные** | 1. Артикль товара: P67890 2. Название товара: Вода минеральная 3. Единица измерения товара: 1 (литр) 4. Стоимость товара: 100 5. Категория товара: 2 (Напитки) 6. Количество товара на складе: 40 7. Описание товара: Вода минеральная низкого качества. 8. Фото товара: file.txt. 9. Статус товара: Доступен. |
| **Ожидаемый результат** | Ошибка «Неподдерживаемый формат файла». |
| **Фактический результат** | Ошибка «Неподдерживаемый формат файла». |
| **Статус** | Зачёт |
| **Предварительное условие** | – |
| **Постусловие** | Товар не добавлен. |
| **Примечания/комментарии** | Проверить допустимые форматы изображений (.jpg, .png). |

# **ЗАКЛЮЧЕНИЕ**

В ходе учебной практики была достигнута поставленная цель – овладение практическими навыками разработки, тестирования и отладки модулей программного обеспечения. В процессе выполнения работы были изучены принципы модульного программирования, архитектура программных систем, а также основные концепции объектно–ориентированного программирования.

В рамках практики разработаны программные модули на языке C# на кроссплатформенном XAML фреймворке для платформы .NET – Avalonia, что позволило закрепить полученные теоретические знания на практике. Также были использованы современные инструменты и среды разработки, что способствовало эффективному созданию и интеграции программных компонентов. Реализованы взаимодействия между модулями программного обеспечения, что позволило оценить важность гибкости и масштабируемости архитектурных решений.

Особое внимание было уделено тестированию программных модулей, включая модульное и интеграционное тестирование, что повысило надёжность разрабатываемого программного обеспечения. В процессе работы также применялись системы контроля версий (Git), что способствовало эффективному управлению разработкой и командной работе.

Таким образом, в результате прохождения учебной практики были приобретены ценные практические навыки, необходимые для дальнейшей профессиональной деятельности в области разработки программного обеспечения.

# **ПРИЛОЖЕНИЯ**

## **Приложение А**

Таблица №А1 – Скрип базы данных для компании «Пиши–стирай»

|  |
| --- |
| **create** **table** **Role**  (  RoleID serial **primary** **key**,  RoleName **varchar**(100) **not** **null**  );  **create** **table** Users  (  UserID serial **primary** **key**,  UserSurname **varchar**(100) **not** **null**,  UserName **varchar**(100) **not** **null**,  UserPatronymic **varchar**(100) **not** **null**,  UserLogin **varchar**(100) **not** **null**,  UserPassword **varchar** **not** **null**,  UserRole **int** **not** **null**,  **constraint** UserRole\_fk **foreign** **key** (UserRole) **references** **Role**(RoleID),  **constraint** unique\_user\_login **unique** (UserLogin)  );  **create** **table** Status  (  StatusID serial **primary** **key**,  StatusName **varchar**(100) **not** **null**  );  **create** **table** PickupPoint  (  PickupPointID serial **primary** **key**,  PickupPointName **varchar**(100) **not** **null**  );  **create** **table** Orders  (  OrderID serial **primary** **key**,  OrderStatus **int** **not** **null**,  OrderDate **date** **not** **null**,  OrderDeliveryDate **date** **not** **null**,  OrderPickupPoint **int** **not** **null**,  OrderClient **int** **not** **null** **DEFAULT** 1,  OrderCodeToReceive **int** **not** **null**,  **constraint** OrderStatus\_fk **foreign** **key** (OrderStatus) **references** Status(StatusID),  **constraint** OrderPickupPoint\_fk **foreign** **key** (OrderPickupPoint) **references** PickupPoint(PickupPointID),  **constraint** OrderClient\_fk **foreign** **key** (OrderClient) **references** Users(UserID)  );  **create** **table** Categories  (  CategoryID serial **primary** **key**,  CategoryName **varchar**(100) **not** **null**  );  **create** **table** UnitOfMeasurement  (  UnitOfMeasurementID serial **primary** **key**,  UnitOfMeasurementName **varchar**(100) **not** **null**  );  **create** **table** Product  (  ProductArticleNumber **varchar**(100) **primary** **key**,  ProductName **varchar** **not** **null**,  ProductUnitOfMeasurement **int** **not** **null**,  ProductCost **float4** **not** **null**,  ProductMaximumPossibleDiscountAmount **smallint** **null**,  ProductCategory **int** **not** **null**,  ProductDiscountAmount **float4** **null**,  ProductQuantityInStock **int** **not** **null**,  ProductDescription **varchar** **not** **null**,  ProductPhoto **varchar** **not** **null**,  ProductStatus **varchar** **null**,  **constraint** ProductUnitOfMeasurement\_fk **foreign** **key** (ProductUnitOfMeasurement) **references** UnitOfMeasurement(UnitOfMeasurementID),  **constraint** ProductCategory\_fk **foreign** **key** (ProductCategory) **references** Categories(CategoryID)  );  **create** **table** OrderProduct  (  OrderProductID serial **primary** **key**,  OrderID **int** **not** **null**,  ProductArticleNumber **varchar**(100) **not** **null**,  ProductQuantity **int** **not** **null**,  **constraint** Order\_OrderProduct\_fk **foreign** **key** (OrderID) **references** Orders(OrderID) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**,  **constraint** Product\_OrderProduct\_fk **foreign** **key** (ProductArticleNumber) **references** Product(ProductArticleNumber)  );  **create** **table** Manufacturers  (  ManufacturerID serial **primary** **key**,  ManufacturerName **varchar**(100) **not** **null**  );  **create** **table** ProductManufacturer  (  ProductManufacturerID serial **primary** **key**,  ProductArticleNumber **varchar**(100) **not** **null**,  ManufacturerID **int** **not** **null**,  **constraint** Product\_ProductManufacturer\_fk **foreign** **key** (ProductArticleNumber) **references** Product(ProductArticleNumber) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**,  **constraint** Manufacturer\_ProductManufacturer\_fk **foreign** **key** (ManufacturerID) **references** Manufacturers(ManufacturerID)  );  **create** **table** Suppliers  (  SupplierID serial **primary** **key**,  SupplierName **varchar**(100) **not** **null**  );  **create** **table** ProductSupplier  (  ProductSupplierID serial **primary** **key**,  ProductArticleNumber **varchar**(100) **not** **null**,  SupplierID **int** **not** **null**,  **constraint** Product\_ProductSupplier\_fk **foreign** **key** (ProductArticleNumber) **references** Product(ProductArticleNumber) **ON** **DELETE** **CASCADE** **ON** **UPDATE** **CASCADE**,  **constraint** Supplier\_ProductSupplier\_fk **foreign** **key** (SupplierID) **references** Suppliers(SupplierID)  ); |

Таблица №А2 – Код DLL–библиотеки для составления списка свободных временных интервалов в графике сотрудника.

|  |
| --- |
| namespace SF2022User\_NN\_Lib  {  public class Calculations  {  public static string[] AvailablePeriods(TimeSpan[] startTimes, int[] durations, TimeSpan beginWorkingTime,  TimeSpan endWorkingTime, int consultationTime)  {  try  {  if (startTimes == null || durations == null)  throw new ArgumentNullException("Входные массивы не могут быть пустыми");  if (startTimes.Length != durations.Length)  throw new ArgumentException("Массивы времени начала и длительности должны иметь одинаковую длину");  if (consultationTime <= 0)  throw new ArgumentException("Время консультации должно быть больше нуля");  if (beginWorkingTime >= endWorkingTime)  throw new ArgumentException("Время начала работы должно быть раньше времени окончания");  List<string> FreeTimeIntervals = new List<string>();  List<(TimeSpan start, TimeSpan end)> BeginningAndEndWorkingDay = new List<(TimeSpan, TimeSpan)>();  for (int i = 0; i < startTimes.Length; i++)  {  BeginningAndEndWorkingDay.Add((startTimes[i], startTimes[i].Add(TimeSpan.FromMinutes(durations[i]))));  }  BeginningAndEndWorkingDay = BeginningAndEndWorkingDay.OrderBy(slot => slot.start).ToList();  TimeSpan currentStart = beginWorkingTime;  foreach ((TimeSpan start, TimeSpan end) slot in BeginningAndEndWorkingDay)  {  if (currentStart.Add(TimeSpan.FromMinutes(consultationTime)) <= slot.start)  {  FreeTimeIntervals.Add($"{currentStart:hh\\:mm}-{slot.start:hh\\:mm}");  }  if (currentStart < slot.end)  {  currentStart = slot.end;  }  }  if (currentStart.Add(TimeSpan.FromMinutes(consultationTime)) <= endWorkingTime)  {  FreeTimeIntervals.Add($"{currentStart:hh\\:mm}-{endWorkingTime:hh\\:mm}");  }  return FreeTimeIntervals.ToArray();  }  catch (Exception ex)  {  Console.WriteLine($"Ошибка: {ex.Message}");  return Array.Empty<string>();  }  }  }  } |

Таблица №А3 – Код модульных тестов для DLL–библиотеки

|  |
| --- |
| using SF2022User\_NN\_Lib;  namespace SF2022UserNNLibUnitTests  {  [TestClass]  public sealed class UnitTests  {  [TestMethod]  public void NoAppointments\_FullDayAvailable()  {  TimeSpan[] startTimes = { };  int[] durations = { };  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(18, 0, 0);  int consultationTime = 30;  string[]? result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  Assert.AreEqual(1, result.Length);  Assert.AreEqual("09:00-18:00", result[0]);  }  [TestMethod]  public void SingleAppointment\_CreatesTwoFreeIntervals()  {  TimeSpan[] startTimes = { new TimeSpan(12, 0, 0) };  int[] durations = { 60 };  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(18, 0, 0);  int consultationTime = 30;  string[]? result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  CollectionAssert.AreEqual(new[] { "09:00-12:00", "13:00-18:00" }, result);  }  [TestMethod]  public void NoFreeIntervals\_TooShortGaps()  {  TimeSpan[] startTimes = { new TimeSpan(10, 0, 0), new TimeSpan(14, 0, 0) };  int[] durations = { 120, 240 };  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(18, 0, 0);  int consultationTime = 130;  string[]? result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  Assert.AreEqual(0, result.Length);  }  [TestMethod]  public void AppointmentAtStart\_DoesNotIncludeBeforeWorkHours()  {  TimeSpan[] startTimes = { new TimeSpan(9, 0, 0) };  int[] durations = { 60 };  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(18, 0, 0);  int consultationTime = 30;  string[]? result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  Assert.AreEqual(1, result.Length);  Assert.AreEqual("10:00-18:00", result[0]);  }  [TestMethod]  public void AppointmentAtEnd\_DoesNotIncludeAfterWorkHours()  {  TimeSpan[] startTimes = { new TimeSpan(17, 0, 0) };  int[] durations = { 60 };  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(18, 0, 0);  int consultationTime = 30;  var result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  Assert.AreEqual(1, result.Length);  Assert.AreEqual("09:00-17:00", result[0]);  }  [TestMethod]  public void BackToBackAppointments\_NoAvailableIntervals()  {  TimeSpan[] startTimes = { new TimeSpan(9, 0, 0), new TimeSpan(10, 0, 0), new TimeSpan(11, 0, 0) };  int[] durations = { 60, 60, 60 };  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(12, 0, 0);  int consultationTime = 30;  var result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  Assert.AreEqual(0, result.Length);  }  [TestMethod]  public void MultipleAppointments\_WithGaps()  {  TimeSpan[] startTimes = { new TimeSpan(9, 30, 0), new TimeSpan(14, 30, 0) };  int[] durations = { 60, 60 };  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(18, 0, 0);  int consultationTime = 30;  var result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  CollectionAssert.AreEqual(new[] { "09:00-09:30", "10:30-14:30", "15:30-18:00" }, result);  }  [TestMethod]  public void NullInput\_ReturnsEmptyArray()  {  TimeSpan[] startTimes = null;  int[] durations = null;  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(18, 0, 0);  int consultationTime = 30;  var result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  Assert.AreEqual(0, result.Length);  }  [TestMethod]  public void MismatchedArrays\_ReturnsEmptyArray()  {  TimeSpan[] startTimes = { new TimeSpan(9, 0, 0) };  int[] durations = { 60, 30 };  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(18, 0, 0);  int consultationTime = 30;  var result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  Assert.AreEqual(0, result.Length);  }  [TestMethod]  public void NegativeConsultationTime\_ReturnsEmptyArray()  {  TimeSpan[] startTimes = { new TimeSpan(9, 0, 0) };  int[] durations = { 60 };  TimeSpan begin = new TimeSpan(9, 0, 0);  TimeSpan end = new TimeSpan(18, 0, 0);  int consultationTime = -30;  var result = Calculations.AvailablePeriods(startTimes, durations, begin, end, consultationTime);  Assert.AreEqual(0, result.Length);  }  }  } |