**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»

Высшая школа цифровых технологий

Кафедра математики и прикладных информационных технологий

**КУРСОВОЙ ПРОЕКТ**

по предмету: «Методы и средства проектирования компьютерных приложений»

на тему: «Разработка автоматизированной информационной системы “Товарный учёт”»

**Выполнил:**

Студент 2 курса группы НТм-23-1

направления 09.04.01 «Информатика и вычислительная техника»

Лонгортов Е. Б.

**Руководитель:**

доцент, канд. пед. наук

Спирин И.С.

Тюмень 2025 г.

**Техническое задание**

Разработка веб-приложения по предметной области «Товарный учёт» на основе фрейморка Blazor в среде разработки Microsoft Visual Studio. Основными функциями является управление данными, обработка и получение информации о товарах, инвентарных операциях и сотрудниках. Разработать веб-приложение со всеми нужными функциями, разработать интуитивный и понятный интерфейс веб-приложения, опираясь на стиль современных приложений. Для разработки веб-приложения потребуется:

* провести анализ предметной области;
* спроектировать модели базы данных;
* организовать подключение базы данных к приложению;
* разработать интерфейс веб-приложения;
* разработать функционал, позволяющий редактировать, искать и добавлять данные, а также сделать возможность формирования отчёта.

**Аннотация**

**Цель** – проектирование веб-приложения «Товарный учёт» на основе фреймворка Blazor.

**Объект работы** – веб-приложение для автоматизации товарного учёта.

**Предмет работы** – процесс разработки веб-приложения для ведения товарного учёта.

**Информационная база исследования**: статьи, учебники и научные работы в области создания веб-приложений на фреймворке Blazor.

**Результат**: разработанное веб-приложение на основе фреймворка Blazor и языка C#.

**Курсовой проект состоит** из введения, трёх глав, заключения, списка использованных источников и приложения.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc196241794)

[1. Описание предметной области 6](#_Toc196241795)

[1.1 Общие сведения 6](#_Toc196241796)

[1.2 Назначение и цели создания (развития) системы 6](#_Toc196241797)

[1.3 Характеристики объектов автоматизации 6](#_Toc196241798)

[1.4 Требования к системе 7](#_Toc196241799)

[1.5 Входная и выходная информация 8](#_Toc196241800)

[2. Этапы и инструменты, методология разработки 10](#_Toc196241801)

[2.1 Методология разработки 10](#_Toc196241802)

[2.2 Инструменты и технологии 10](#_Toc196241803)

[2.3 Постановка задачи 10](#_Toc196241804)

[2.4 Описание автоматизированных функций 11](#_Toc196241805)

[2.5 Диаграмма вариантов использования 12](#_Toc196241806)

[2.6 Диаграмма деятельности 15](#_Toc196241807)

[2.7 Логическая модель 17](#_Toc196241808)

[2.8 Физическая модель 19](#_Toc196241809)

[2.9 Архитектура системы 21](#_Toc196241810)

[2.10 Разработка моделей системы 22](#_Toc196241811)

[3 Описание работы программы 25](#_Toc196241812)

[3.1 Общие сведения 25](#_Toc196241813)

[3.2 Описание пользовательского интерфейса 25](#_Toc196241814)

[ЗАКЛЮЧЕНИЕ 31](#_Toc196241815)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 32](#_Toc196241816)

[*Приложение* 34](#_Toc196241817)

# **ВВЕДЕНИЕ**

В современном ритме торговли, особенно в розничной сфере, ведение точного и своевременного товарного учёта является критически важным. Наличие автоматизированной системы учёта позволяет повысить прозрачность бизнес-процессов, упростить анализ остатков и продаж, а также своевременно принимать управленческие решения.

Курсовая работа посвящена разработке приложения «Товарный учёт», предназначенного для отслеживания движения товаров в магазине. Реализованное программное обеспечение автоматизирует процессы учёта, позволяет формировать отчёты о продажах и инвентаризациях, а также предоставляет удобный веб-интерфейс на базе технологии Blazor.

Целью данной работы является создание приложения, обеспечивающего удобный и функциональный товарный учёт в магазине.

Для достижения поставленной цели необходимо решить следующие задачи:

* разработать классы для описания товаров, сотрудников и операций;
* создать интерфейс для ввода и анализа данных инвентаризации;
* реализовать функциональность генерации отчётов по продажам;
* построить систему с использованием Blazor в качестве веб-интерфейса.

В результате выполнения данных задач будет создано приложение, способное упростить процессы управления запасами товаров и анализа продаж в магазине.

# **1. Описание предметной области**

## **1.1 Общие сведения**

Полное наименование системы: «Приложение для товарного учёта в магазине электроники».

Наименование предприятий разработчика и заказчика системы: Тюменский индустриальный университет.

Разработчик: студент группы НТм-23-1 Лонгортов Е.Б.

Плановые сроки начала и окончания работы по созданию системы: начало семестра - конец семестра.

Сведения об источниках и порядке финансирования работ: внутренние источники.

## **1.2 Назначение и цели создания (развития) системы**

1. Назначение системы — автоматизация процессов отслеживания движения товаров на складе и в торговом зале, а также упрощение формирования отчётности по продажам.
2. Цели создания системы:

* повышение точности и скорости ведения товарного учёта;
* предоставление удобного пользовательского интерфейса;
* автоматическое формирование отчётных документов по продажам и инвентаризации.

## **1.3 Характеристики объектов автоматизации**

Объектами автоматизации являются:

* товар — единица складского учёта;
* операция — любое действие, связанное с перемещением товара (поставка, продажа, инвентаризация);
* сотрудник — пользователь, выполняющий операции в системе;
* инвентаризация — процесс сверки фактического наличия товара с данными учёта.

## **1.4 Требования к системе**

1. Требования к системе в целом:

* система должна быть реализована на языке программирования C# и фрейворка .NET с Blazor;
* система должна быть протестирована на соответствие требованиям.

1. Требования к функциям (задачам), выполняемым системой; система должна обеспечивать выполнение следующих функций:

* поддержка учёта товаров (создание, редактирование, удаление, просмотр);
* фиксация операций движения товаров;
* ведение инвентаризаций;
* генерация отчётов по продажам;
* возможность фильтрации и анализа данных;
* реализация веб-интерфейса с использованием Blazor.

1. Требования к видам обеспечения:

Для математического обеспечения необходимо определить методы подсчёта суммы продаж.

Для информационного обеспечения системы приводят требования:

К структуре данных:

* данные о товарах, сотрудниках, операциях;
* структурированное хранение и возможность экспорта отчётов.

К контролю, хранению, обновлению и восстановлению данных: данные должны храниться в безопасном месте и быть доступны для восстановления.

Для лингвистического обеспечения системы приводят требования к способам организации диалога. Диалог должен происходить в интерактивном режиме.

Для программного обеспечения системы:

* используемые технологии: Blazor, Entity Framework (при наличии бд), ASP.NET Core;
* независимость от конкретного оборудования;
* возможность развертывания на локальном сервере или в облаке.

К независимости программных средств от используемых технических средств и операционной среды; операционная среда Windows 10.

Для технического обеспечения системы приводят требования к видам технических средств, сеть персональных компьютеров клиентской части с единым сервером для базы данных и управления транзакциями.

## **1.5 Входная и выходная информация**

Входной информацией является:

Информация о товаре:

* идентификатор товара;
* наименование;
* категория;
* цена;
* количество на складе.

Информация о сотруднике:

* идентификатор сотрудника;
* ФИО;
* должность.

Информация об операции (движении товара):

* идентификатор операции;
* тип операции;
* дата операции;
* связанный товар;
* количество;
* ответственный сотрудник.

Информация для инвентаризации:

* перечень фактически наличных товаров;
* данные по плановым остаткам;
* разница между фактом и планом.

Выходной информацией является:

Отчет по продажам:

* наименование товара;
* количество проданного товара;
* сумма продажи.

Аналитика по остаткам:

* список товаров;
* остаток.

# **2. Этапы и инструменты, методология разработки**

## **2.1 Методология разработки**

Проект «Товарный учёт» был разработан с использованием инкрементальной методологии, при которой система создавалась поэтапно, с последовательным добавлением новых функций и их тестированием. Это позволило гибко вносить изменения и адаптировать функциональность в процессе разработки.

## **2.2 Инструменты и технологии**

В рамках реализации были использованы следующие инструменты и технологии:

* язык программирования: C# (логика приложения), JavaScript/Blazor (интерфейс);
* фреймворк: Blazor Server (для реализации интерактивного веб-интерфейса);
* СУБД: MSSQL (для хранения информации о товарах, операциях и пользователях);
* ORM: Entity Framework Core (для удобной работы с базой данных);
* среда разработки: Visual Studio 2022;
* доп. инструменты: Git (контроль версий).

## **2.3 Постановка задачи**

Цель автоматизации — повышение эффективности управления товарными запасами за счёт минимизации ручного труда, обеспечения точности данных и оперативного формирования аналитической отчётности.

Основные задачи:

* автоматизация учёта движения товаров (приход, продажа, списание, возврат);
* реальное время обновления остатков на складе;
* генерация отчётов по продажам и инвентаризации;
* интеграция удобного веб-интерфейса для взаимодействия с системой.

Инструменты реализации:

* веб-интерфейс: фреймворк Blazor (C#);
* серверная часть: ASP.NET Core;
* хранение данных: Microsoft SQL Server.

## **2.4 Описание автоматизированных функций**

Ключевые функции системы:

1. Управление товарами:

* добавление/редактирование товаров (наименование, категория, цена, остатки);
* автоматический пересчёт количества после каждой операции.

1. Учёт операций:

* фиксация операций движения товаров (тип, дата, ответственный);
* валидация данных для предотвращения ошибок (например, списание при нулевом остатке).

1. Работа с сотрудниками:

* ведение базы сотрудников (ФИО, должность);
* привязка операций к ответственным лицам.

1. Аналитика и отчёты:

* формирование отчётов в форматах PDF/Excel;
* визуализация данных (графики продаж, динамика остатков).

1. Веб-интерфейс:

* интуитивное управление данными через браузер;
* кросс-платформенная доступность.

## **2.5 Диаграмма вариантов использования**

Пользователь системы (сотрудник магазина или администратор) взаимодействует с приложением через веб-интерфейс. Основные варианты использования включают управление товарами, сотрудниками, операциями и отчётами. На рисунке 2.1 представлена диаграмма вариантов использования.

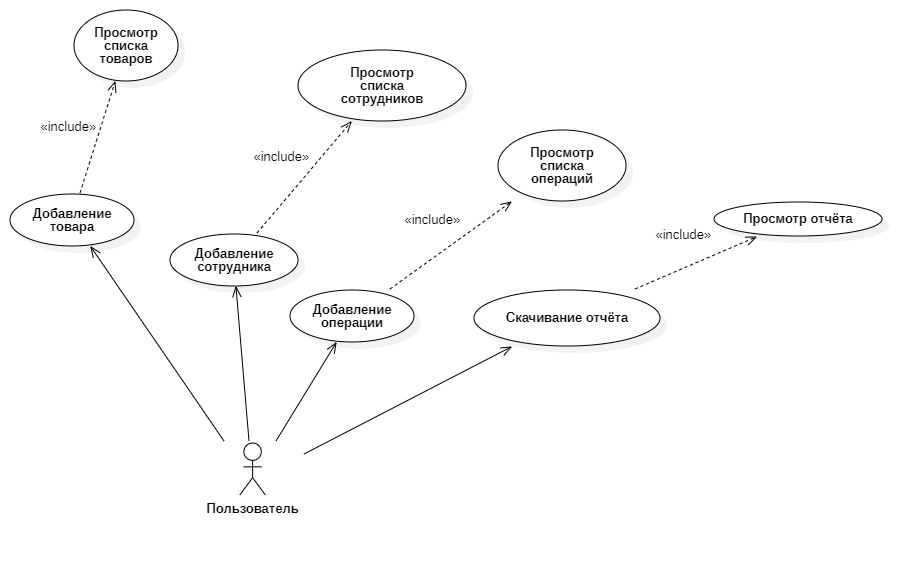


Рисунок 2.1 – Варианты использования

**Вариант использования «Просмотр списка товаров»** описывает процесс просмотра перечня товаров, доступных на складе.

***Основной поток событий:***

1. Пользователь выбирает раздел «Товары»;
2. Система отображает таблицу с данными: наименование, категория, цена, количество на складе;
3. Пользователь просматривает список, применяя фильтры (категория, остатки) при необходимости.

**Вариант использования «Добавление товара»** описывает процесс внесения нового товара в систему.

***Основной поток событий:***

1. Пользователь нажимает кнопку «Добавить товар».
2. Система открывает форму с полями:

* наименование;
* категория;
* цена;
* начальное количество.

1. Пользователь заполняет данные и подтверждает ввод.
2. Система проверяет корректность данных и сохраняет товар в базу.

**Вариант использования «Просмотр списка сотрудников»** описывает процесс просмотра данных о сотрудниках магазина.

***Основной поток событий:***

1. Пользователь переходит в раздел «Сотрудники».
2. Система выводит таблицу с ФИО, должностями и идентификаторами.
3. Пользователь использует поиск для быстрого доступа к информации.

**Вариант использования «Добавление сотрудника»** описывает процесс регистрации нового сотрудника в системе.

***Основной поток событий:***

1. Пользователь нажимает «Добавить сотрудника».
2. Система предоставляет форму с полями:

* ФИО;
* должность.

1. Пользователь вводит данные и сохраняет запись.
2. Система присваивает сотруднику уникальный идентификатор.

**Вариант использования «Просмотр списка операций»** описывает процесс анализа движений товаров (приход, продажа, списание).

***Основной поток событий:***

1. Пользователь выбирает раздел «Операции».
2. Система отображает таблицу с типом операции, датой, товаром, количеством и ответственным.
3. Пользователь фильтрует данные по дате или типу операции.

**Вариант использования «Добавление операции»** описывает процесс регистрации движения товара (например, продажи или поступления).

***Основной поток событий:***

1. Пользователь нажимает «Создать операцию».
2. Система предлагает выбрать тип операции (приход, расход, списание).
3. Пользователь заполняет:

* товар;
* количество;
* ответственный сотрудник.

1. Система проверяет доступность товара и обновляет остатки.

**Вариант использования «Генерация отчёта»** описывает процесс формирования отчёта по продажам или инвентаризации.

***Основной поток событий:***

1. Пользователь выбирает «Создать отчёт».
2. Система запрашивает параметры: период, тип отчёта (продажи, остатки).
3. Пользователь подтверждает выбор.
4. Система генерирует отчёт в формате PDF/Excel и предоставляет ссылку для скачивания.

**Вариант использования «Просмотр отчёта»** описывает процесс анализа сгенерированных отчётов.

***Основной поток событий:***

1. Пользователь выбирает раздел «Отчёты».
2. Система отображает список доступных отчётов.
3. Пользователь открывает нужный отчёт, изучает данные (графики, таблицы).

## **2.6 Диаграмма деятельности**

На рисунках 2.2-2.4 изображены соответствующие диаграммы деятельности.

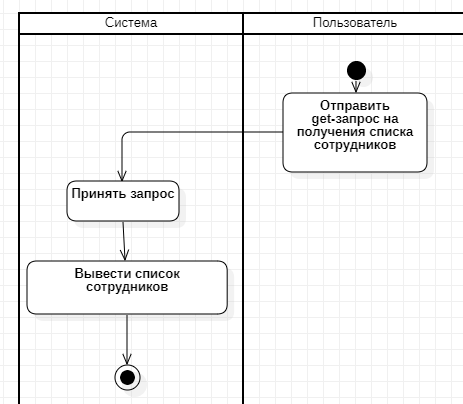


Рисунок 2.2 – Диаграмма деятельности «Получить список сотрудников»

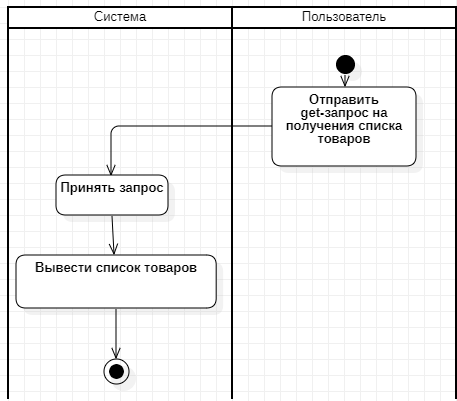


Рисунок 2.3 – Диаграмма деятельности «Получить список товаров»

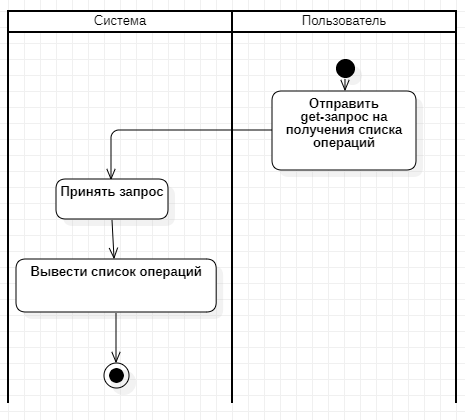


Рисунок 2.4 – Диаграмма деятельности «Получить список операций»

## **2.7 Логическая модель**

Сущности и их атрибуты:

1. **Employee** (Сотрудник):

* Id (PK, int) — уникальный идентификатор;
* FullName (string) — ФИО сотрудника;
* Position (string) — должность.

1. **Product** Товар):

* Id (PK, int) — уникальный идентификатор;
* Name (string) — наименование товара;
* Category (string) — категория товара;
* Price (decimal) — цена за единицу;
* QuantityInStock (int) — количество на складе.

1. **InventoryOperation** (Операция):

* Id (PK, int) — уникальный идентификатор операции;
* ProductId (FK, int) — ссылка на товар;
* Quantity (int) — количество товара в операции;
* OperationType (enum) — тип операции (Purchase, Sale, WriteOff, Return);
* Date (DateTime) — дата операции;
* EmployeeId (FK, int, nullable) — ссылка на сотрудника.

Взаимодействие с другими web-сервисами осуществляется по протоколу HTTP.

Связи между сущностями:

* один товар может участвовать в множестве операций;
* сотрудник может быть связан с несколькими операциями (или ни с одной, если EmployeeId = null).

На рисунке 2.5 представлена логическая модель.



Рисунок 2.5 – Логическая модель

В таблице 2.1 перечислены атрибуты сущности «Сотрудник».

Таблица 2.1 – Атрибуты сущности «Сотрудник»

|  |  |
| --- | --- |
| Имя атрибута | Тип данных |
| Код Сотрудника | Число |
| Полное имя | Текст |
| Должность | Текст |

В таблице 2.2 перечислены атрибуты сущности «Товар».

Таблица 2.2 – Атрибуты сущности «Товар»

|  |  |
| --- | --- |
| Имя атрибута | Тип данных |
| Код Товара | Число |
| Наименование товара | Текст |
| Категория | Текст |
| Цена | Число |
| Количество | Число |

В таблице 2.3 перечислены атрибуты сущности «Инвентарные операции».

Таблица 2.3 – Атрибуты сущности «Инвентарные операции»

|  |  |
| --- | --- |
| Имя атрибута | Тип данных |
| Код Операции | Число |
| Код Товара | Число |
| Код Сотрудника | Число |
| Количество | Число |
| Тип операции | Текст |
| Дата | Дата |

## **2.8 Физическая модель**

Физическая модель данных в нотации IDEF1X представляет собой структурированное описание базы данных, которое определяет физическую организацию данных, их типы и отношения между ними.

Физическая модель данных IDEF1X состоит из следующих основных элементов:

1. Сущности представляют отдельные объекты или концепции в системе, которые требуют хранения данных. Каждая сущность представляется прямоугольником с именем сущности внутри.
2. Атрибуты определяют характеристики или свойства сущностей. Они представлены овалами, связанными с соответствующей сущностью. Например, если у сущности "Сотрудник" есть атрибуты "Имя", "Фамилия" и "Дата рождения", то эти атрибуты будут связаны с сущностью "Сотрудник".
3. Отношения определяют связи между сущностями. Они представлены линиями, соединяющими сущности, и имеют имя, которое описывает характер связи. Например, отношение "Работает в" может соединять сущности "Сотрудник" и "Отдел".
4. Ключи определяют уникальные идентификаторы для сущностей. Они могут быть первичными ключами (Primary Keys), которые однозначно идентифицируют каждую запись в таблице, или внешними ключами (Foreign Keys), которые связывают записи в разных таблицах.
5. Ограничения определяют правила и условия, которые должны соблюдаться при взаимодействии с базой данных. Например, ограничение "NOT NULL" требует, чтобы определенное поле содержало значение, а ограничение "UNIQUE" требует, чтобы значения в поле были уникальными.

В качестве СУБД используется MSSQL.

На рисунке 2.6 представлена физическая модель БД.

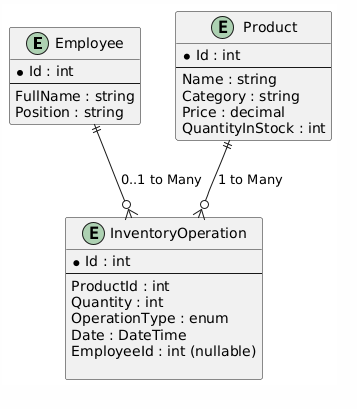


Рисунок 2.6 – Физическая модель

В таблице 2.4 перечислены поля отношений «Employee».

Таблица 2.4 – Поля отношений «Employee»

|  |  |  |
| --- | --- | --- |
| Содержание атрибута | Имя атрибута | Тип данных |
| Код Сотрудника | Id | INTEGER |
| ФИО сотрудника | FullName | VARCHAR |
| Должность | Position | VARCHAR |

В таблице 2.5 перечислены поля отношений «Product».

Таблица 2.5 – Поля отношений «Product»

|  |  |  |
| --- | --- | --- |
| Содержание атрибута | Имя атрибута | Тип данных |
| Код Товара | Id | INTEGER |
| Наименование товара | Name | VARCHAR |
| Категория | Category | VARCHAR |
| Цена | Price | DECIMAL |
| Количество | QuantityInStock | INTEGER |

В таблице 2.6 перечислены поля отношений «InventoryOperation».

Таблица 2.6 – Поля отношений «InventoryOperation»

|  |  |  |
| --- | --- | --- |
| Содержание атрибута | Имя атрибута | Тип данных |
| Код Операции | Id | INTEGER |
| Код Товара | ProductId | VARCHAR |
| Код Сотрудника | EmployeeId | VARCHAR |
| Тип операции | OperationType | ENUM |
| Дата | Date | DATETIME |

Результатом проектирования информационного обеспечения системы является построение логической и физической моделей БД.

## **2.9 Архитектура системы**

Разработка с использованием фреймворка Blazor подводит к использованию шаблона проектирования MVC, суть которого сводится к разделению модели данных, графического интерфейса и бизнес-логики.

На рисунке 2.7 представлена диаграмма размещения системы.



Рисунок 2.7 – Диаграмма размещения системы

На рисунке 2.8 изображена диаграмма компонентов системы.

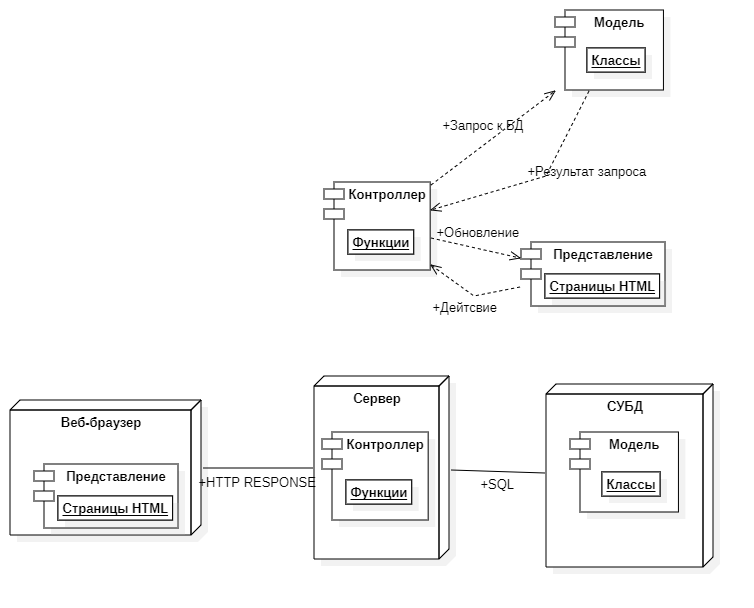


Рисунок 2.8 – Диаграмма компонентов системы

## **2.10 Разработка моделей системы**

Описание таблиц базы данных в фреймворке Blazor происходит через компонент Model. Каждой таблице соответствует модель. Диаграмма классов представлена на рисунке 2.9.

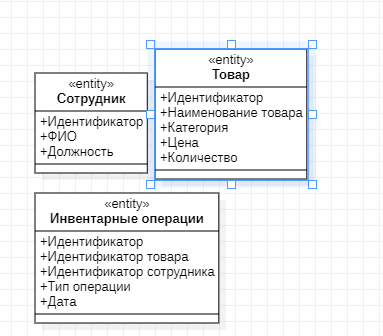


Рисунок 2.9 – Диаграмма классов

Компонент View позволяет отображать контент через шаблоны – HTML-страницы. Не считая страниц с регистрацией и авторизацией всего страниц 8.

Описание страниц производилось через HTML – язык гипертекстовой разметки, CSS – каскадные таблицы стилей.

Диаграмма класса шаблонов продемонстрирована на рисунке 2.10.

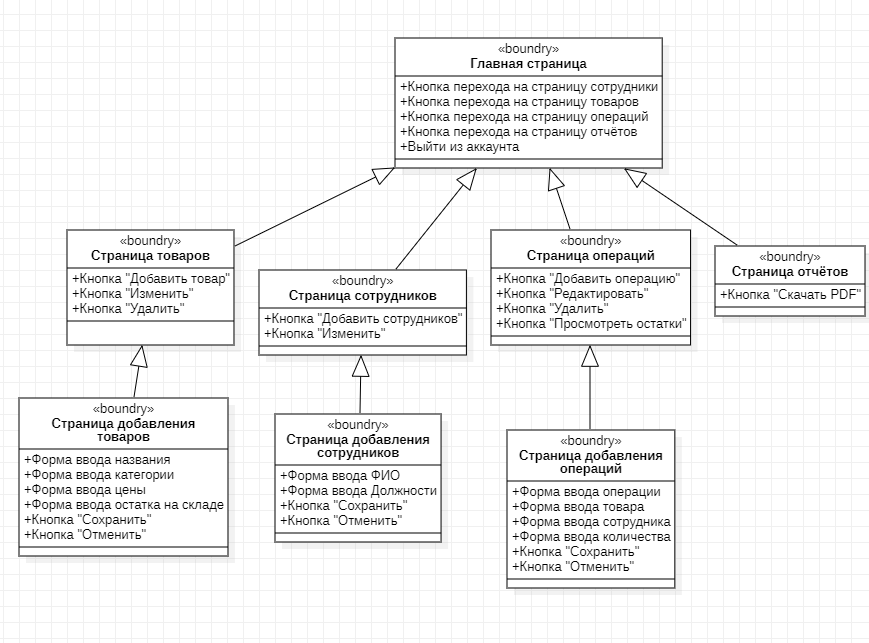


Рисунок 2.10 – Диаграмма класса шаблонов

Через компонент Controller происходит исполнение бизнес-логики посредством взаимодействия с модулями View и Model для отображения шаблонов.

На рисунке 2.11 изображены функции отображения шаблонов.



Рисунок 2.11 – Функции отображения шаблонов

# **3 Описание работы программы**

## **3.1 Общие сведения**

Web-сервис экспорта табличных данных выполнен на языке программирования Python в среде разработки PyCharm 2021 с использованием фрейморка Flask.

Система является приложением, осуществляющую учёт товара.

Чтобы воспользоваться системой необходимо пройти процесс регистрации и входа в систему, как авторизованный пользователь.

## **3.2 Описание пользовательского интерфейса**

При входе в систему пользователя встречает главная страница (рисунок 3.1).

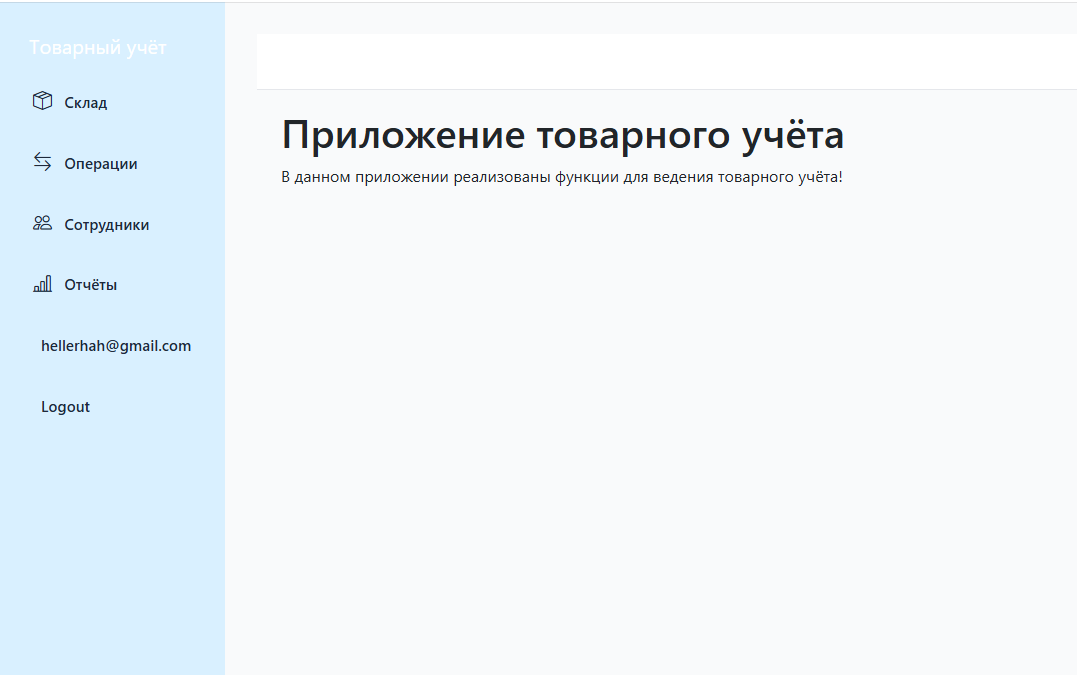


Рисунок 3.1 – Главная страница

Отсюда пользователь может перейти на страницы Склад (рисунок 3.2), Операции (рисунок 3.3), Сотрудники (рисунок 3.4) и Отчёты (рисунок 3.5).

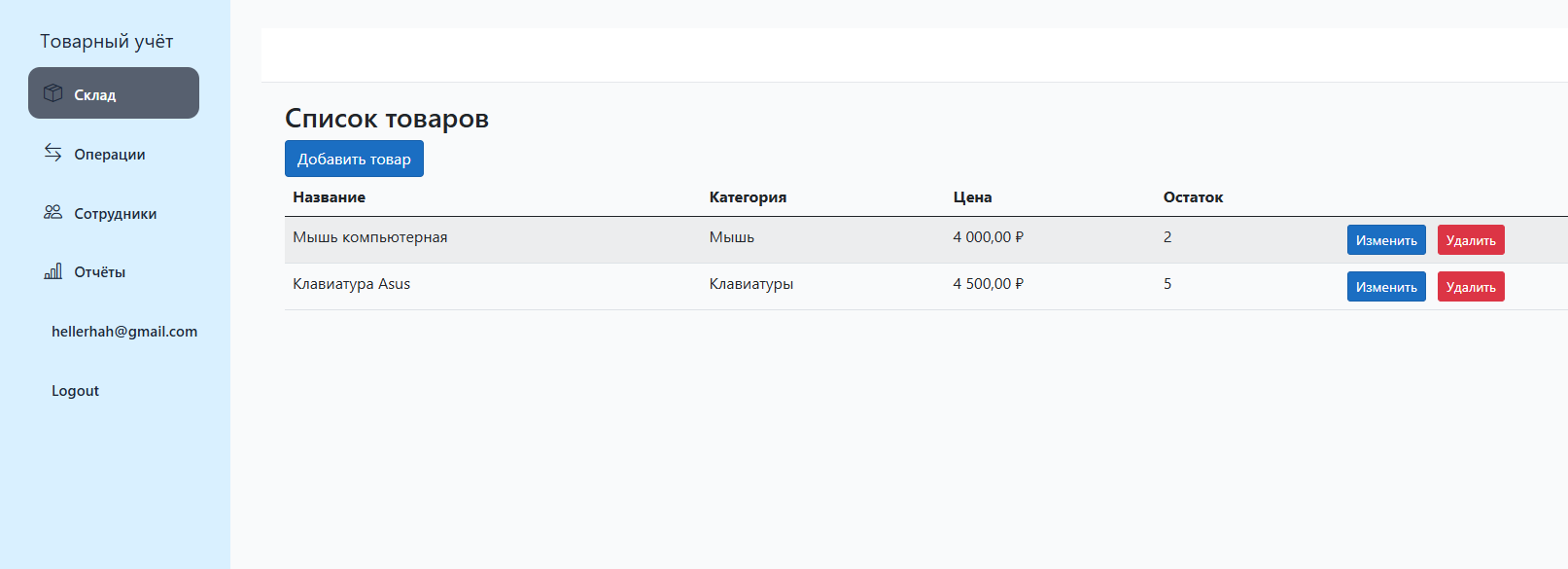


Рисунок 3.2 – Склад

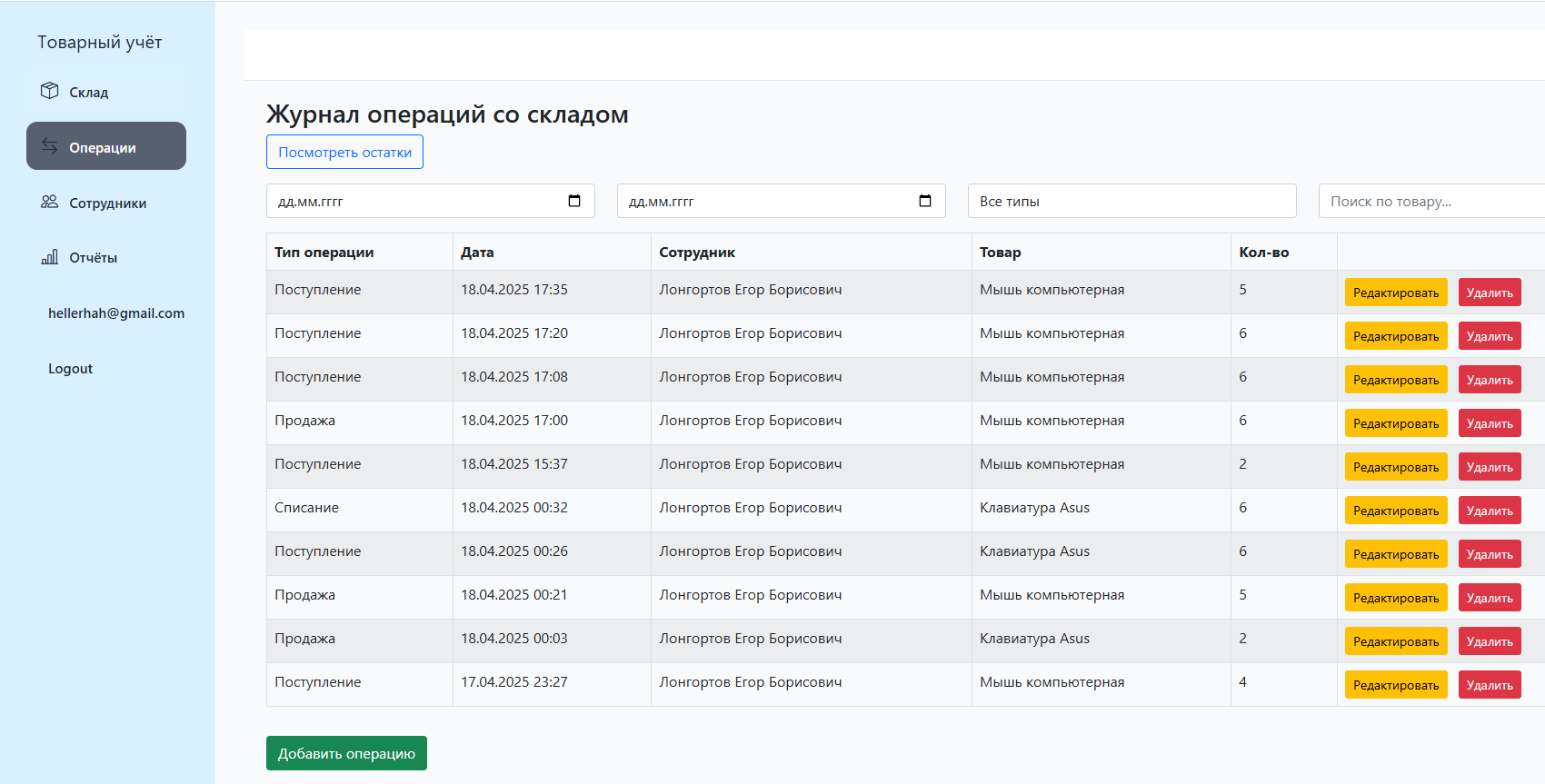


Рисунок 3.3 – Операции

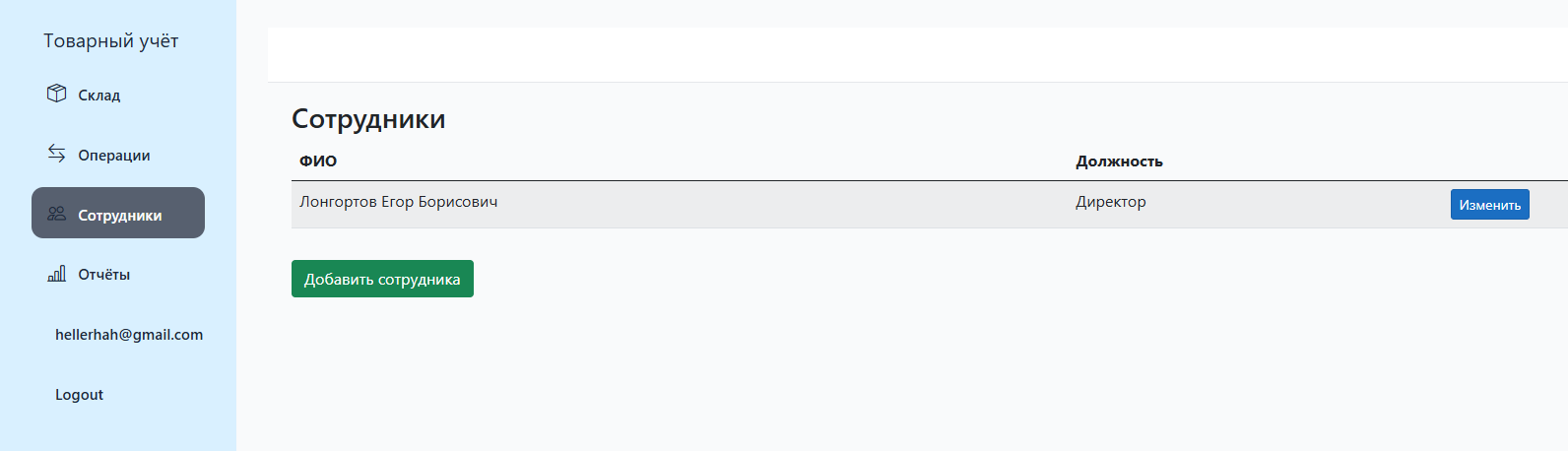


Рисунок 3.4 – Сотрудники

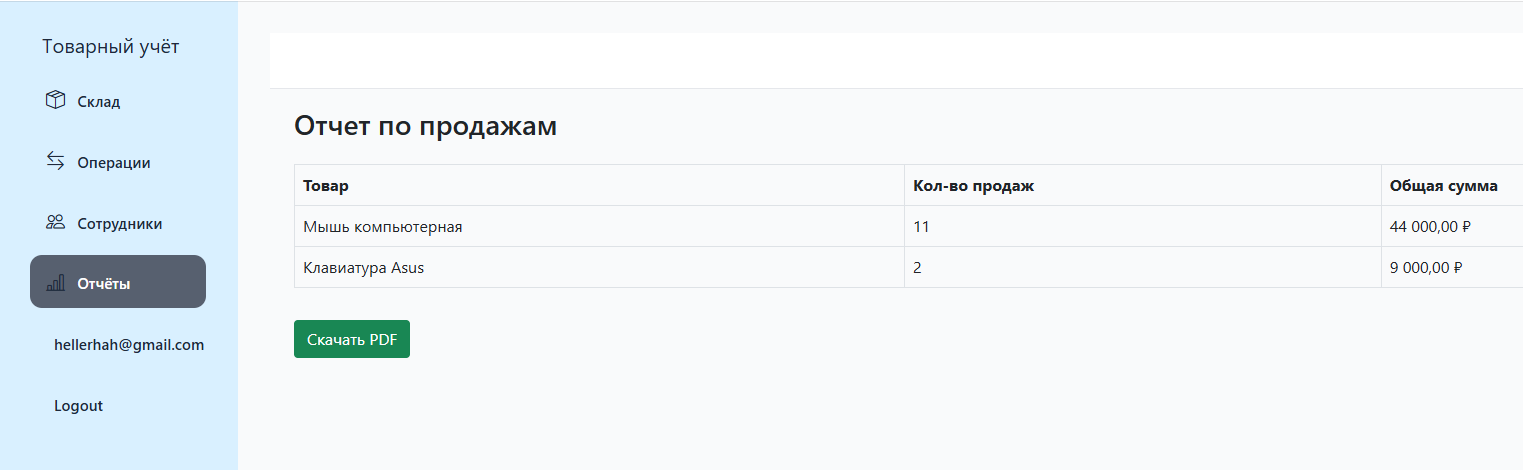


Рисунок 3.5 – Отчёты

На странице склада пользователь может просмотреть список товаров, а также редактировать его, как нажав кнопки «Изменить», «Удалить», так и нажав кнопку «Добавить товар», нажав на которую его переместит на форму ввода данных товара (рисунок 3.6).

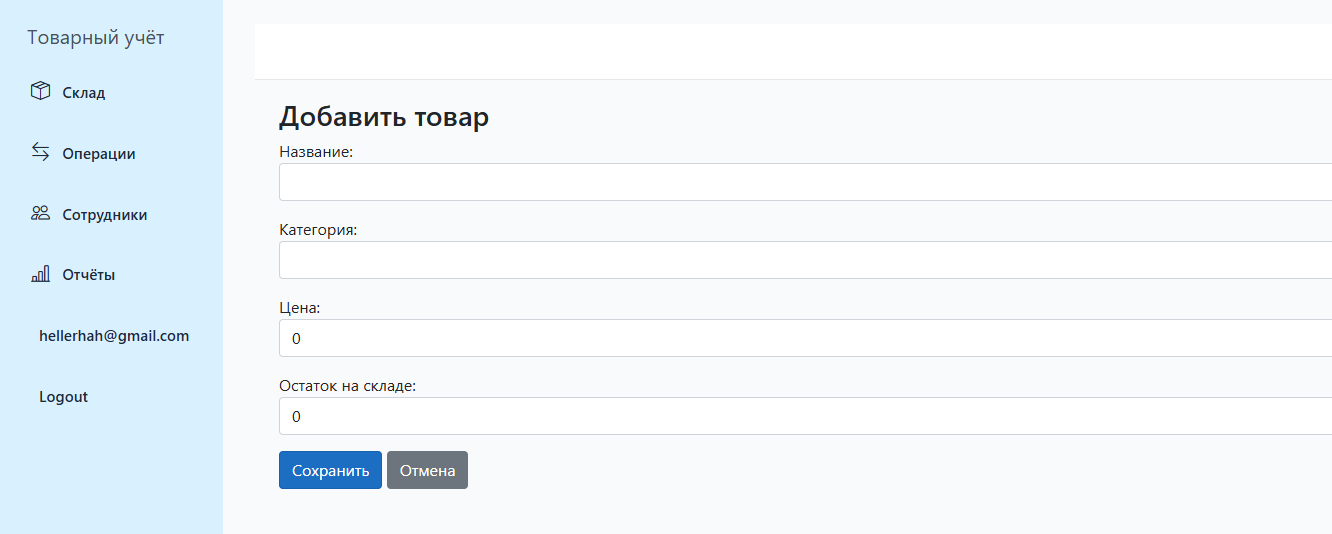


Рисунок 3.6 – Добавление товара

После заполнения форм, необходимо нажать кнопку «Сохранить».

В ходе демонстрации работы был добавлен товар «Монитор Acer» (рисунок 3.7).

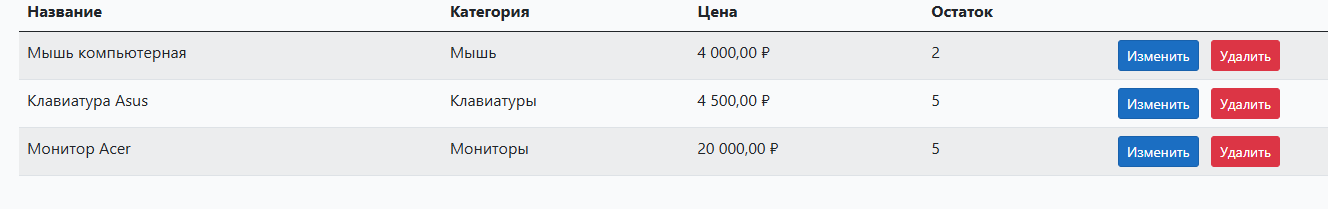


Рисунок 3.7 – Демонстрация добавления товара

При нажатии на кнопку «Изменить» пользователя переведёт на форму аналогичную добавления товара.

На странице операций у пользователя есть возможность добавления новых операций путём нажатия на кнопку «Добавить операцию», в результате чего его переведёт на новую форму (рисунок 3.8).

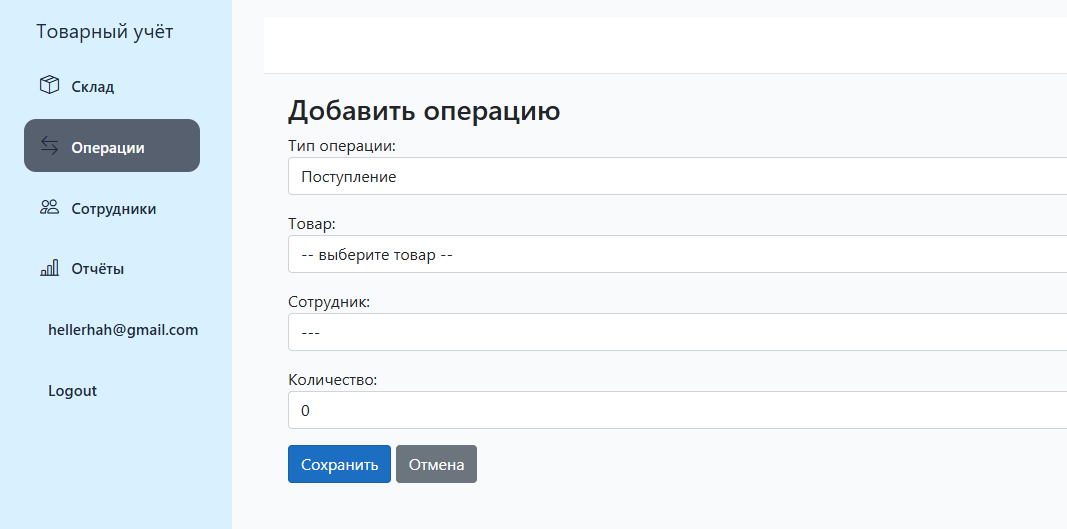


Рисунок 3.8 – Форма добавления операции

Необходимо нажать на кнопку «Сохранить» после заполнения всех форм.

Кнопка «Редактировать» переведёт пользователя на похожую форму.

Также на странице операций есть фильтрация данных. Например, можно ввести название товара (рисунок 3.9).

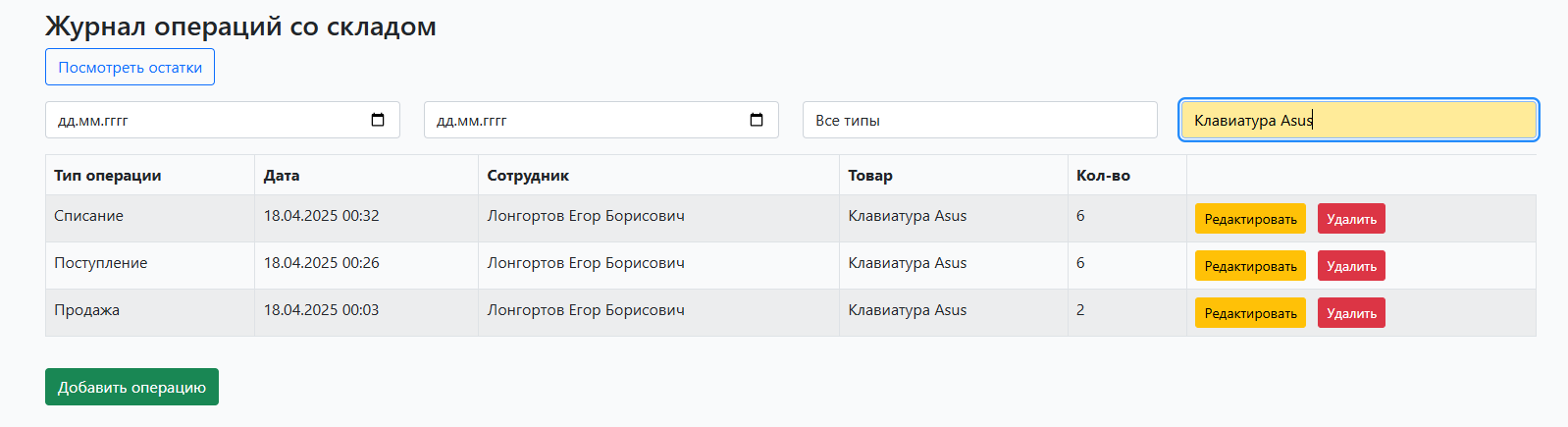


Рисунок 3.9 – Фильтрация по названию

Помимо такой фильтрации, есть фильтрация по типу операции и по дате (рисунок 3.10).

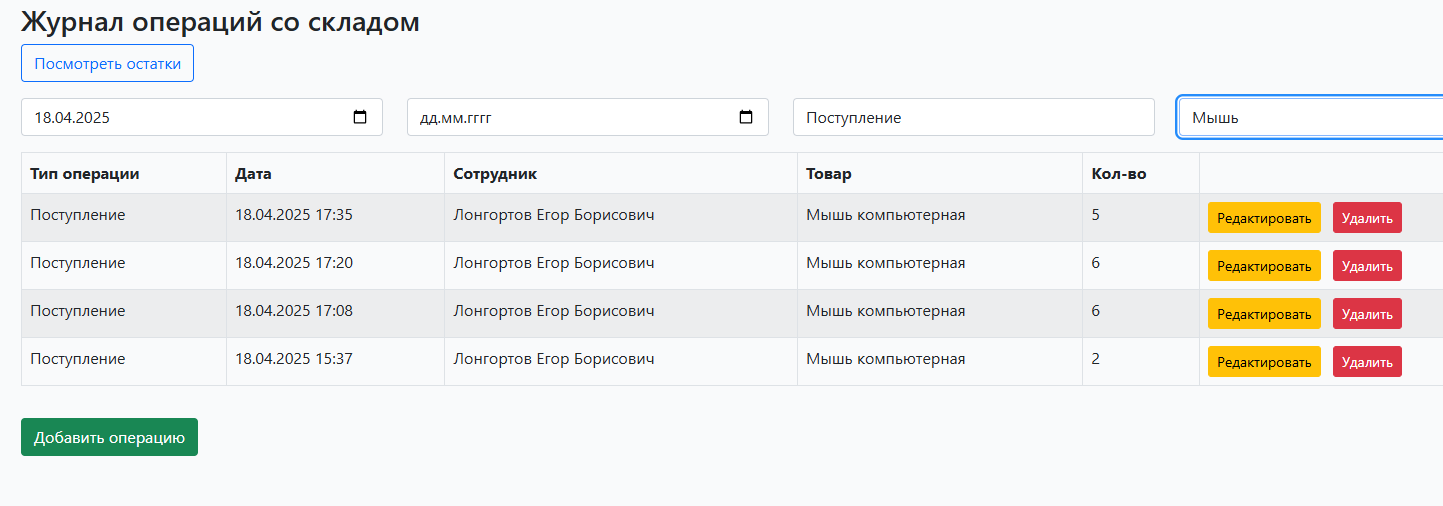


Рисунок 3.10 – Другие типы фильтрации

Кнопка «Посмотреть остатки» покажет пользователю, сколько осталось товаров после продажи (рисунок 3.11).

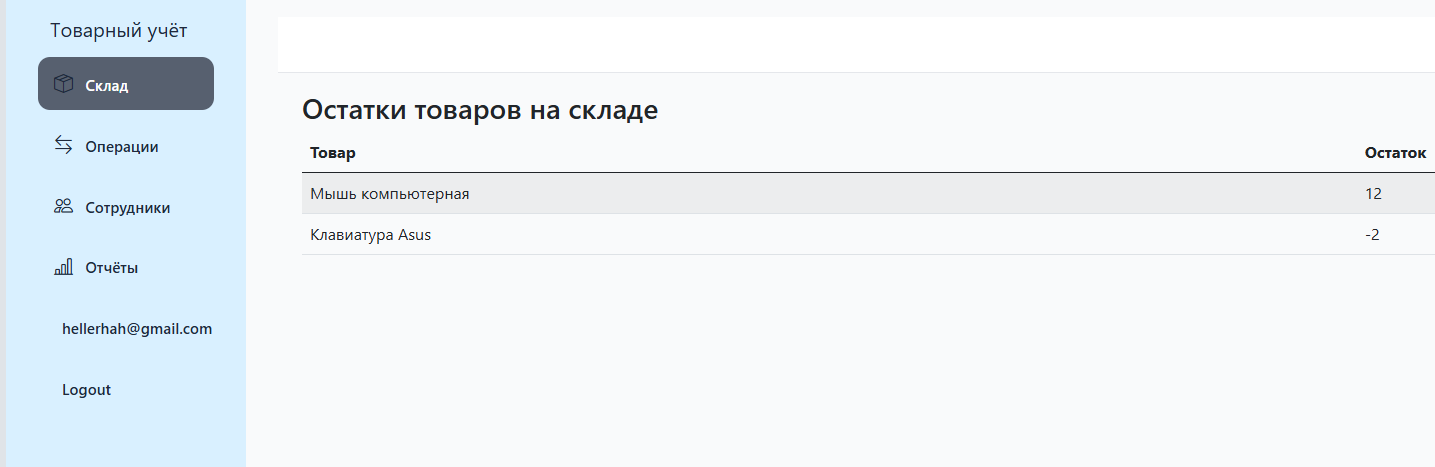


Рисунок 3.11 – Остатки товаров на складе

Кнопка «Добавить сотрудника» переведёт пользователя на страницу с формами ввода данных о новом сотруднике (рисунок 3.12).

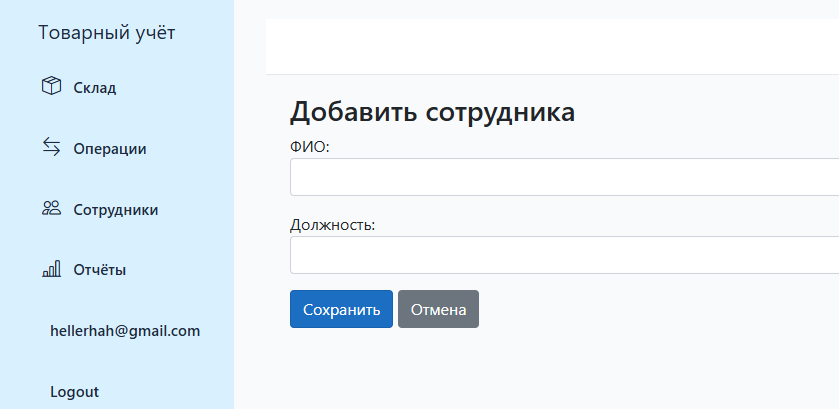


Рисунок 3.12 – Форма ввода нового сотрудника

На странице «Отчёты» есть кнопка «Сохранить PDF», которая сохраняет отчет по продажам в формате PDF-файла (рисунок 3.13).



Рисунок 3.13 – Сохранённый отчёт в формате PDF

Код программы представлен в приложении.

# **ЗАКЛЮЧЕНИЕ**

В рамках курсовой работы была успешно разработана система автоматизированного товарного учёта для розничного магазина, отвечающая современным требованиям к управлению бизнес-процессами. Реализованное приложение решает ключевые задачи, связанные с учётом движения товаров, анализом продаж и формированием отчётности, что подтверждает достижение поставленной цели.

Основные результаты работы включают:

* создание объектно-ориентированной модели данных с классами для описания товаров, сотрудников и операций, обеспечивающей гибкость и масштабируемость системы;
* разработку интуитивного веб-интерфейса на базе технологии Blazor, позволяющего сотрудникам магазина легко вносить данные инвентаризации и отслеживать изменения в режиме реального времени;
* реализацию функционала генерации отчётов по продажам.

Приложение демонстрирует практическую ценность для малого и среднего бизнеса, сокращая временные затраты на рутинные операции и минимизируя риски ошибок при учёте. Использование Blazor в качестве основы для веб-интерфейса обеспечило кросс-платформенность и высокую производительность, что соответствует современным тенденциям в разработке программного обеспечения.

Перспективой развития системы может стать добавление модуля прогнозирования спроса на основе машинного обучения, интеграция с онлайн-кассами и мобильным приложением для сотрудников. Реализованное решение служит фундаментом для дальнейшего расширения функциональности и адаптации под специфические потребности различных торговых предприятий.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. ANK Co., Ltd. HTML/CSS. Вся разработка в схемах и иллюстрациях / ANK Co., Ltd ; пер. с англ. [не указан]. - Санкт-Петербург : Питер, 2025. - 208 с. - ISBN [отсутствует в данных]. - Текст : непосредственный.
2. Бондарь, А.Г. Microsoft SQL Server 2022 : практическое руководство / А.Г. Бондарь. - Санкт-Петербург : БХВ, 2023. - 528 с. - (В подлиннике). - Текст : непосредственный.
3. Максимчук, Р.А. UML для простых смертных : руководство / Р.А. Максимчук, Э.Дж. Нейбург ; пер. с англ. [не указан]. - Москва : Лори, 2024. - 300 с. - ISBN [отсутствует в данных]. - Текст : непосредственный.
4. Наков, С. C#. Основы программирования : учебное пособие / С. Наков, А. Крастев, А. Лазарев. - Москва : АСТ, 2024. - 496 с. - (Полный курс для начинающих программистов). - ISBN 978-5-17-162128-5. - Текст : непосредственный.
5. Прайс, М. C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Прайс ; пер. с англ. [не указан]. - Санкт-Петербург : Питер, 2023. - 848 с. - (Для профессионалов). - ISBN 978-5-4461-2345-9. - Текст : непосредственный.
6. Роббинс, Д. HTML5. Карманный справочник / Д. Роббинс ; пер. с англ. [не указан]. - Москва : Диалектика-Вильямс, 2020. - 192 с. - ISBN [отсутствует в данных]. - Текст : непосредственный.
7. Сэйнти, К. Blazor в действии : практическое руководство / К. Сэйнти ; пер. с англ. [не указан]. - Москва : ДМК Пресс, 2023. - 380 с. - ISBN 978-5-93700-179-5. - Текст : непосредственный.
8. Умрихин, Е.Д. Разработка веб-приложений с помощью Blazor : учебное пособие / Е.Д. Умрихин. - Санкт-Петербург : БХВ, 2025. - 400 с. - (Профессиональное программирование). - ISBN 978-5-9775-1984-7. - Текст : непосредственный.
9. Флэнаган, Д. JavaScript. Полное руководство / Д. Флэнаган ; пер. с англ. [не указан]. - Москва : Диалектика-Вильямс, 2021. - 720 с. - ISBN [отсутствует в данных]. - Текст : непосредственный.
10. Шилдс, У. SQL: быстрое погружение / У. Шилдс ; пер. с англ. [не указан]. - Санкт-Петербург : Питер, 2022. - 224 с. - (Библиотека программиста). - ISBN [отсутствует в данных]. - Текст : непосредственный.

# *Приложение*

Файл Program.cs:

using Microsoft.AspNetCore.Components.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using QuestPDF.Infrastructure;

using TovanyUchetV2.Components;

using TovanyUchetV2.Components.Account;

using TovanyUchetV2.Data;

using TovanyUchetV2.Services;

namespace TovanyUchetV2

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorComponents()

.AddInteractiveServerComponents();

builder.Services.AddCascadingAuthenticationState();

builder.Services.AddControllers();

builder.Services.AddScoped<IdentityUserAccessor>();

builder.Services.AddScoped<IdentityRedirectManager>();

builder.Services.AddScoped<AuthenticationStateProvider, IdentityRevalidatingAuthenticationStateProvider>();

builder.Services.AddAuthentication(options =>

{

options.DefaultScheme = IdentityConstants.ApplicationScheme;

options.DefaultSignInScheme = IdentityConstants.ExternalScheme;

})

.AddIdentityCookies();

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddIdentityCore<ApplicationUser>(options => options.SignIn.RequireConfirmedAccount = true)

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddSignInManager()

.AddDefaultTokenProviders();

builder.Services.AddSingleton<IEmailSender<ApplicationUser>, IdentityNoOpEmailSender>();

builder.Services.AddScoped<IDataService, MSSQLDataService>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseMigrationsEndPoint();

*Продолжение приложения*

}

else

{

app.UseExceptionHandler("/Error");

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseAntiforgery();

app.MapControllers();

app.MapRazorComponents<App>()

.AddInteractiveServerRenderMode();

// Add additional endpoints required by the Identity /Account Razor components.

app.MapAdditionalIdentityEndpoints();

QuestPDF.Settings.License = LicenseType.Community;

app.Run();

}

}

}

Файл appsettings.json:

{

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-TovanyUchetV2-77a722ba-4e80-427a-9555-ce4aa0522983;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

Файл ApplicationDbContext.cs:

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using TovanyUchetV2.Data.Models;

namespace TovanyUchetV2.Data

{

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Product> Products { get; set; } = null!;

public DbSet<Employee> Employees { get; set; } = null!;

public DbSet<InventoryOperation> InventoryOperations { get; set; } = null!;

public DbSet<AuditLog> AuditLogs => Set<AuditLog>();

*Продолжение приложения*

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<InventoryOperation>()

.HasOne(op => op.Product)

.WithMany()

.HasForeignKey(op => op.ProductId)

.OnDelete(DeleteBehavior.Cascade); // Каскадное удаление

modelBuilder.Entity<InventoryOperation>()

.HasOne(op => op.Employee)

.WithMany()

.HasForeignKey(op => op.EmployeeId)

.OnDelete(DeleteBehavior.SetNull); // Установить NULL при удалении сотрудника

modelBuilder.Entity<Product>()

.Property(p => p.Price)

.HasPrecision(18, 2); // 18 всего знаков, из них 2 после запятой

}

}

Файл MSSQLDataService.cs:

using Microsoft.EntityFrameworkCore;

using TovanyUchetV2.Data;

using TovanyUchetV2.Data.Models;

namespace TovanyUchetV2.Services

{

public class MSSQLDataService : IDataService

{

private readonly ApplicationDbContext \_db;

public MSSQLDataService(ApplicationDbContext db)

{

\_db = db;

}

public MSSQLDataService()

{

}

public async Task<List<InventoryOperation>> GetAllInventoryOperationsAsync()

{

return await \_db.InventoryOperations

.Include(op => op.Product)

.Include(op => op.Employee)

.OrderByDescending(op => op.Date)

.ToListAsync();

}

public async Task<List<Product>> GetAllProductsAsync()

{

return await \_db.Products.ToListAsync();

}

public async Task<Product?> GetProductByIdAsync(int id)

{

return await \_db.Products.FindAsync(id);

}

*Продолжение приложения*

public async Task AddOrUpdateProductAsync(Product product)

{

if (product.Id == 0)

\_db.Products.Add(product);

else

\_db.Products.Update(product);

await \_db.SaveChangesAsync();

}

public async Task DeleteProductAsync(int id)

{

var product = await \_db.Products.FindAsync(id);

if (product != null)

{

\_db.Products.Remove(product);

await \_db.SaveChangesAsync();

}

}

public async Task<List<Employee>> GetAllEmployeesAsync()

{

return await \_db.Employees.ToListAsync(); // Добавлено получение сотрудников

}

public async Task AddInventoryOperationAsync(InventoryOperation operation)

{

// Проверка, что товар существует

var product = await \_db.Products.FindAsync(operation.ProductId);

if (product == null)

throw new ArgumentException("Товар не найден");

\_db.InventoryOperations.Add(operation);

// Аудит

\_db.AuditLogs.Add(new AuditLog

{

Action = "Добавление",

Entity = $"Operation (ProductId: {operation.ProductId}, Qty: {operation.Quantity})",

Date = DateTime.Now,

User = "admin" // если авторизация, бери из контекста

});

\_db.InventoryOperations.Add(operation);

await \_db.SaveChangesAsync();

}

public async Task<Employee?> GetEmployeeByIdAsync(int id)

{

return await \_db.Employees.FindAsync(id);

}

public async Task AddOrUpdateEmployeeAsync(Employee employee)

{

if (employee.Id == 0)

{

\_db.Employees.Add(employee);

}

else

{

\_db.Employees.Update(employee);

*Продолжение приложения*

}

await \_db.SaveChangesAsync();

}

public async Task DeleteInventoryOperationAsync(int id)

{

var op = await \_db.InventoryOperations.FindAsync(id);

if (op != null)

{

\_db.InventoryOperations.Remove(op);

await \_db.SaveChangesAsync();

}

}

public async Task<InventoryOperation?> GetInventoryOperationByIdAsync(int id)

{

return await \_db.InventoryOperations

.Include(o => o.Product)

.Include(o => o.Employee)

.FirstOrDefaultAsync(o => o.Id == id);

}

public async Task UpdateInventoryOperationAsync(InventoryOperation operation)

{

\_db.InventoryOperations.Update(operation);

await \_db.SaveChangesAsync();

}

}

}

Файл SalesReportDocument.cs:

using QuestPDF.Fluent;

using QuestPDF.Helpers;

using QuestPDF.Infrastructure;

using static TovanyUchetV2.Components.Pages.SalesReports;

public class SalesReportDocument : IDocument

{

private readonly List<SalesReportRow> \_report;

public SalesReportDocument(List<SalesReportRow> report)

{

\_report = report;

}

public DocumentMetadata GetMetadata() => DocumentMetadata.Default;

public void Compose(IDocumentContainer container)

{

container.Page(page =>

{

page.Margin(20);

page.Content()

.Table(table =>

{

table.ColumnsDefinition(columns =>

{

columns.RelativeColumn();

columns.ConstantColumn(100);

columns.ConstantColumn(100);

*Продолжение приложения*

});

table.Header(header =>

{

header.Cell().Element(CellStyle).Text("Товар");

header.Cell().Element(CellStyle).Text("Кол-во");

header.Cell().Element(CellStyle).Text("Сумма");

});

foreach (var row in \_report)

{

table.Cell().Element(CellStyle).Text(row.ProductName);

table.Cell().Element(CellStyle).Text(row.TotalQuantity.ToString());

table.Cell().Element(CellStyle).Text($"{row.TotalSum:C}");

}

static IContainer CellStyle(IContainer container) =>

container.Padding(5).BorderBottom(1).BorderColor(Colors.Grey.Lighten2);

});

});

}

}

Файл Employee.cs:

namespace TovanyUchetV2.Data.Models

{

public class Employee

{

public int Id { get; set; }

public string FullName { get; set; } = "";

public string Position { get; set; } = "";

}

}

Файл InventoryOperation.cs :

using System.ComponentModel.DataAnnotations;

namespace TovanyUchetV2.Data.Models

{

public enum OperationType

{

Purchase, // поступление

Sale, // продажа

WriteOff, // списание

Return // возврат

}

public class InventoryOperation

{

public int Id { get; set; }

[Required(ErrorMessage = "Товар обязателен")]

[Range(1, int.MaxValue, ErrorMessage = "Выберите товар")]

public int ProductId { get; set; }

public Product Product { get; set; } = null!;

[Required(ErrorMessage = "Количество обязательно")]

[Range(1, int.MaxValue, ErrorMessage = "Минимум 1 единица")]

public int Quantity { get; set; }

*Продолжение приложения*

[Required(ErrorMessage = "Тип операции обязателен")]

public OperationType OperationType { get; set; }

[Required(ErrorMessage = "Дата операции обязательна")]

public DateTime Date { get; set; }

public int? EmployeeId { get; set; }

public Employee? Employee { get; set; }

public string? UserName { get; set; }

}

}

Файл Prodcut.cs:

namespace TovanyUchetV2.Data.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = "";

public string Category { get; set; } = "";

public decimal Price { get; set; }

public int QuantityInStock { get; set; }

}

}

Файл IDataService.cs:

using TovanyUchetV2.Data.Models;

public interface IDataService

{

Task<List<Product>> GetAllProductsAsync();

Task<Product?> GetProductByIdAsync(int id);

Task AddOrUpdateProductAsync(Product product);

Task DeleteProductAsync(int id);

Task<List<Employee>> GetAllEmployeesAsync();

Task<Employee?> GetEmployeeByIdAsync(int id);

Task AddOrUpdateEmployeeAsync(Employee employee);

Task<List<InventoryOperation>> GetAllInventoryOperationsAsync();

Task AddInventoryOperationAsync(InventoryOperation operation);

Task DeleteInventoryOperationAsync(int id);

Task<InventoryOperation?> GetInventoryOperationByIdAsync(int id);

Task UpdateInventoryOperationAsync(InventoryOperation operation);

}

Файл ReportsController.cs:

// Controllers/ReportsController.cs

using Microsoft.AspNetCore.Mvc;

using QuestPDF.Fluent;

using TovanyUchetV2.Data.Models;

using static TovanyUchetV2.Components.Pages.SalesReports; // Для SalesReportRow

[Route("api/reports")]

[ApiController]

public class ReportsController : ControllerBase

{

private readonly IDataService \_dataService;

public ReportsController(IDataService dataService)

*Продолжение приложения*

{

\_dataService = dataService;

}

[HttpGet("sales/pdf")]

public async Task<IActionResult> GetSalesReportPdf()

{

var sales = await \_dataService.GetAllInventoryOperationsAsync();

var report = sales

.Where(o => o.OperationType == OperationType.Sale)

.GroupBy(o => o.Product)

.Select(g => new SalesReportRow

{

ProductName = g.Key.Name,

TotalQuantity = g.Sum(x => x.Quantity),

TotalSum = g.Sum(x => x.Quantity \* x.Product.Price)

})

.ToList();

var doc = new SalesReportDocument(report);

var pdfBytes = doc.GeneratePdf();

return File(pdfBytes, "application/pdf", $"SalesReport\_{DateTime.Now:yyyyMMdd\_HHmm}.pdf");

}

}

Файл \_Imports.razor:

@using System.Net.Http

@using System.Net.Http.Json

@using Microsoft.AspNetCore.Components.Authorization

@using Microsoft.AspNetCore.Components.Forms

@using Microsoft.AspNetCore.Components.Routing

@using Microsoft.AspNetCore.Components.Web

@using static Microsoft.AspNetCore.Components.Web.RenderMode

@using Microsoft.AspNetCore.Components.Web.Virtualization

@using Microsoft.JSInterop

@using TovanyUchetV2

@using TovanyUchetV2.Components

@using TovanyUchetV2.Data

@using TovanyUchetV2.Data.Models

@using Microsoft.EntityFrameworkCore

@using TovanyUchetV2.Services

@using Microsoft.AspNetCore.Authorization

Файл Inventory.razor:

@page "/inventory"

@inject IDataService DbService

@inject NavigationManager navigate

@rendermode InteractiveServer

@attribute [Authorize]

<h3>Список товаров</h3>

@if (products == null)

{

<p><em>Загрузка...</em></p>

}

else if (!products.Any())

{

<button class="btn btn-primary" @onclick="AddProduct">Добавить товар</button>

<p>Товары не найдены.</p>

}

else

*Продолжение приложения*

{

<button class="btn btn-primary" @onclick="AddProduct">Добавить товар</button>

<table class="table table-striped">

<thead>

<tr>

<th>Название</th>

<th>Категория</th>

<th>Цена</th>

<th>Остаток</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var product in products)

{

<tr>

<td>@product.Name</td>

<td>@product.Category</td>

<td>@product.Price.ToString("C")</td>

<td>@product.QuantityInStock</td>

<td>

<button class="btn btn-sm btn-primary" @onclick="@(() => EditProduct(product.Id))">Изменить</button>

<button class="btn btn-sm btn-danger ms-2" @onclick="@(() => DeleteProduct(product.Id))">Удалить</button>

</td>

</tr>

}

</tbody>

</table>

}

@code {

private List<Product>? products;

protected override async Task OnInitializedAsync()

{

products = await DbService.GetAllProductsAsync();

}

private void EditProduct(int id)

{

navigate.NavigateTo($"/editinventory/{id}");

}

private void AddProduct()

{

Console.WriteLine("Навигация выполняется!"); // Проверьте вывод в консоли

navigate.NavigateTo("editinventory/add");

}

private async Task DeleteProduct(int id)

{

await DbService.DeleteProductAsync(id);

products = await DbService.GetAllProductsAsync(); // обновляем список

}

}

Файл InventoryOpeation.razor:

@page "/inventoryoperation"

@inject IDataService DbService

@inject NavigationManager navigate

@rendermode InteractiveServer

*Продолжение приложения*

@attribute [Authorize]

<h3>Журнал операций со складом</h3>

<a href="inventory/stock" class="btn btn-outline-primary mb-3">Посмотреть остатки</a>

<div class="row mb-3">

<div class="col">

<**InputDate** @bind-Value="fromDate" class="form-control" placeholder="С..." />

</div>

<div class="col">

<**InputDate** @bind-Value="toDate" class="form-control" placeholder="По..." />

</div>

<div class="col">

<**InputSelect** @bind-Value="selectedType" class="form-control">

<option value="">Все типы</option>

@foreach (var type in Enum.GetValues<OperationType>())

{

<option value="@type">@GetOperationTypeName(type)</option>

}

</**InputSelect**>

</div>

<div class="col">

<**InputText** @bind-Value="searchText" class="form-control" placeholder="Поиск по товару..." />

</div>

</div>

@if (filteredOperations == null)

{

<p><em>Загрузка...</em></p>

}

else if (!filteredOperations.Any())

{

<p>Операции не найдены.</p>

}

else

{

<table class="table table-bordered table-striped">

<thead>

<tr>

<th>Тип операции</th>

<th>Дата</th>

<th>Сотрудник</th>

<th>Товар</th>

<th>Кол-во</th>

</tr>

</thead>

<tbody>

@foreach (var op in filteredOperations)

{

<tr>

<td>@GetOperationTypeName(op.OperationType)</td>

<td>@op.Date.ToString("dd.MM.yyyy HH:mm")</td>

<td>@op.Employee?.FullName</td>

<td>@op.Product?.Name</td>

<td>@op.Quantity</td>

<td>

<button class="btn btn-sm btn-warning me-2" @onclick='() => navigate.NavigateTo($"/inventoryoperation/edit/{op.Id}")'>Редактировать</button>

<button class="btn btn-sm btn-danger" @onclick="() => Delete(op.Id)">Удалить</button>

*Продолжение приложения*

</td>

</tr>

}

</tbody>

</table>

}

<button class="btn btn-success mt-3" @onclick='() => navigate.NavigateTo("/inventoryoperation/add")'>

Добавить операцию

</button>

@code {

private List<TovanyUchetV2.Data.Models.InventoryOperation>? operations;

private string searchText = "";

private string? selectedType = null;

private DateTime? fromDate = null;

private DateTime? toDate = null;

private IEnumerable<TovanyUchetV2.Data.Models.InventoryOperation> filteredOperations =>

operations == null ? Enumerable.Empty<TovanyUchetV2.Data.Models.InventoryOperation>() :

operations

.Where(o => string.IsNullOrWhiteSpace(searchText) ||

(o.Product?.Name?.Contains(searchText, StringComparison.OrdinalIgnoreCase) ?? false))

.Where(o => string.IsNullOrEmpty(selectedType) || o.OperationType.ToString() == selectedType)

.Where(o => !fromDate.HasValue || o.Date >= fromDate.Value)

.Where(o => !toDate.HasValue || o.Date <= toDate.Value);

protected override async Task OnInitializedAsync()

{

operations = await DbService.GetAllInventoryOperationsAsync();

}

private async Task Delete(int id)

{

await DbService.DeleteInventoryOperationAsync(id);

operations = await DbService.GetAllInventoryOperationsAsync(); // обновим список

}

private string GetOperationTypeName(OperationType type) =>

type switch

{

OperationType.Purchase => "Поступление",

OperationType.Sale => "Продажа",

OperationType.WriteOff => "Списание",

OperationType.Return => "Возврат",

\_ => type.ToString()

};

}

Файл EmployeeList.razor:

@page "/employees"

*Продолжение приложения*

@inject IDataService DataService

@inject NavigationManager NavManager

@rendermode InteractiveServer

@attribute [Authorize]

<h3>Сотрудники</h3>

@if (employees == null)

{

<p><em>Загрузка...</em></p>

}

else if (!employees.Any())

{

<p>Сотрудники не найдены.</p>

}

else

{

<table class="table table-striped">

<thead>

<tr>

<th>ФИО</th>

<th>Должность</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var e in employees)

{

<tr>

<td>@e.FullName</td>

<td>@e.Position</td>

<td>

<button class="btn btn-sm btn-primary" @onclick="() => EditEmployee(e.Id)">Изменить</button>

</td>

</tr>

}

</tbody>

</table>

}

<button class="btn btn-success mt-3" @onclick="AddEmployee">Добавить сотрудника</button>

@code {

private List<Employee> employees = new();

protected override async Task OnInitializedAsync()

{

employees = await DataService.GetAllEmployeesAsync();

}

private void AddEmployee()

{

NavManager.NavigateTo("/employee/add");

}

private void EditEmployee(int id)

{

NavManager.NavigateTo($"/employee/{id}");

}

}

Файл SalesReports.razor:

*Продолжение приложения*

@page "/sales"

@inject IDataService DataService

@attribute [Authorize]

<h3>Отчет по продажам</h3>

@if (report == null)

{

<p>Загрузка...</p>

}

else

{

<table class="table table-bordered mt-4">

<thead>

<tr>

<th>Товар</th>

<th>Кол-во продаж</th>

<th>Общая сумма</th>

</tr>

</thead>

<tbody>

@foreach (var row in report)

{

<tr>

<td>@row.ProductName</td>

<td>@row.TotalQuantity</td>

<td>@row.TotalSum.ToString("C")</td>

</tr>

}

</tbody>

</table>

<a class="btn btn-success mt-3" href="/api/reports/sales/pdf" target="\_blank">Скачать PDF</a>

}

@code {

private List<SalesReportRow>? report;

public class SalesReportRow

{

public string ProductName { get; set; } = "";

public int TotalQuantity { get; set; }

public decimal TotalSum { get; set; }

}

protected override async Task OnInitializedAsync()

{

await LoadData();

}

private async Task LoadData()

{

var sales = await DataService.GetAllInventoryOperationsAsync();

report = sales

.Where(o => o.OperationType == OperationType.Sale)

.GroupBy(o => o.Product)

.Select(g => new SalesReportRow

{

ProductName = g.Key.Name,

TotalQuantity = g.Sum(x => x.Quantity),

TotalSum = g.Sum(x => x.Quantity \* x.Product.Price)

})

*Продолжение приложения*

.ToList();

StateHasChanged();

}

}

Файл EditInventory.razor:

@page "/editinventory/add"

@page "/editinventory/{ProductId:int}"

@inject IDataService DataService

@inject NavigationManager NavManager

@rendermode InteractiveServer

@attribute [Authorize]

<h3>@(IsEditMode ? "Редактировать товар" : "Добавить товар")</h3>

<**EditForm** **Model**="@product" **OnValidSubmit**="Save">

<**DataAnnotationsValidator** />

<FormComponents.ValidationSummary />

<FormComponents.ValidationMessage For="() => operation.ProductId" />

<div class="mb-3">

<label>Название:</label>

<**InputText** class="form-control" @bind-Value="product.Name" />

<**ValidationMessage** **For**="() => product.Name" />

</div>

<div class="mb-3">

<label>Категория:</label>

<**InputText** class="form-control" @bind-Value="product.Category" />

</div>

<div class="mb-3">

<label>Цена:</label>

<**InputNumber** class="form-control" @bind-Value="product.Price" />

<**ValidationMessage** **For**="() => product.Price" />

</div>

<div class="mb-3">

<label>Остаток на складе:</label>

<**InputNumber** class="form-control" @bind-Value="product.QuantityInStock" />

<**ValidationMessage** **For**="() => product.QuantityInStock" />

</div>

<button type="submit" class="btn btn-primary">Сохранить</button>

<button type="button" class="btn btn-secondary" @onclick='() => NavManager.NavigateTo("/inventory")'>

Отмена

</button>

</**EditForm**>

@code {

[Parameter]

public int? ProductId { get; set; }

private Product product = new();

private bool IsEditMode => ProductId.HasValue;

protected override async Task OnInitializedAsync()

{

if (IsEditMode && ProductId != null)

{

product = await DataService.GetProductByIdAsync(ProductId.Value) ?? new();

}

}

private async Task Save()

*Продолжение приложения*

{

await DataService.AddOrUpdateProductAsync(product);

NavManager.NavigateTo("/inventory");

}

}

Файл AddInventoryOperation.razor:

@page "/inventoryoperation/add"

@using Microsoft.AspNetCore.Identity

@using TovanyUchetV2.Data.Models

@inject IDataService DataService

@inject NavigationManager NavManager

@inject AuthenticationStateProvider AuthProvider

@inject UserManager<ApplicationUser> UserManager

@rendermode InteractiveServer

<h3>Добавить операцию</h3>

<**EditForm** **Model**="@operation" **OnValidSubmit**="Save">

<**DataAnnotationsValidator** />

<**ValidationSummary** />

<div class="mb-3">

<label>Тип операции:</label>

<**InputSelect** class="form-control" @bind-Value="operation.OperationType">

@foreach (var type in Enum.GetValues<OperationType>())

{

<option value="@type">@GetOperationTypeName(type)</option>

}

</**InputSelect**>

<**ValidationMessage** **For**="() => operation.OperationType" />

</div>

<div class="mb-3">

<label>Товар:</label>

<**InputSelect** class="form-control" @bind-Value="operation.ProductId">

<option value="0">-- выберите товар --</option>

@foreach (var p in products)

{

<option value="@p.Id">@p.Name</option>

}

</**InputSelect**>

<**ValidationMessage** **For**="() => operation.ProductId" />

</div>

<div class="mb-3">

<label>Сотрудник:</label>

<**InputSelect** class="form-control" @bind-Value="operation.EmployeeId">

<option value="@((int?)null)">---</option>

@foreach (var e in employees)

{

<option value="@e.Id">@e.FullName</option>

}

</**InputSelect**>

</div>

<div class="mb-3">

<label>Количество:</label>

<**InputNumber** class="form-control" @bind-Value="operation.Quantity" />

<**ValidationMessage** **For**="() => operation.Quantity" />

</div>

<button type="submit" class="btn btn-primary">Сохранить</button>

*Продолжение приложения*

<button type="button" class="btn btn-secondary" @onclick='() => NavManager.NavigateTo("/inventoryoperation")'>

Отмена

</button>

</**EditForm**>

@code {

private TovanyUchetV2.Data.Models.InventoryOperation operation = new() { Date = DateTime.Now };

private List<Product> products = new();

private List<Employee> employees = new();

protected override async Task OnInitializedAsync()

{

// Загружаем товары и сотрудников

products = await DataService.GetAllProductsAsync();

employees = await DataService.GetAllEmployeesAsync();

// Очищаем автоматическую привязку — пусть пользователь сам выбирает сотрудника

operation.UserName = ""; // или можно вообще не трогать, если поле не обязательно

}

private async Task Save()

{

operation.Date = DateTime.Now;

try

{

await DataService.AddInventoryOperationAsync(operation);

Console.WriteLine("✅ Операция успешно добавлена пользователем " + operation.UserName);

NavManager.NavigateTo("/inventoryoperation");

}

catch (Exception ex)

{

Console.WriteLine("❌ Ошибка при сохранении: " + ex.Message);

}

}

private string GetOperationTypeName(OperationType type) =>

type switch

{

OperationType.Purchase => "Поступление",

OperationType.Sale => "Продажа",

OperationType.WriteOff => "Списание",

OperationType.Return => "Возврат",

\_ => type.ToString()

};

}

Файл EditInventoryOperation.razor:

@page "/inventoryoperation/edit/{Id:int}"

@inject IDataService DataService

@inject NavigationManager NavManager

@rendermode InteractiveServer

@attribute [Authorize]

<h3>Редактировать операцию</h3>

*Продолжение приложения*

<**EditForm** **Model**="@operation" **OnValidSubmit**="Save">

<**DataAnnotationsValidator** />

<**ValidationSummary** />

<div class="mb-3">

<label>Тип операции:</label>

<**InputSelect** class="form-control" @bind-Value="operation.OperationType">

@foreach (var type in Enum.GetValues<OperationType>())

{

<option value="@type">@type</option>

}

</**InputSelect**>

<**ValidationMessage** **For**="() => operation.OperationType" />

</div>

<div class="mb-3">

<label>Товар:</label>

<**InputSelect** class="form-control" @bind-Value="operation.ProductId">

<option value="0">-- выберите товар --</option>

@foreach (var p in products)

{

<option value="@p.Id">@p.Name</option>

}

</**InputSelect**>

<**ValidationMessage** **For**="() => operation.ProductId" />

</div>

<div class="mb-3">

<label>Сотрудник:</label>

<**InputSelect** class="form-control" @bind-Value="operation.EmployeeId">

<option value="@((int?)null)">---</option>

@foreach (var e in employees)

{

<option value="@e.Id">@e.FullName</option>

}

</**InputSelect**>

</div>

<div class="mb-3">

<label>Количество:</label>

<**InputNumber** class="form-control" @bind-Value="operation.Quantity" />

<**ValidationMessage** **For**="() => operation.Quantity" />

</div>

<button type="submit" class="btn btn-primary">Сохранить</button>

<button type="button" class="btn btn-secondary" @onclick='() => NavManager.NavigateTo("/inventoryoperation")'>

Отмена

</button>

</**EditForm**>

@code {

private TovanyUchetV2.Data.Models.InventoryOperation operation = new() { Date = DateTime.Now };

private List<Product> products = new();

private List<Employee> employees = new();

[Parameter] public int Id { get; set; }

protected override async Task OnInitializedAsync()

{

*Продолжение приложения*

products = await DataService.GetAllProductsAsync();

employees = await DataService.GetAllEmployeesAsync();

var existing = await DataService.GetInventoryOperationByIdAsync(Id);

if (existing != null)

operation = existing;

}

private async Task Save()

{

await DataService.UpdateInventoryOperationAsync(operation);

NavManager.NavigateTo("/inventoryoperation");

}

}

Файл EditEmployee.razor:

@page "/employee/add"

@page "/employee/{EmployeeId:int}"

@inject IDataService DataService

@inject NavigationManager NavManager

@rendermode InteractiveServer

@attribute [Authorize]

<h3>@(IsEditMode ? "Редактировать сотрудника" : "Добавить сотрудника")</h3>

<**EditForm** **Model**="@employee" **OnValidSubmit**="Save">

<**DataAnnotationsValidator** />

<**ValidationSummary** />

<div class="mb-3">

<label>ФИО:</label>

<**InputText** class="form-control" @bind-Value="employee.FullName" />

<**ValidationMessage** **For**="() => employee.FullName" />

</div>

<div class="mb-3">

<label>Должность:</label>

<**InputText** class="form-control" @bind-Value="employee.Position" />

<**ValidationMessage** **For**="() => employee.Position" />

</div>

<button type="submit" class="btn btn-primary">Сохранить</button>

<button type="button" class="btn btn-secondary" @onclick='() => NavManager.NavigateTo("/employees")'>Отмена</button>

</**EditForm**>

@code {

[Parameter] public int? EmployeeId { get; set; }

private Employee employee = new();

private bool IsEditMode => EmployeeId.HasValue;

protected override async Task OnInitializedAsync()

{

if (IsEditMode)

{

employee = await DataService.GetEmployeeByIdAsync(EmployeeId.Value) ?? new();

}

}

*Продолжение приложения*

private async Task Save()

{

await DataService.AddOrUpdateEmployeeAsync(employee);

NavManager.NavigateTo("/employees");

}

}

Файл InventoryInStock.razor:

@page "/inventory/stock"

@inject IDataService DataService

@attribute [Authorize]

<h3>Остатки товаров на складе</h3>

@if (stock == null)

{

<p>Загрузка...</p>

}

else

{

<table class="table table-striped">

<thead>

<tr>

<th>Товар</th>

<th>Остаток</th>

</tr>

</thead>

<tbody>

@foreach (var row in stock)

{

<tr>

<td>@row.ProductName</td>

<td>@row.Quantity</td>

</tr>

}

</tbody>

</table>

}

@code {

private List<StockRow>? stock;

protected override async Task OnInitializedAsync()

{

var operations = await DataService.GetAllInventoryOperationsAsync();

stock = operations

.GroupBy(o => o.Product)

.Select(g => new StockRow

{

ProductName = g.Key.Name,

Quantity = g.Sum(x => x.OperationType switch

{

OperationType.Purchase => x.Quantity,

OperationType.Return => x.Quantity,

OperationType.Sale => -x.Quantity,

OperationType.WriteOff => -x.Quantity,

\_ => 0

})

})

.ToList();

}

public class StockRow

*Продолжение приложения*

{

public string ProductName { get; set; } = "";

public int Quantity { get; set; }

}

}

Файл NavMenu.razor:

@implements IDisposable

@inject NavigationManager NavigationManager

<div class="top-row ps-3 navbar navbar-dark">

<div class="container-fluid">

<a class="navbar-brand sidebar-title" href="">Товарный учёт</a>

</div>

</div>

<div class="nav-scrollable" onclick="document.querySelector('.navbar-toggler').click()">

<nav class="flex-column">

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="inventory">

<span class="bi bi-box-seam me-2" aria-hidden="true"></span> Склад

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="inventoryoperation">

<span class="bi bi-arrow-left-right me-2" aria-hidden="true"></span> Операции

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="employees">

<span class="bi bi-people me-2" aria-hidden="true"></span> Сотрудники

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="sales">

<span class="bi bi-bar-chart-line me-2" aria-hidden="true"></span> Отчёты

</**NavLink**>

</div>

<**AuthorizeView**>

<**Authorized**>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="Account/Manage">

<span class="bi bi-person-fill-nav-menu" aria-hidden="true"></span> @context.User.Identity?.Name

</**NavLink**>

</div>

<div class="nav-item px-3">

<form action="Account/Logout" method="post">

<**AntiforgeryToken** />

<input type="hidden" name="ReturnUrl" value="@currentUrl" />

*Продолжение приложения*

<button type="submit" class="nav-link">

<span class="bi bi-arrow-bar-left-nav-menu" aria-hidden="true"></span> Logout

</button>

</form>

</div>

</**Authorized**>

<**NotAuthorized**>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="Account/Register">

<span class="bi bi-person-nav-menu" aria-hidden="true"></span> Register

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="Account/Login">

<span class="bi bi-person-badge-nav-menu" aria-hidden="true"></span> Login

</**NavLink**>

</div>

</**NotAuthorized**>

</**AuthorizeView**>

</nav>

</div>

@code {

private string? currentUrl;

protected override void OnInitialized()

{

currentUrl = NavigationManager.ToBaseRelativePath(NavigationManager.Uri);

NavigationManager.LocationChanged += OnLocationChanged;

}

private void OnLocationChanged(object? sender, LocationChangedEventArgs e)

{

currentUrl = NavigationManager.ToBaseRelativePath(e.Location);

StateHasChanged();

}

public void Dispose()

{

NavigationManager.LocationChanged -= OnLocationChanged;

}

}