МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Вятский государственный университет»**

**Колледж ВятГУ**

**КУРСОВОЙ ПРОЕКТ**

| По междисциплинарному курсу | | | | | МДК 05.01 Проектирование и дизайн информационных | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| систем | | | | |
| Тема | Проектирование информационной системы для магазина программного обеспечения | | | | | | | | | |
|  | | | | | | | | | |
| Студента (ки) | | | | Корзунина Евгения Сергеевича | | | | | | |
|  | | | *ФИО (полностью в родительном падеже)* | | | | | | | |
| Курс | | 3 | | | | Форма обучения | очная | |
|  | | *(арабской цифрой)* | | | |  | | *(очная, заочная)* | | |
| Основная профессиональная образовательная программа по специальности | | | | | | | | | | |
| 09.02.07 Информационные системы и программирование | | | | | | | | | | |
| (код и наименование специальности без кавычек) | | | | | | | | | | |

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия, инициалы)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(прописью, без сокращений)*

Киров, 2025

**РЕФЕРАТ**

Пояснительная записка к курсовому проекту содержит: 52 страницы, 21 рисунок, 10 использованных источников.

Объектом и предметом исследования является информационная система магазина программного обеспечения.

Цель работы – ознакомление с процессом разработки информационной системы для магазина программного обеспечения

Поставлена задача разработать техническое задание для информационной системы выбранного объекта исследования.

В процессе работы были проведены следующие исследования: 1) обзор предметной области, обзор аналогов и сравнительный анализ; 2) написание самого технического задания, состоящего из: назначения разработки, функциональных характеристик, условий эксплуатации и требований к составу и параметрам технических средств; 3) описание решения и концепции; 4) архитектура решения; 5) разработка схем бизнес-процессов с их описанием; 6) разработка схем алгоритмов и кода на естественном языке; 7) проектирование прототипа пользовательского интерфейса с описанием.

Элементами научного новшества полученных результатов является автоматизация процессов управления каталогом товаров, формирования корзины, расчета стоимости заказа и хранения данных, обеспечивая базовый функционал для магазина ПО.

Областью возможного практического применения являются: различный бизнес, например магазины или отделы, продающие программное обеспечение.

Технико-экономическая и социальная значимость: внедрение систем увеличения эффективности продаж программного обеспечения.



**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ 4**](#_gjdgxs)

[**1. Возможности использования ЯВУ 5**](#_30j0zll)

[**1.1. Среда разработки Python 5**](#_1fob9te)

[**1.2. Обзор аналогов**](#_2et92p0) **7**

[**1.3. Описание предметной области 9**](#_4d34og8)

[**2. СХЕМЫ БИЗНЕС-ПРОЦЕССОВ И ИХ ОПИСАНИЕ 11**](#_17dp8vu)

[**2.1. Функциональный анализ ПО 11**](#_3rdcrjn)

[**2.2 Логическая модель ПО 15**](#_tyjcwt)

[**3. Разработка приложения 16**](#_3dy6vkm)

[**3.1. Выбор СУБД 16**](#_1t3h5sf)

[**3.2. Физическая модель БД 17**](#_26in1rg)

[**3.3. Разработка экранных форм 21**](#_lnxbz9)

[**ЗАКЛЮЧЕНИЕ 29**](#_35nkun2)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30**](#_1ksv4uv)

# **ВВЕДЕНИЕ**

Сегодня программное обеспечение стало неотъемлемой частью нашей жизни, и управление его продажами требует современных подходов. Если ранее продажа ПО могла напоминать традиционный процесс с бумажной документацией и ручным учётом, то теперь этот подход теряет свою актуальность. В отличие от «классического» подхода, наша информационная система (ИС) стремится предоставить более гибкий, автоматизированный и удобный способ управления продажами программного обеспечения для небольших магазинов. Здесь нет места ручным записям и сложным процессам - всё направлено на оптимизацию работы.

Что же представляет собой информационная система для магазина ПО в современном понимании? В наши дни она больше напоминает платформу для эффективного управления каталогом товаров и заказами. Это не просто хранилище данных о программных продуктах, но динамичная система, обеспечивающая автоматизацию процессов добавления, редактирования, и учета товаров, а также управления корзиной покупателя. В этой среде информация о продуктах доступна в удобном формате, а процессы заказа автоматизированы, что позволяет ускорить и упростить процесс взаимодействия с покупателем.

Данная система включает в себя роли для сотрудников магазина, позволяя им управлять каталогом товаров, обрабатывать заказы и обеспечивать наличие актуальной информации о программных продуктах. Пользователю предоставляется удобный интерфейс также создание соответствующей базы данных, что позволит улучшить эффективность управления, снизить трудозатраты и повысить удобство работы для сотрудников и покупателей.

Целью выполнения данной курсовой работы является разработка системы «магазина программного обеспечения» и соответствующей ей базы данных.

Для достижения поставленной цели в курсовой работе предусмотрено решение следующих задач:

* Изучение теоретических аспектов работы приложений для магазина программного обеспечения;
* Разработка базы данных;
* Разработка приложения;
* Проведение тестирования системы;
* Разработка документации;

# **Возможности использования ЯВУ**

## **Среда разработки Python**

Python — это объектно-ориентированный язык программирования, разработанный на языке C. По своему характеру этот язык программирования является языком высокого уровня, поэтому позволяет выполнять как простые, так и сложные операции. Наряду с этим Python поставляется с широким набором дополнений, а также библиотек, что позволяет ему поддерживать множество различных языков программирования, таких как Java, C, C++, и формат JSON.

Visual Studio Code (VS Code) — это кроссплатформенная интегрированная среда разработки, разработанная компанией Microsoft. Она поддерживает множество языков программирования, включая Python, JavaScript, TypeScript и многие другие. VS Code предлагает пользователю широкий набор инструментов для редактирования кода, отладки и работы с версиями, а также интеграцию с системами контроля версий. Основное внимание уделяется расширяемости: пользователи могут установить различные расширения для настройки функциональности редактора в соответствии с своими потребностями и предпочтениями.

Синтаксис языка-программирования Python всегда выделял его на фоне других языков программирования. Он не страдает избыточностью, схожесть синтаксиса с обычным английским позволяет понять код даже обычному пользователю, кроме того, программист пишет меньше строк кода, потому что нет необходимости использовать символы: «;», «{», «}». Вложенность обозначается отступами, что повышает читаемость кода и приучает c самого начала его использования правильному оформлению.

## **Обзор аналогов**

1 «Allsoft»

Один из самых популярных сайтов по покупке программного обеспечения. Недостатками аналога являются отсутствие фильтров и сортировки: каталог программного обеспечения не обладает гибкими фильтрами и сортировкой, что усложняет выбор нужного продукта. Также недостатком можно считать неудобный поиск: поиск по сайту не интуитивен и не выдает точных результатов на данный момент.

Ссылка: <https://golnk.ru/0189M>

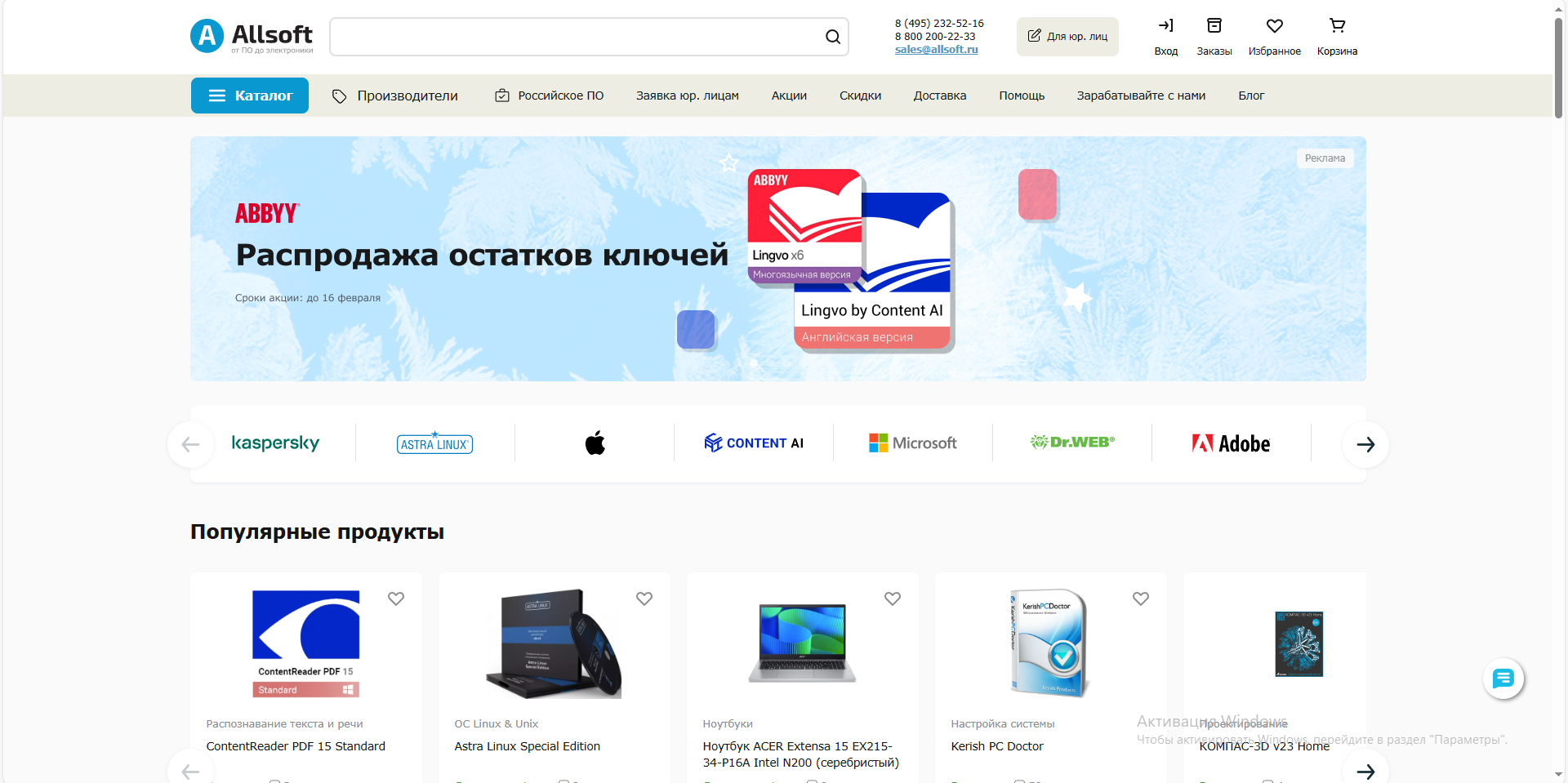


Рисунок 1 – главное окно магазина «Allsoft»

Плюсы:

* Возможность найти практически любое программное обеспечение.
* Наличие дополнительных услуг, таких как техподдержка и возможность получения акций и скидок.

Минусы:

* Каталог программного обеспечения не обладает гибкими фильтрами и сортировкой, что усложняет поиск и выбор необходимого продукта.
* Также недостатком можно считать неудобный поиск: поиск по сайту не интуитивен и не выдает точных результатов.

2 «Softline»

Это комплексная платформа для предоставления ИТ-решений и сервисов, ориентированная на корпоративных клиентов. Она выделяется широким ассортиментом, профессиональной экспертизой, предоставлением дополнительных услуг, а также надежностью и репутацией.

Ссылка: <https://golnk.ru/OWl5w>

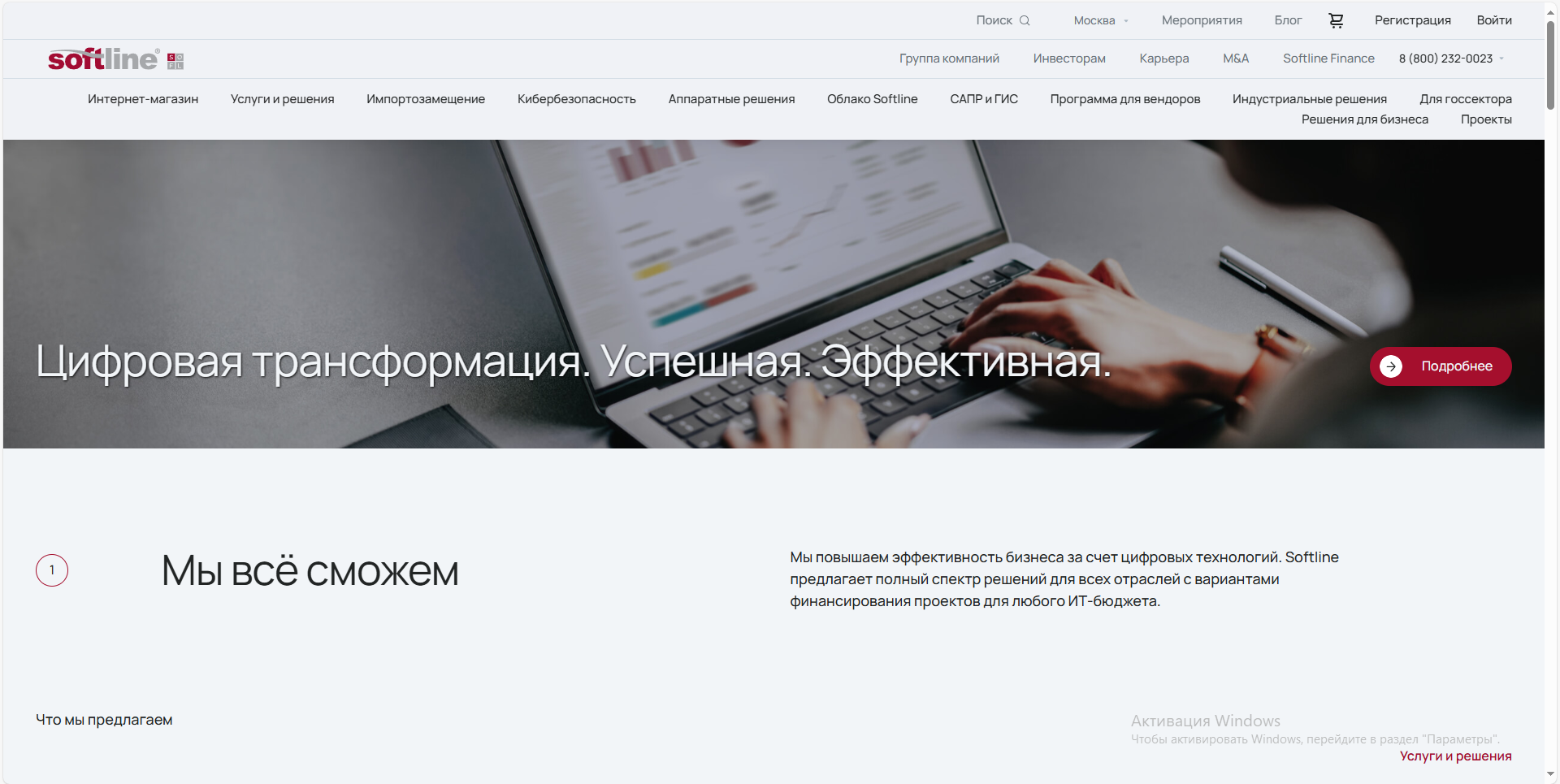


Рисунок 2 – главное окно магазина «Softline»

Плюсы:

* Ориентация на предоставление комплексных решений для бизнеса, включая интеграцию, внедрение и поддержку.
* Предоставляет специализированное ПО для различных отраслей (производство, финансы, образование и т. д.)

Минусы:

* неинтуитивная навигация: сложно найти нужный продукт, особенно если вы ищете что-то узкоспециализированное. Категории раздела «Офисные программы» не всегда логичны, и не всегда понятно, где искать нужную программу.
* Еще одним недостатком является перегруженность информацией на странице: страницы с описанием программ сильно перегружены информацией, что затрудняет восприятие. Отсутствие выделения ключевых моментов и удобного сравнения разных версий программы создает ощущение хаоса.

## **Описание предметной области**

Информационная система для магазина программного обеспечения (ПО) – это специализированная система, предназначенная для автоматизации процессов управления продажами и учета программных продуктов. Она позволяет магазинам эффективно управлять своим каталогом ПО, упрощать взаимодействие с покупателями и контролировать процесс продаж.

Программный продукт – это цифровой продукт, предоставляемый в виде программного обеспечения, лицензии на использование, или цифрового ключа доступа.

Магазин программного обеспечения – это организация, занимающаяся продажей различных видов программных продуктов.

Покупатель — это физическое или юридическое лицо, приобретающее программные продукты в магазине.

Корзина покупок – это временное хранилище выбранных покупателем программных продуктов.

Процесс покупки программного обеспечения включает в себя несколько пунктов:

* Просмотр каталога: Покупатель просматривает каталог доступных программных продуктов.
* Выбор товара: Покупатель выбирает необходимые программные продукты.
* Добавление в корзину: Выбранные товары добавляются в корзину.
* Редактирование корзины: Покупатель может удалить ненужные товары из корзины.
* Завершение заказа: Покупатель оформляет заказ, после чего происходит сохранение данных заказанного товара и отображение информации в соответствующей экранной форме.

# **СХЕМЫ БИЗНЕС-ПРОЦЕССОВ И ИХ ОПИСАНИЕ**

## **Функциональный анализ ПО**

На рисунке 1 изображена контекстная модель



Рисунок 1 – контекстная диаграмма сопоставление кандидатов и вакансий

Внешней входящей информацией являются:

* Данные о товарах;
* Данные о персонале;
* Данные о продажах.

Управляющим воздействием будет являться:

* ГОСТ

Механизмом управления будут являться:

* Персонал;

Внешней исходящей информацией будет являться:

* Данные о добавленном товаре;
* Данные о добавленном персонале;
* Продажи.

Для разбиения сложного процесса на составляющие подфункции применяется декомпозиция.

На рисунке 2 показан результат декомпозиции процесса

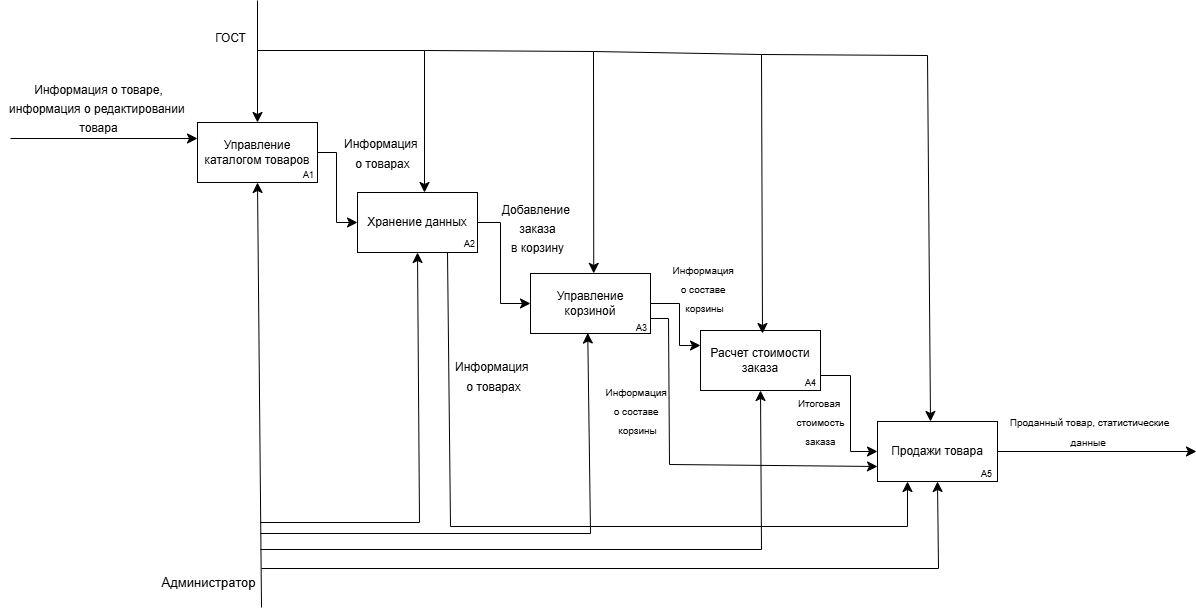


Рисунок 2 – результат процесса декомпозиции процесса

Диаграмма потоков данных показана на рисунке 3.

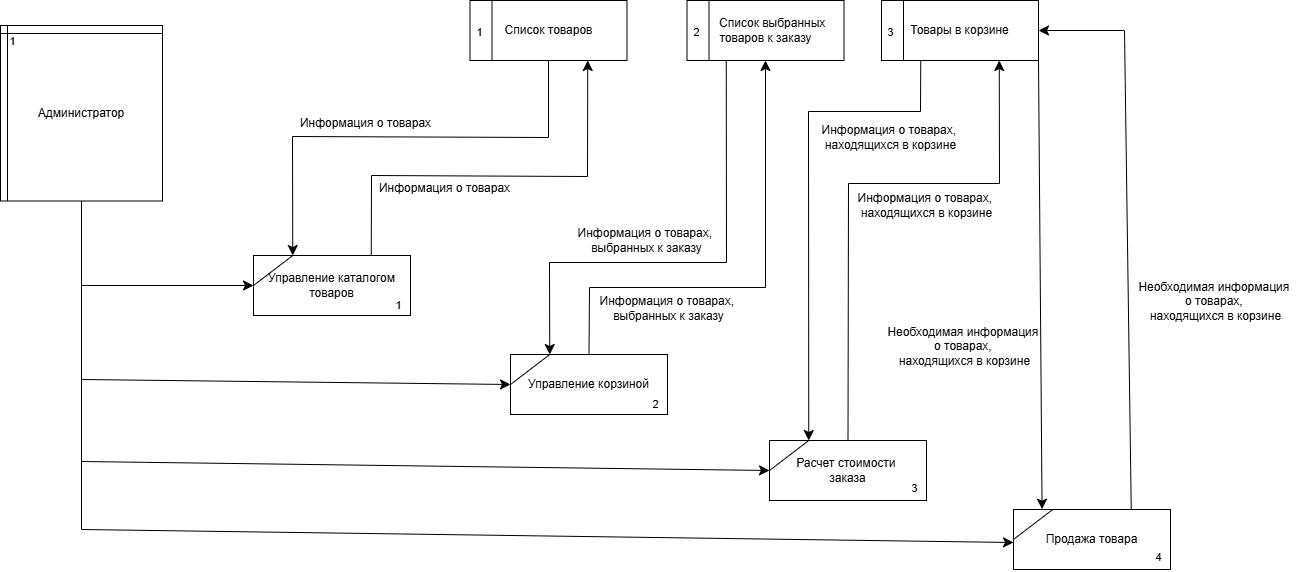


Рисунок 3 – диаграмма потоков данных

Диаграмма «Use Case» – диаграмма вариантов использования.

Она состоит из графической диаграммы, описывающей действующие лица, прецеденты, спецификации.

Диаграмма вариантов использования представлена на рисунке 4.

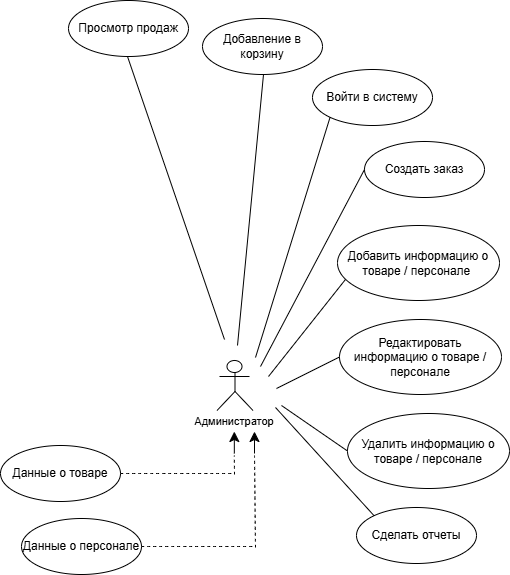


Рисунок 4 – диаграмма Use Case

## 

## **2.2 Логическая модель ПО**

Логическая модель данных является визуальным графическим представлением структур данных, их атрибутов и связей. Логическая модель представляет данные таким образом, чтобы они легко воспринимались.

Основными компонентами логической модели являются:

* сущности;
* атрибуты сущности;
* связи между сущностями.

Для описания мы используем нотацию IDEF1X, которая представляет структуру данных, описывающего предприятия. Методология IDEF1X – один из подходов к моделированию данных, основанный на концепции «сущность – связь».

Сущность в методологии IDEF1X является не зависимой, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями, а если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности, то сущность такая называется зависимой.

Логическая модель по нотации IDEF1X представлена на рисунке 4.

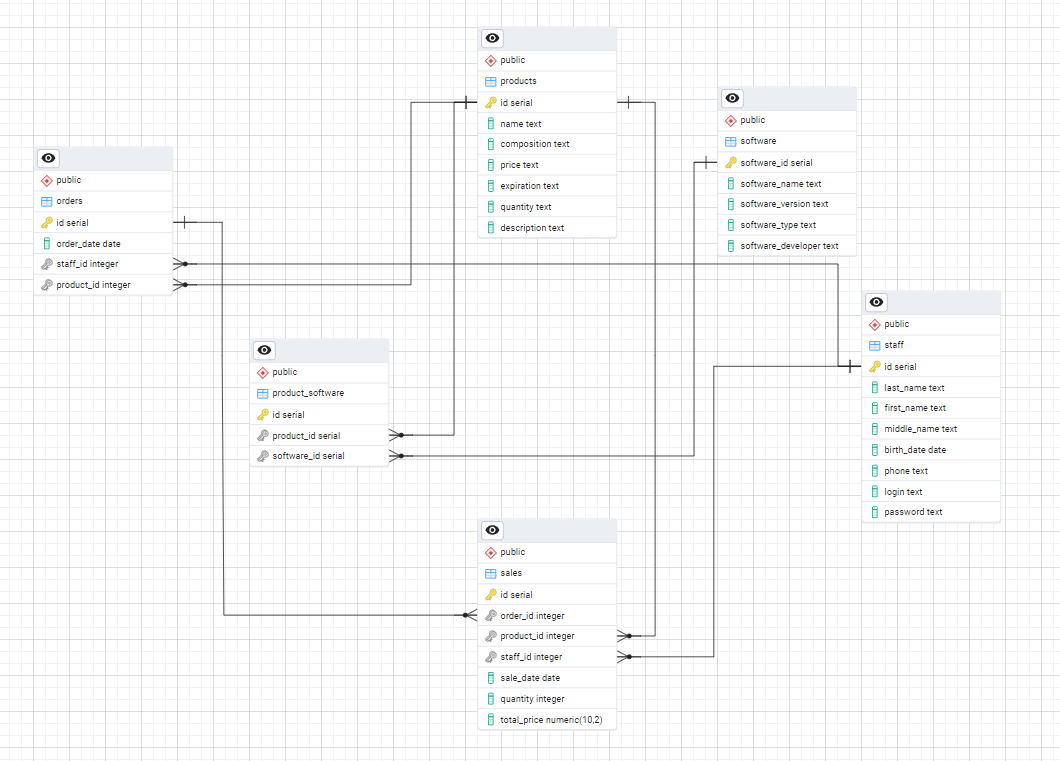


Рисунок 6 – логическая модель программного обеспечения

Логическая модель описывает, как связываются данные о работодателях, вакансиях, кандидатах и претендентах на вакансии. Каждая таблица содержит отдельную сущность с уникальными атрибутами, при этом используются внешние ключи для определения взаимосвязей между таблицами, что позволяет строить эффективную и структурированную базу данных для управления процессами подбора персонала.

# **Разработка приложения**

## **Выбор СУБД**

Для реализации хранения данных была выбрана PostgreSQL. Это решение обусловлено рядом факторов, которые делают данную систему управления базами данных (СУБД) подходящей для проекта по разработке информационной системы для кадрового агентства.

## **Физическая модель БД**

Физическая модель базы данных содержит все детали, необходимые конкретной СУБД для создания базы: наименования таблиц и столбцов, типы данных, определения первичных и внешних ключей.

Физическая модель строится на основе логической с учетом ограничений, накладываемых возможностями выбранной СУБД.

На рисунке 5 представлена физическая модель программы, которая описывает 4 таблиц, которые между собой имеют связи.

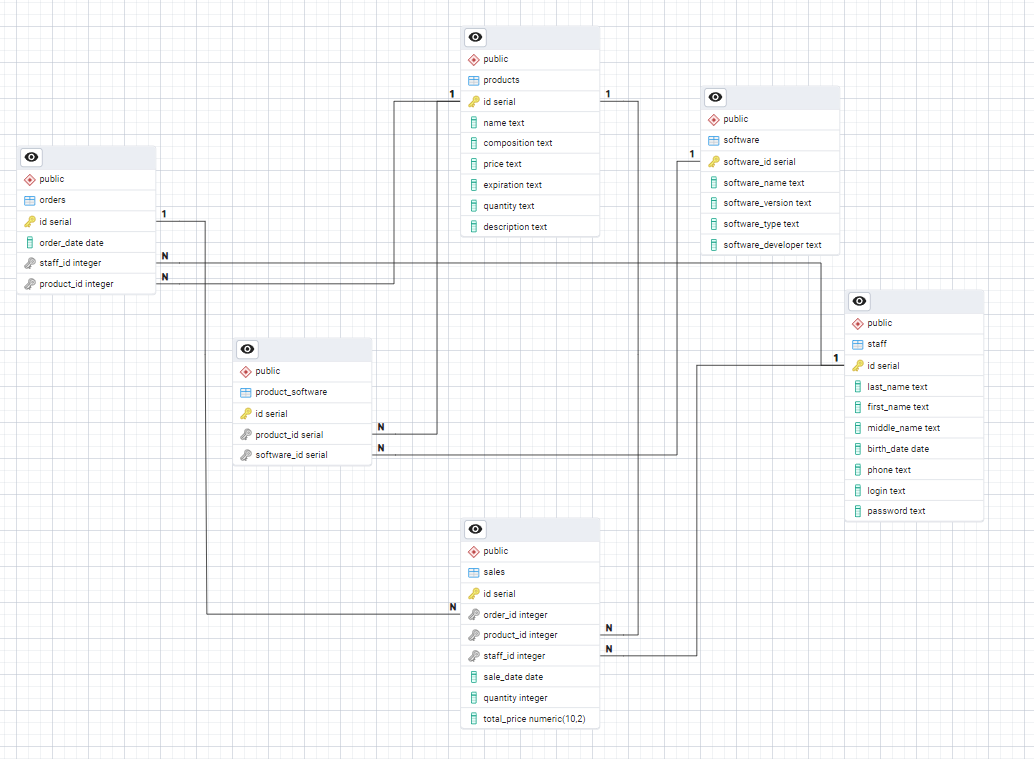


Рисунок 7 – физическая модель программного обеспечения

На рисунках ниже представлены физические модели программы, которая описывает 4 таблицы, которые между собой имеют связи.

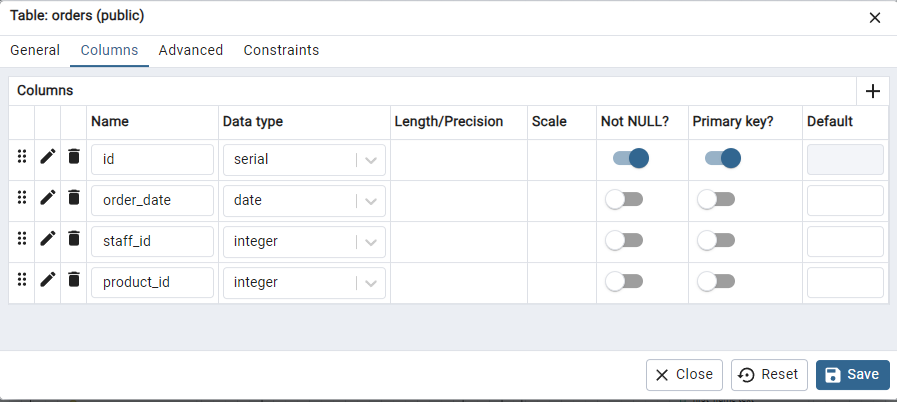


Рисунок 8 – Структура таблицы «Orders»

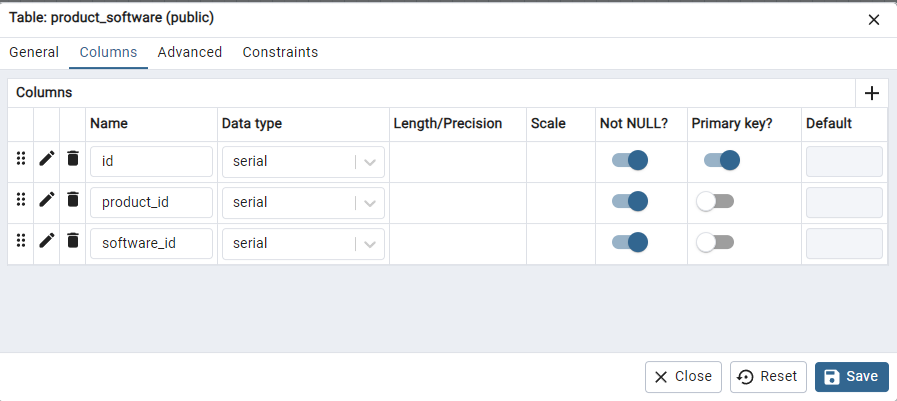


Рисунок 9 – Структура таблицы «product\_software»

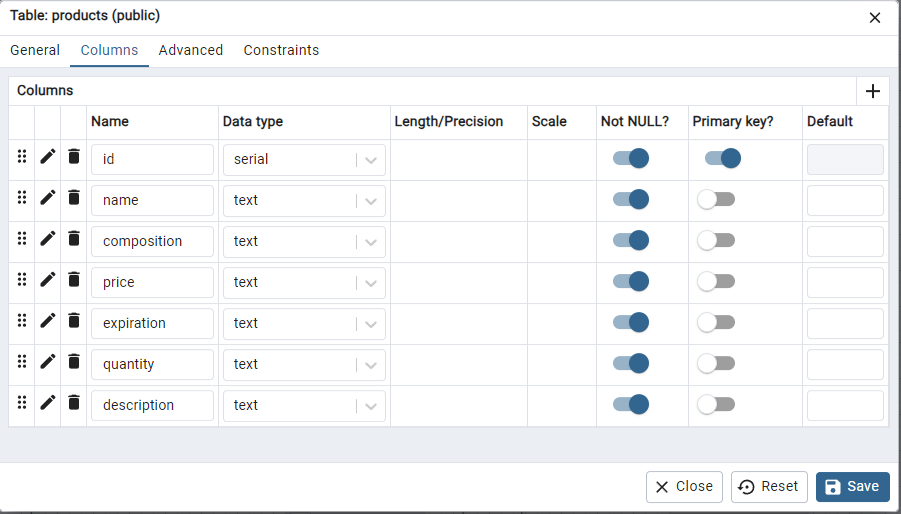


Рисунок 10 – Структура таблицы «products»

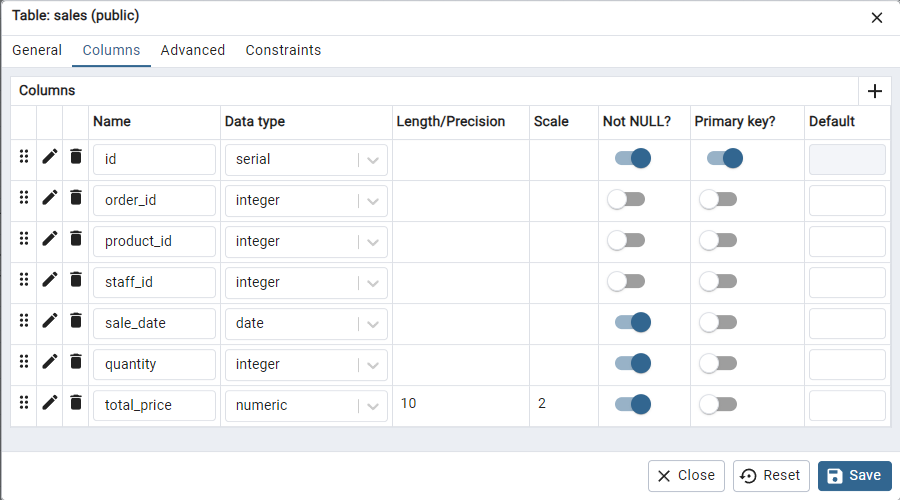


Рисунок 11 – Структура таблицы «sales»

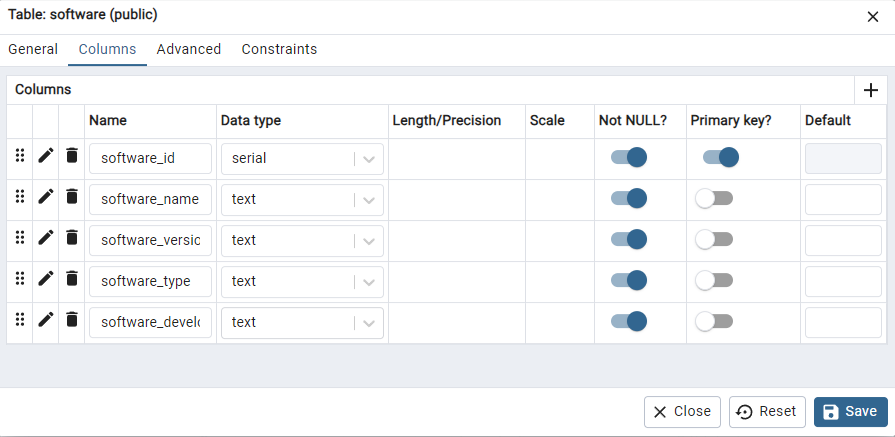


Рисунок 12 – Структура таблицы «software»

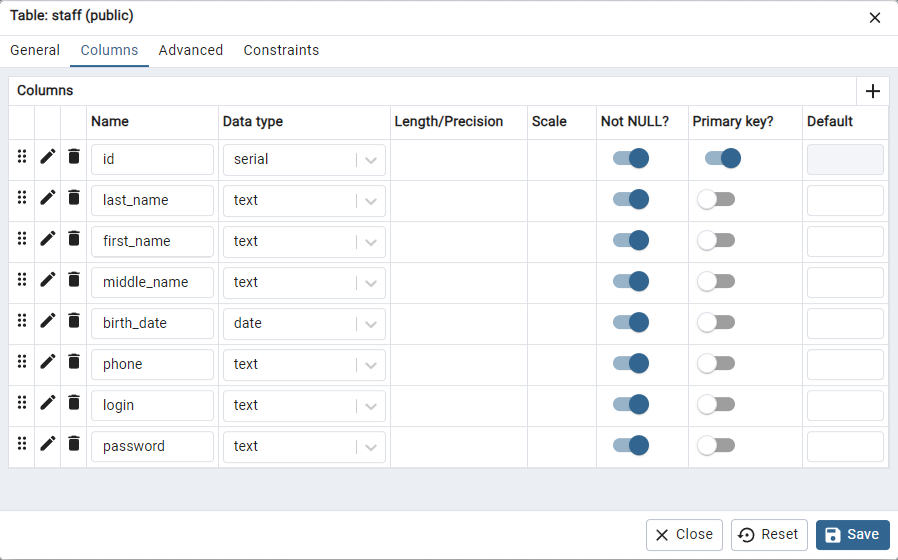


Рисунок 12 – Структура таблицы «staff»

## **Разработка экранных форм**

Для работы с программным продуктом используется простой, интуитивно понятный интерфейс. Последовательность работы с объектами формы определяется доступностью командных кнопок, целостность данных определяется набором используемых в программе проверок.

Переход от одного объекта формы к другому осуществляется при нажатии левой кнопки мыши по соответствующему объекту. При запуске системы открывается главная экранная форма приложения.

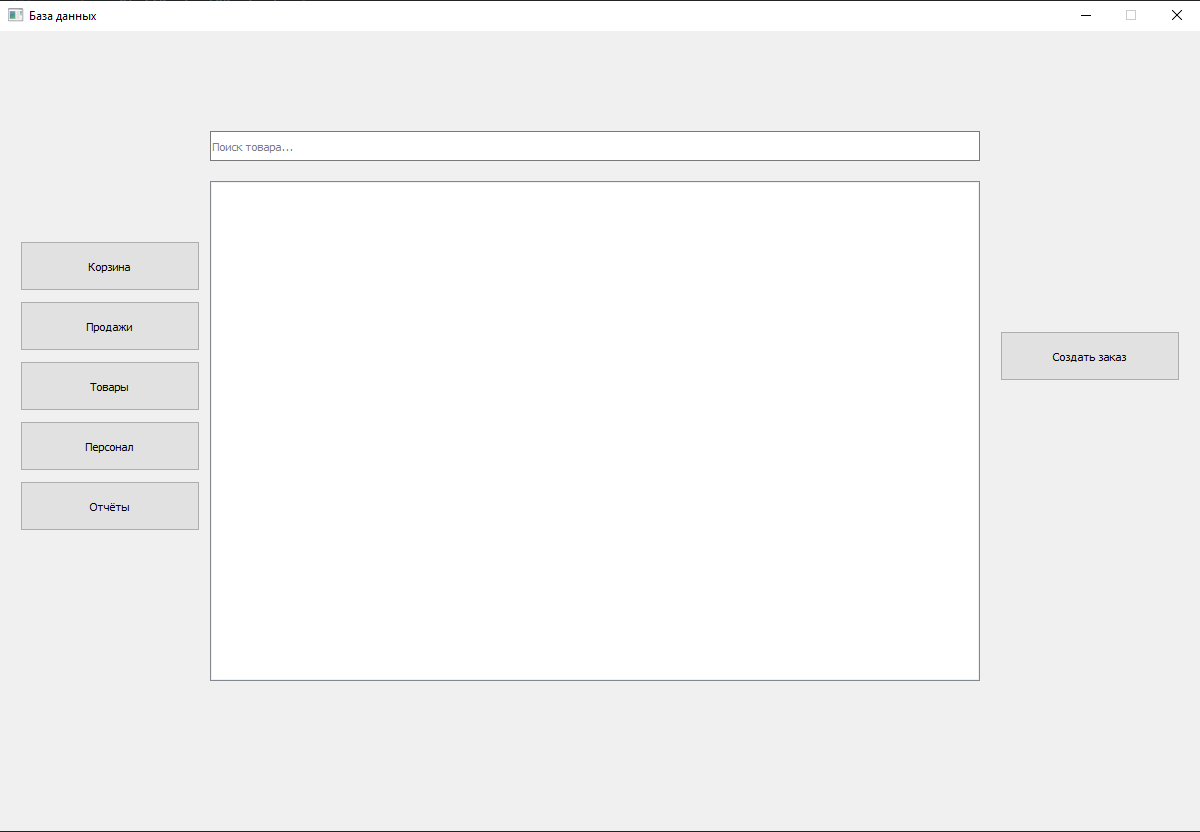


Рисунок 13 – Главная экранная форма приложения.

При запуске приложения появляется главное окно просмотра товаров. Администратор может выбрать несколько товаров и создать заказ, быстро выбрав товар, либо перейти на какие-либо другие экранные формы для выполнения соответствующих действий.

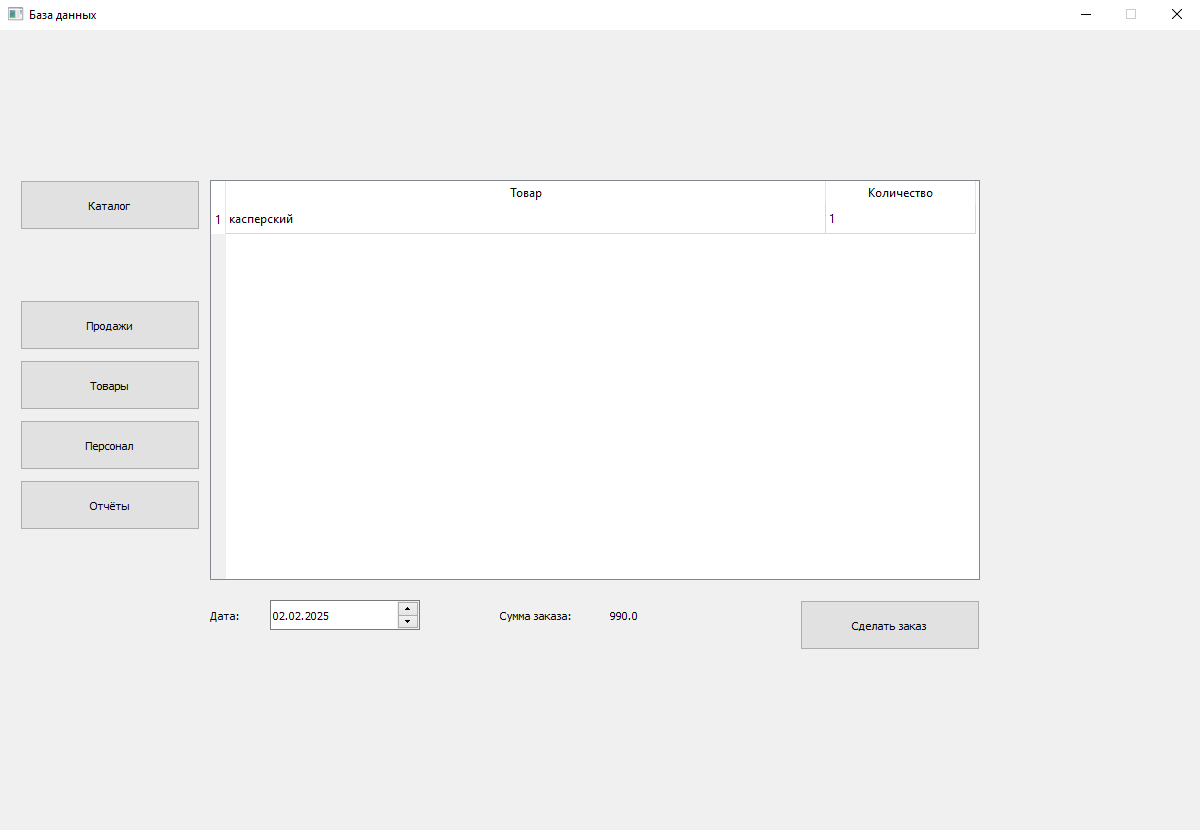


Рисунок 14 – экранная форма «Корзина».

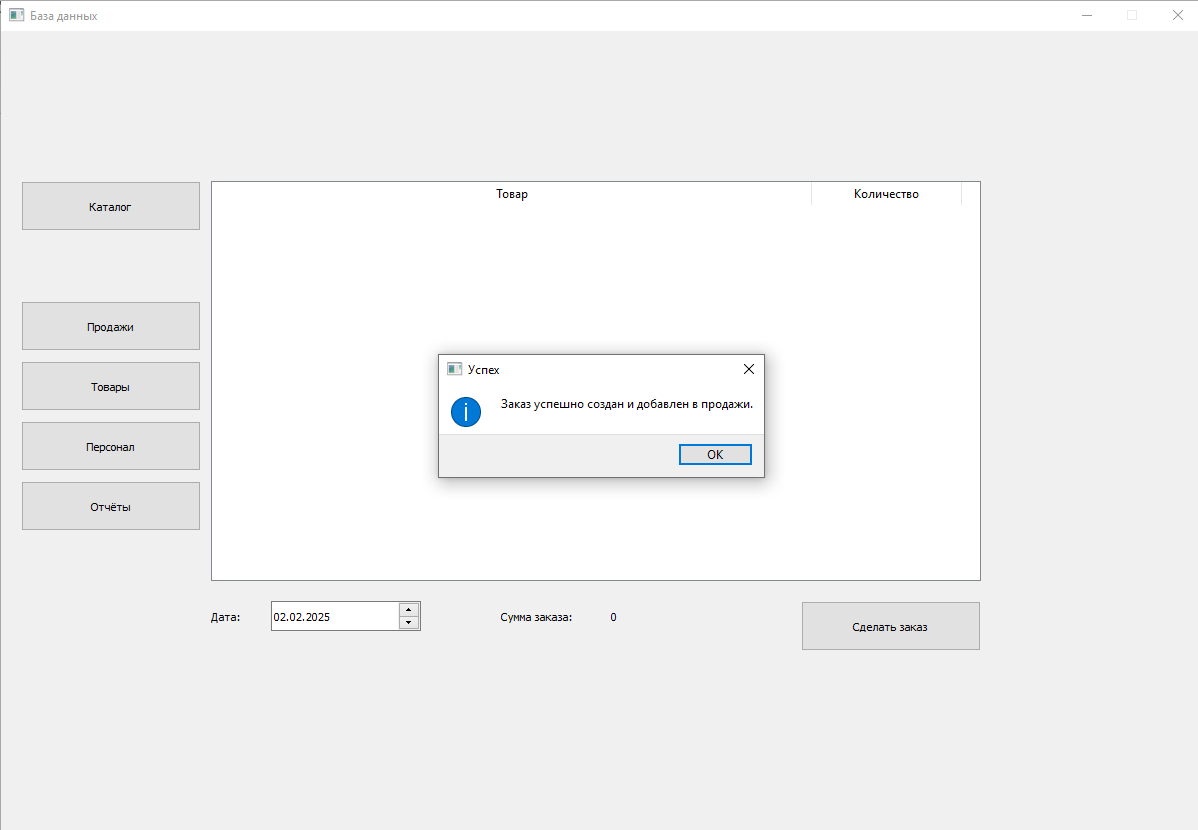


Рисунок 15 – создание заказа и перенос его в «Продажи».

При нажатии на кнопку «Корзина» (Рис. 14), открывается соответствующая экранная форма корзины. Чтобы заполнить корзину, нужно создать товар на экранной форме «Товары», после заполнения всей информации о товаре, можно будет выбрать опцию «Поместить в корзину». Корзина выводит данные о выбранном товаре на текущую дату, а также показывает количество выбранного товара и общую сумму заказа, отображенную чуть ниже. При нажатии на кнопку «Сделать заказ» создается новый заказ, при этом товар исчезает из корзины и появляется на экранной форме «Продажи» (Рис. 15).

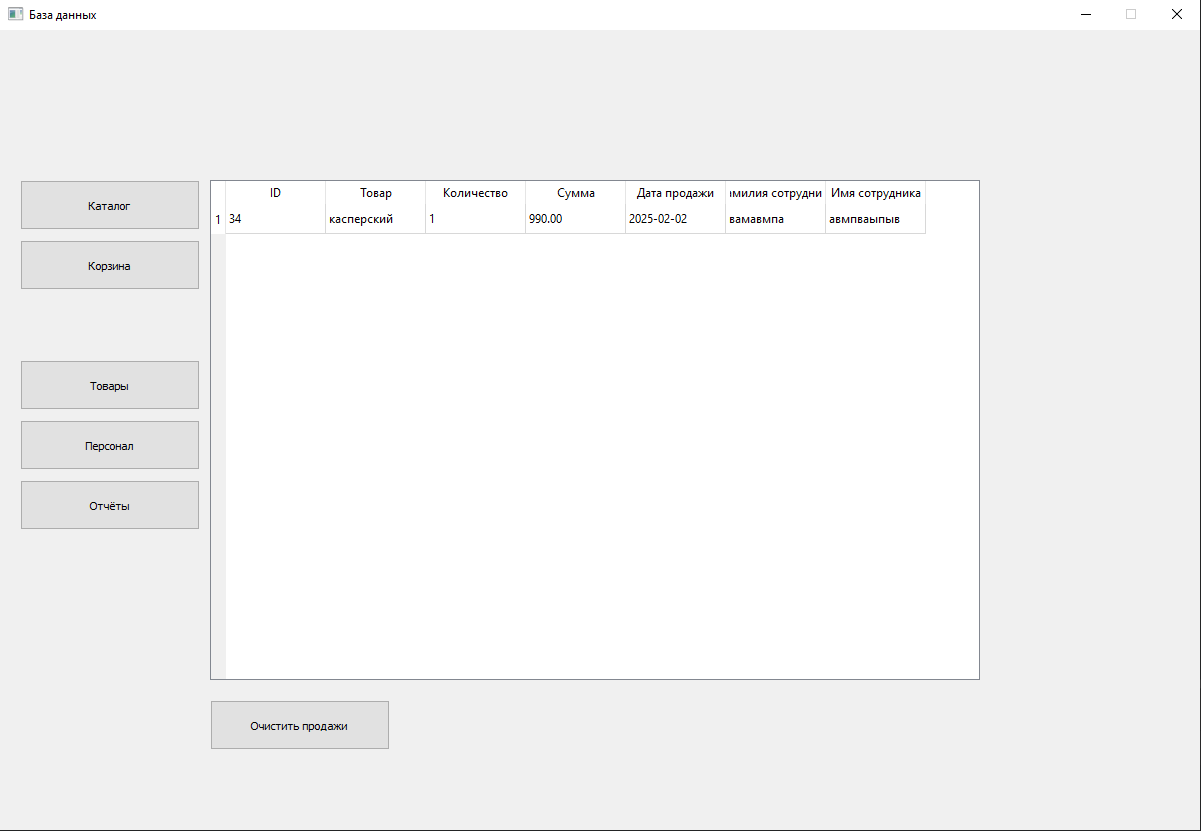


Рисунок 16 – форма «Продажи».

При нажатии на «Продажи», открывается соответствующая экранная форма. Здесь мы можем увидеть прошедшие продажи за все время, а также какой сотрудник ее произвел. При нажатии на кнопку «Очистить продажи» вся история продаж будет очищена и записи о продажах будут удалены.

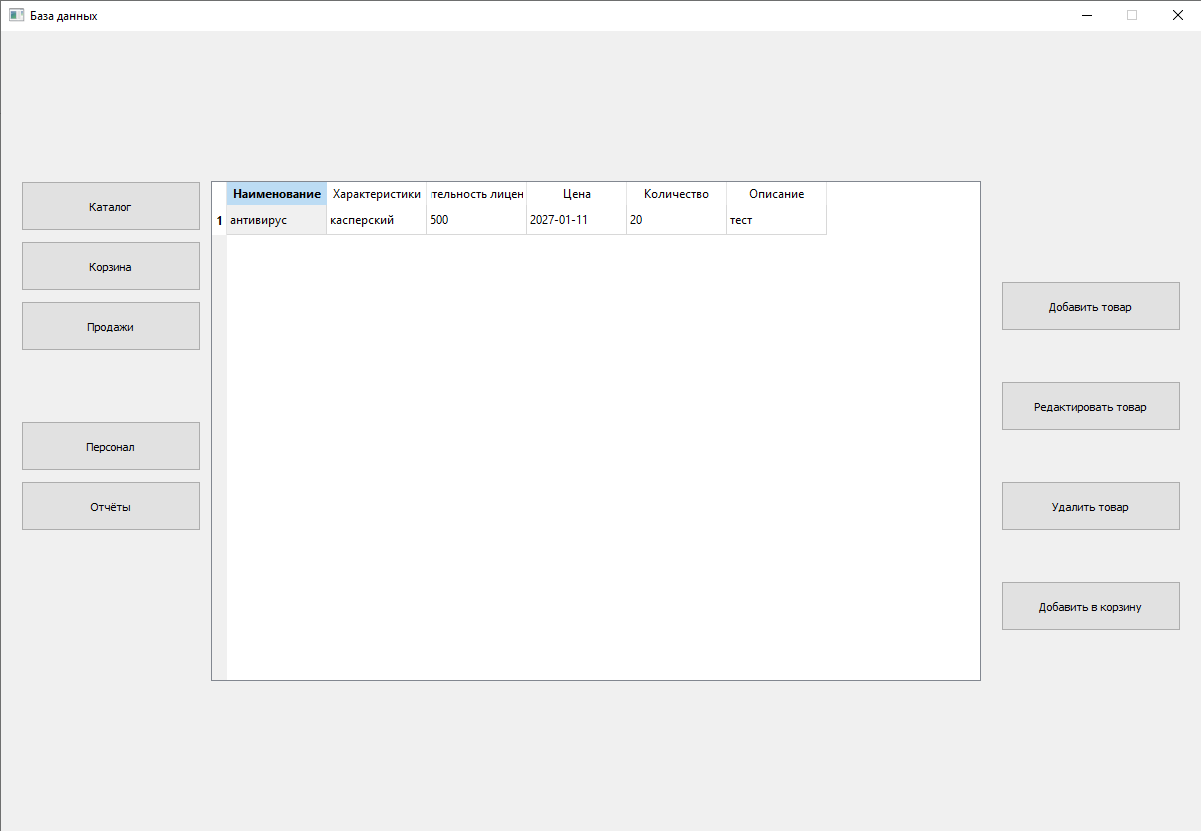


Рисунок 17 – экранная форма «Товары».

При нажатии на вкладку «Товары» открывается соответствующая экранная форма. На ней видно наименование, характеристики, длительность лицензии, цену, количество и описание товара. По бокам также присутствуют четыре кнопки, необходимые для корректного взаимодействия с товарами. Можно добавить новый товар, отредактировать существующий, удалить товар и добавить товар в корзину.

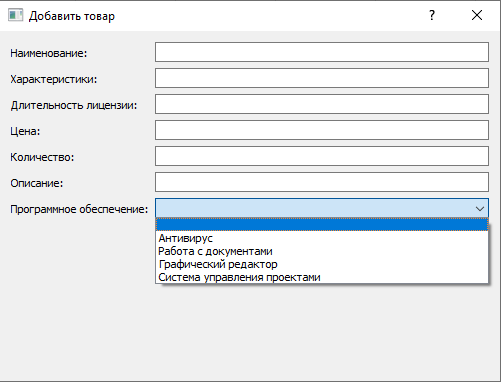


Рисунок 18 – экранная форма «Добавить / редактировать товар».

При нажатии на кнопки «Добавить товар» и «Редактировать товар» открываются соответствующие экранные формы, на которых можно изменить информацию о том или ином товаре. Экранная форма включает в себя изменения: наименования товара, характеристик товара, длительности лицензии товара, цену, количество, описание товара, а также вид программного обеспечения.

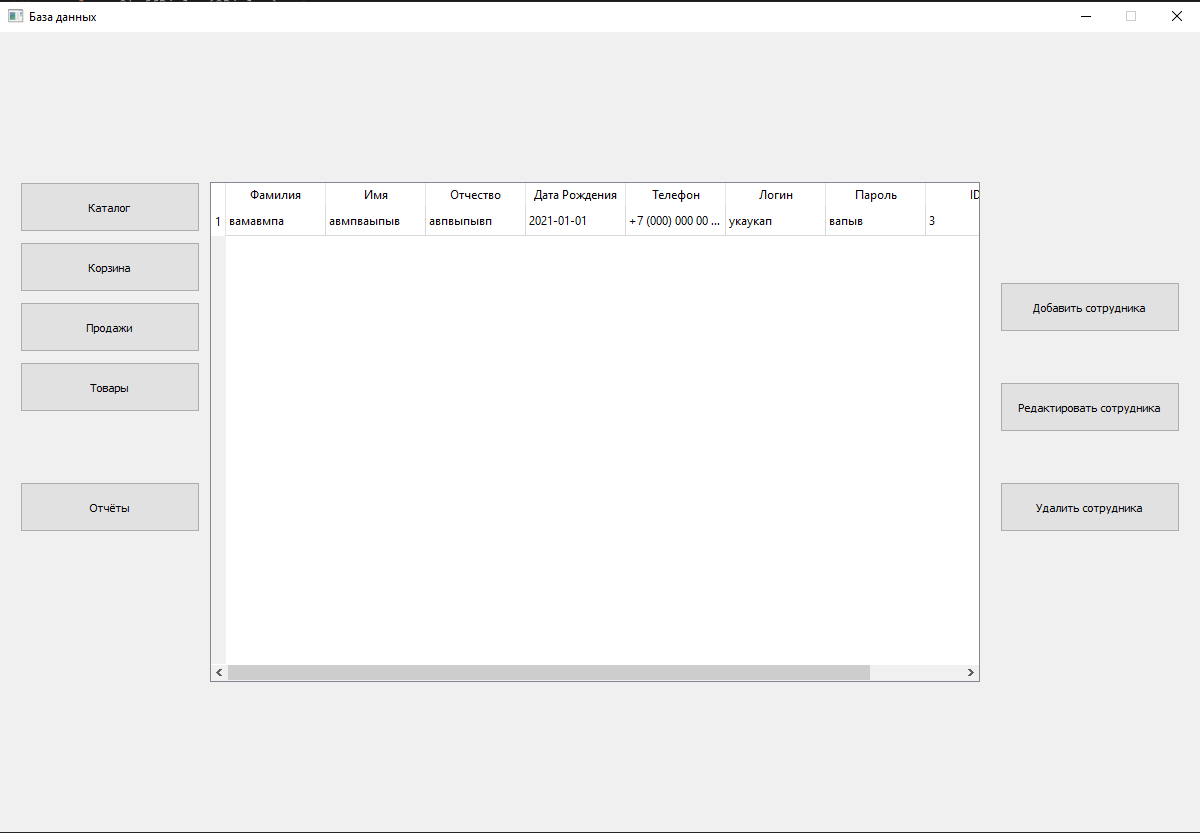


Рисунок 19 – экранная форма «Персонал».

При нажатии на вкладку «Персонал» открывается соответствующая экранная форма. На ней видно фамилию, имя, отчество, дату рождения, номер телефона, логин, пароль и ID выбранного сотрудника. По бокам также присутствуют три кнопки, необходимые для корректного взаимодействия с товарами. Можно добавить нового сотрудника, редактировать сотрудника или удалить сотрудника.

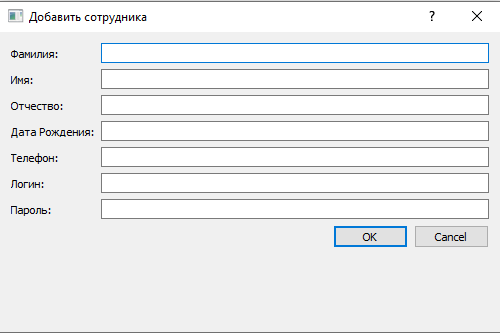


Рисунок 20 – экранная форма «Добавить / редактировать сотрудника».

При нажатии на кнопки «Добавить сотрудника» и «Редактировать сотрудника» открываются соответствующие экранные формы, на которых можно изменить информацию о том или ином сотруднике. Экранная форма включает в себя изменения: фамилии, имени, отчества, даты рождения, телефона, логина и пароля сотрудника.

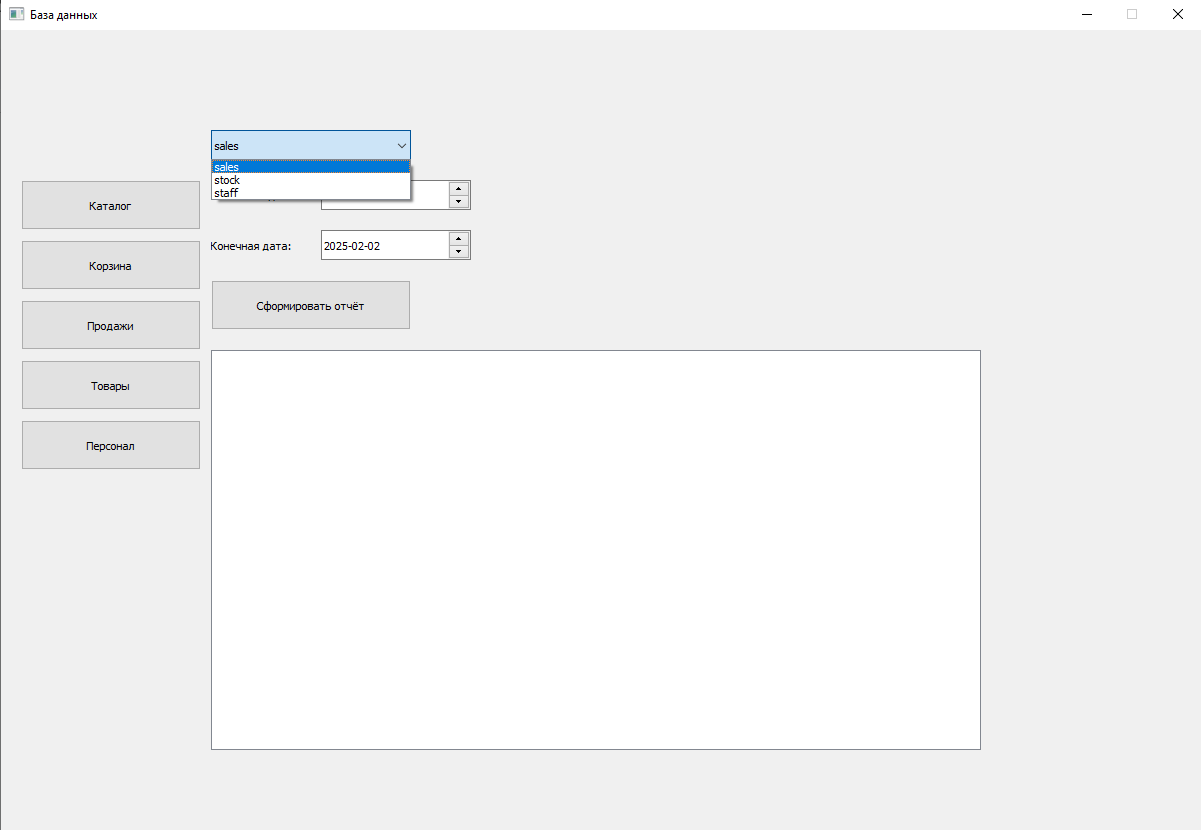


Рисунок 21 – экранная форма «Отчеты».

При нажатии на экранную форму «Отчеты» открывается соответствующая экранная форма. Здесь из выпадающего меню можно выбрать, как будет сформирован отчет: сколько сейчас имеется новых сотрудников, сколько было выполнено продаж и сколько сейчас осталось непроданных товаров за определенную дату. Для формирования нового отчета необходимо нажать на кнопку «Сформировать отчет», а также убедиться, что все остальные экранные формы были заполнены.

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения данной работы был проведен анализ задач, связанных с автоматизацией процессов магазина программного обеспечения, и предложены пути их решения. В ходе исследования были изучены принципы работы аналогичных систем, что позволило сформировать более полное представление о необходимых функциональных возможностях. Благодаря проделанной работе был достигнут ряд поставленных целей.

Целью данной работы было ознакомление с процессом проектирования и разработки информационной системы для магазина программного обеспечения, от анализа предметной области до создания модели данных и схем бизнес-процессов.

В процессе выполнения работы были проведены: обзор предметной области, разработка технического задания, описание концепции проекта, моделирование бизнес-процессов с их подробным описанием, а также создание схемы потоков данных. Особое внимание было уделено автоматизации основных процессов, таких как управление каталогом товаров, управление корзиной и формирование заказов.

В процессе создания информационной системы для магазина программного обеспечения были получены навыки, необходимые для обеспечения эффективной работы автоматизированного решения. Также были освоены навыки написания технического задания, учитывающего специфику предметной области и потребности пользователей. Были отработаны навыки создания схем потоков данных, которые наглядно представляют, как происходит обработка информации в системе.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы (пример) – Текст электронный // Корпоративные хранилища данных. Интеграция систем. Проектная документация: [сайт] – 2010. – URL: https://www.prj-exp.ru/patterns/pattern\_tech\_task.php#tech\_task2 (дата обращения: 09.11.2020).
2. Логическая модель ИС – Текст электронный // Студвуд: [сайт] – 2020. – URL: https://studwood.ru/1050371/informatika/logicheskaya\_model (дата обращения: 22.11.2020).
3. DFD – диаграммы потоков данных – Текст электронный // Инфопедия: [сайт] – 2016. – URL: https://infopedia.su/12x891b.html (дата обращения: 27.11.2020).
4. Проектирование USE CASE диаграммы – Текст электронный // Программные решения для бизнеса: [сайт]. – 2020. – URL: https://nationalteam.worldskills.ru/skills/proektirovanie-use-case-diagrammy-opredelenie-funktsionalnykh-vozmozhnostey-sistemy/ (дата обращения: 30.11.2020).
5. Основы UML. Диаграммы последовательностей – Текст электронный // Блог Программиста: [сайт]. – 2016. – URL: https://pro-prof.com/archives/2769 (дата обращения: 02.12.2020).
6. Целостность баз данных – Текст электронный // URL: http://www.ssofta.narod.ru/bd/13.htm (дата обращения: 02.12.2020).
7. ГОСТ 34. Разработка автоматизированной системы управления (АСУ). – Текст: электронный // Электронный фонд правовой и нормативно-технической документации: официальный сайт. - 2020. - URL: http://docs.cntd.ru/document/gost-34-602-89 (дата обращения: 12.11.2020).
8. Документация к PostgresSQL 17.2 // PostgresPro. URL: https://postgrespro.ru/docs/postgresql/17/index (дата обращения: 01.10.2024).
9. Rак работать с PostgreSQL в Python // Академия Selectel. URL: https://selectel.ru/blog/tutorials/postgresql-python (дата обращения: 11.10.2024).
10. Марк Лутц Изучаем Python. — 1 изд. — Диалектика, 2019. — 833 с.

# **ПРИЛОЖЕНИЕ А**

# 

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ КОД**

import sys

from datetime import datetime, date

import psycopg2

import re

from PyQt5 import QtWidgets, QtCore

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QListWidget, QListWidgetItem, QTableWidget, \

QTableWidgetItem, QDialog, QFormLayout, QMessageBox

from PyQt5.QtWidgets import QComboBox, QDateEdit, QLabel

from PyQt5.QtCore import QDate

from PyQt5 import QtWidgets, QtCore

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QListWidget, QListWidgetItem, QTableWidget, \

QTableWidgetItem, QDialog, QFormLayout, QMessageBox, QLabel, QTextEdit

import logging

logging.basicConfig(filename='app.log', level=logging.DEBUG)

def add\_product(self):

dialog = ProductDialog()

if dialog.exec\_() == QDialog.Accepted:

product\_data = dialog.get\_product\_data()

logging.debug(f"Data from ProductDialog: {product\_data}")

# Валидация данных

if not product\_data["name"] or not product\_data["composition"] or not product\_data["price"] or not product\_data[

"expiration"] or not product\_data["quantity"] or not product\_data["description"]:

QMessageBox.warning(self, "Ошибка", "Все поля должны быть заполнены.")

logging.warning("Not all fields are filled.")

return

try:

price = product\_data["price"]

quantity = int(product\_data["quantity"])

expiration\_date = datetime.strptime(product\_data["expiration"], "%Y-%m-%d").date()

except ValueError as e:

QMessageBox.warning(self, "Ошибка", f"Некорректные данные: {e}")

logging.warning(f"Invalid input {e}")

return

try:

row\_position = self.product\_table.rowCount()

self.product\_table.insertRow(row\_position)

self.product\_table.setItem(row\_position, 0, QTableWidgetItem(product\_data["name"]))

self.product\_table.setItem(row\_position, 1, QTableWidgetItem(product\_data["composition"]))

self.product\_table.setItem(row\_position, 2, QTableWidgetItem(str(price)))

self.product\_table.setItem(row\_position, 3, QTableWidgetItem(expiration\_date.strftime("%Y-%m-%d")))

self.product\_table.setItem(row\_position, 4, QTableWidgetItem(str(quantity)))

self.product\_table.setItem(row\_position, 5, QTableWidgetItem(product\_data["description"]))

query = """

INSERT INTO products (name, composition, price, expiration, quantity, description)

VALUES (%s, %s, %s, %s, %s, %s)

"""

logging.debug(f"add\_product query: {query}")

self.cursor.execute(query, (

product\_data["name"],

product\_data["composition"],

price,

expiration\_date,

quantity,

product\_data["description"]

))

self.connection.commit()

QMessageBox.information(self, "Успех", "Товар успешно добавлен в базу данных.")

logging.info("Product added successfully to DB.")

except Exception as e:

self.connection.rollback()

QMessageBox.warning(self, "Ошибка", f"Не удалось добавить товар в базу данных: {e}")

logging.error(f"Error adding product to database: {e}")

def is\_valid\_phone(phone):

pattern = r'^\+7 \(\d{3}\) \d{3} \d{2} \d{2}$'

return re.fullmatch(pattern, phone) is not None

class ProductDialog(QDialog):

def \_\_init\_\_(self, product\_data=None, connection=None):

super().\_\_init\_\_()

self.setWindowTitle("Добавить товар" if product\_data is None else "Редактировать товар")

self.setFixedSize(500, 350) # Увеличим высоту

self.connection = connection

self.initUI(product\_data)

self.load\_all\_software() # Загружаем список ПО

if product\_data:

self.load\_software\_for\_product(product\_data.get("id"))

def initUI(self, product\_data):

layout = QFormLayout()

self.name\_input = QtWidgets.QLineEdit(self)

self.composition\_input = QtWidgets.QLineEdit(self)

self.price\_input = QtWidgets.QLineEdit(self)

self.expiration\_input = QtWidgets.QLineEdit(self)

self.quantity\_input = QtWidgets.QLineEdit(self)

self.description\_input = QtWidgets.QLineEdit(self)

layout.addRow("Наименование:", self.name\_input)

layout.addRow("Характеристики:", self.composition\_input)

layout.addRow("Длительность лицензии:", self.expiration\_input)

layout.addRow("Цена:", self.price\_input)

layout.addRow("Количество:", self.quantity\_input)

layout.addRow("Описание:", self.description\_input)

# Добавляем ComboBox для выбора ПО

self.software\_combo = QComboBox(self)

layout.addRow("Программное обеспечение:", self.software\_combo)

# Использование QDialogButtonBox для правильной работы кнопок

self.button\_box = QtWidgets.QDialogButtonBox(QtWidgets.QDialogButtonBox.Ok | QtWidgets.QDialogButtonBox.Cancel, self)

self.button\_box.accepted.connect(self.accept)

self.button\_box.rejected.connect(self.reject)

layout.addWidget(self.button\_box)

self.setLayout(layout)

if product\_data:

self.name\_input.setText(product\_data.get("name", ""))

self.composition\_input.setText(product\_data.get("composition", ""))

self.price\_input.setText(str(product\_data.get("price", "")))

self.expiration\_input.setText(str(product\_data.get("expiration", "")))

self.quantity\_input.setText(str(product\_data.get("quantity", "")))

self.description\_input.setText(product\_data.get("description", ""))

def load\_all\_software(self):

try:

cursor = self.connection.cursor()

cursor.execute("SELECT software\_id, software\_name FROM software")

software = cursor.fetchall()

self.software\_combo.clear()

self.software\_combo.addItem("", None) # Пустой элемент

for software\_id, software\_name in software:

self.software\_combo.addItem(software\_name, software\_id)

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось загрузить список ПО: {e}")

def load\_software\_for\_product(self, product\_id):

try:

cursor = self.connection.cursor()

cursor.execute("""

SELECT s.software\_id, s.software\_name FROM software s

JOIN product\_software ps ON s.software\_id = ps.software\_id

WHERE ps.product\_id = %s

""", (product\_id,))

software\_data = cursor.fetchall()

self.software\_combo.clear()

self.software\_combo.addItem("", None) # Пустой элемент

for software\_id, software\_name in software\_data:

self.software\_combo.addItem(software\_name, software\_id)

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось загрузить список ПО для товара: {e}")

def get\_product\_data(self):

return {

"name": self.name\_input.text(),

"composition": self.composition\_input.text(),

"price": self.price\_input.text(),

"expiration": self.expiration\_input.text(),

"quantity": self.quantity\_input.text(),

"description": self.description\_input.text(),

"software\_id": self.software\_combo.currentData()

}

class StaffDialog(QDialog):

def \_\_init\_\_(self, staff\_data=None):

super().\_\_init\_\_()

self.setWindowTitle("Добавить сотрудника" if staff\_data is None else "Редактировать сотрудника")

self.setFixedSize(500, 300)

self.initUI()

if staff\_data:

# Заполнение полей для редактирования

self.last\_name\_input.setText(staff\_data["last\_name"])

self.first\_name\_input.setText(staff\_data["first\_name"])

self.middle\_name\_input.setText(staff\_data["middle\_name"])

self.birth\_date\_input.setText(staff\_data["birth\_date"])

self.phone\_input.setText(staff\_data["phone"])

self.login\_input.setText(staff\_data["login"])

self.password\_input.setText(staff\_data["password"])

def initUI(self):

layout = QFormLayout()

self.last\_name\_input = QtWidgets.QLineEdit(self)

self.first\_name\_input = QtWidgets.QLineEdit(self)

self.middle\_name\_input = QtWidgets.QLineEdit(self)

self.birth\_date\_input = QtWidgets.QLineEdit(self) # Можно использовать QDateEdit для даты

self.phone\_input = QtWidgets.QLineEdit(self)

self.login\_input = QtWidgets.QLineEdit(self)

self.password\_input = QtWidgets.QLineEdit(self)

layout.addRow("Фамилия:", self.last\_name\_input)

layout.addRow("Имя:", self.first\_name\_input)

layout.addRow("Отчество:", self.middle\_name\_input)

layout.addRow("Дата Рождения:", self.birth\_date\_input)

layout.addRow("Телефон:", self.phone\_input)

layout.addRow("Логин:", self.login\_input)

layout.addRow("Пароль:", self.password\_input)

# Использование QDialogButtonBox для правильной работы кнопок

self.button\_box = QtWidgets.QDialogButtonBox(QtWidgets.QDialogButtonBox.Ok | QtWidgets.QDialogButtonBox.Cancel, self)

self.button\_box.accepted.connect(self.accept)

self.button\_box.rejected.connect(self.reject)

layout.addWidget(self.button\_box)

self.setLayout(layout)

def get\_staff\_data(self):

return {

"last\_name": self.last\_name\_input.text(),

"first\_name": self.first\_name\_input.text(),

"middle\_name": self.middle\_name\_input.text(),

"birth\_date": self.birth\_date\_input.text(),

"phone": self.phone\_input.text(),

"login": self.login\_input.text(),

"password": self.password\_input.text()

}

class Window(QWidget):

def \_\_init\_\_(self, window\_id, total\_windows):

super().\_\_init\_\_()

self.window\_id = window\_id

self.total\_windows = total\_windows

self.setFixedSize(1200, 800) # Устанавливаем размер окна

self.setWindowTitle('База данных') # Заголовок окна

# Подключение к базе данных PostgreSQL

self.connection = psycopg2.connect(

dbname="my\_tab\_db",

user="postgres",

password="Ryzen52600!",

host="localhost",

port="5432"

)

self.cursor = self.connection.cursor()

# Инициализация таблицы продаж

self.sales\_table = None # Инициализируем как None

self.connection = self.connection

# Инициализация кнопок и других элементов

self.initUI()

def initUI(self):

# Уникальное содержимое для каждого окна

if self.window\_id == 0:

self.create\_catalog\_content()

elif self.window\_id == 1:

self.create\_cart\_content()

elif self.window\_id == 2:

self.create\_sales\_content() # Вызов метода для создания интерфейса продаж

elif self.window\_id == 3:

self.create\_product\_content()

elif self.window\_id == 4:

self.create\_staff\_content()

elif self.window\_id == 5:

self.create\_report\_content()

else:

self.create\_navigation\_buttons()

def create\_report\_content(self):

self.create\_navigation\_buttons()

# Комбобокс для выбора типа отчёта

self.report\_type\_combo = QComboBox(self)

self.report\_type\_combo.addItems(["sales", "stock", "staff"])

self.report\_type\_combo.setGeometry(210, 100, 200, 30)

# Поля для ввода даты (если нужно)

self.start\_date\_label = QLabel("Начальная дата:", self)

self.start\_date\_label.setGeometry(210, 150, 100, 30)

self.start\_date\_input = QDateEdit(self)

self.start\_date\_input.setDisplayFormat("yyyy-MM-dd")

self.start\_date\_input.setDate(QDate.currentDate().addDays(-7)) # Устанавливаем неделю назад

self.start\_date\_input.setGeometry(320, 150, 150, 30)

self.end\_date\_label = QLabel("Конечная дата:", self)

self.end\_date\_label.setGeometry(210, 200, 100, 30)

self.end\_date\_input = QDateEdit(self)

self.end\_date\_input.setDisplayFormat("yyyy-MM-dd")

self.end\_date\_input.setDate(QDate.currentDate()) # Устанавливаем текущую дату

self.end\_date\_input.setGeometry(320, 200, 150, 30)

# Кнопка для генерации отчёта

self.generate\_report\_button = QPushButton("Сформировать отчёт", self)

self.generate\_report\_button.setGeometry(210, 250, 200, 50)

self.generate\_report\_button.clicked.connect(self.generate\_report)

# Таблица для отображения отчёта

self.report\_table = QTableWidget(self)

self.report\_table.setGeometry(210, 320, 770, 400)

self.report\_table.setColumnCount(0)

self.report\_table.setRowCount(0)

def get\_report\_query(self, report\_type, start\_date=None, end\_date=None):

*"""Генерирует SQL-запрос на основе типа отчета"""*

if report\_type == "sales":

query = """

SELECT o.order\_date, p.name, p.price, s.last\_name, s.first\_name

FROM orders o

JOIN products p ON o.product\_id = p.id

JOIN staff s ON o.staff\_id = s.id

WHERE o.order\_date BETWEEN %s AND %s;

"""

return query, (start\_date, end\_date)

elif report\_type == "stock":

return "SELECT name, quantity FROM products", None

elif report\_type == "staff":

return "SELECT last\_name, first\_name, middle\_name FROM staff", None

elif report\_type == "low\_stock":

return "SELECT name, quantity FROM products WHERE quantity < 10", None # Пример: если товаров меньше 10

elif report\_type == "latest\_products":

return "SELECT name, price, expiration FROM products ORDER BY id DESC LIMIT 10", None # Пример: 10 последних добавленных

else:

return None, None

def fetch\_report\_data(self, query, params):

*"""Выполняет SQL-запрос и возвращает данные"""*

try:

self.cursor.execute(query, params) if params else self.cursor.execute(query)

return self.cursor.fetchall()

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось загрузить данные для отчета: {e}")

return None

def display\_report(self, report\_type, start\_date=None, end\_date=None):

*"""Выполняет запрос, обрабатывает данные и отображает отчет в таблице."""*

query, params = self.get\_report\_query(report\_type, start\_date, end\_date)

if not query:

QMessageBox.warning(self, "Ошибка", "Неизвестный тип отчета.")

return

data = self.fetch\_report\_data(query, params)

if data is None:

return

self.report\_table.clearContents()

self.report\_table.setRowCount(0) # Очищаем предыдущие данные

if not data:

QMessageBox.information(self, "Информация", "Нет данных для данного отчета")

return

# Настройка столбцов в зависимости от типа отчета

headers = []

if report\_type == "sales":

headers = ["Дата заказа", "Товар", "Цена", "Фамилия сотрудника", "Имя сотрудника"]

elif report\_type == "stock":

headers = ["Название", "Количество"]

elif report\_type == "staff":

headers = ["Фамилия", "Имя", "Отчество"]

elif report\_type == "low\_stock":

headers = ["Название", "Количество"]

elif report\_type == "latest\_products":

headers = ["Название", "Цена", "Срок годности"]

self.report\_table.setColumnCount(len(headers))

self.report\_table.setHorizontalHeaderLabels(headers)

for row\_num, row\_data in enumerate(data):

self.report\_table.insertRow(row\_num)

for col\_num, cell\_data in enumerate(row\_data):

if isinstance(cell\_data, date):

self.report\_table.setItem(row\_num, col\_num, QTableWidgetItem(cell\_data.strftime("%Y-%m-%d")))

else:

self.report\_table.setItem(row\_num, col\_num, QTableWidgetItem(str(cell\_data)))

def generate\_report(self):

*"""Обработчик нажатия на кнопку формирования отчета"""*

report\_type = self.report\_type\_combo.currentText()

start\_date = self.start\_date\_input.date().toString(

"yyyy-MM-dd") if self.start\_date\_input else None # Получение даты начала

end\_date = self.end\_date\_input.date().toString(

"yyyy-MM-dd") if self.end\_date\_input else None # Получение даты конца

self.display\_report(report\_type, start\_date, end\_date)

def create\_navigation\_buttons(self):

button\_labels = ["Каталог", "Корзина", "Продажи", "Товары", "Персонал", "Отчёты"]

for i, label in enumerate(button\_labels):

if i != self.window\_id: # Не создаем кнопку для самого окна

btn = QPushButton(label, self)

btn.setGeometry(20, 150 + i \* 60, 180, 50) # Задаем геометрию кнопок

btn.clicked.connect(lambda checked, idx=i: self.go\_to\_window(idx))

def create\_cart\_content(self):

self.create\_navigation\_buttons()

# Таблица для товаров

self.cart\_table = QTableWidget(self)

self.cart\_table.setGeometry(210, 150, 770, 400)

self.cart\_table.setColumnCount(2)

self.cart\_table.setHorizontalHeaderLabels(["Товар", "Количество"])

self.cart\_table.setColumnWidth(0, 600)

self.cart\_table.setColumnWidth(1, 150)

self.cart\_table.setRowCount(0)

# Поле для выбора даты

self.date\_label = QLabel("Дата:", self)

self.date\_label.setGeometry(210, 570, 50, 30)

self.date\_input = QDateEdit(self)

self.date\_input.setGeometry(270, 570, 150, 30)

self.date\_input.setDate(QDate.currentDate())

# Поле для суммы заказа

self.total\_label = QLabel("Сумма заказа:", self)

self.total\_label.setGeometry(500, 570, 100, 30)

self.total\_value\_label = QLabel("0", self)

self.total\_value\_label.setGeometry(610, 570, 100, 30)

# Кнопка "Сделать заказ"

self.create\_order\_button = QPushButton("Сделать заказ", self)

self.create\_order\_button.setGeometry(800, 570, 180, 50)

self.create\_order\_button.clicked.connect(self.create\_order) # Подключаем метод create\_order

def create\_order(self):

try:

# Получаем текущую дату

sale\_date = datetime.now().date()

# Получаем данные из корзины и добавляем их в продажи

for row in range(self.cart\_table.rowCount()):

product\_name = self.cart\_table.item(row, 0).text()

quantity = int(self.cart\_table.item(row, 1).text())

# Получаем ID товара и цену

self.cursor.execute("SELECT id, price FROM products WHERE name = %s", (product\_name,))

product\_data = self.cursor.fetchone()

product\_id, price = product\_data

# Получаем ID сотрудника (первый в списке)

self.cursor.execute("SELECT id FROM staff LIMIT 1")

staff\_id = self.cursor.fetchone()[0]

# Вычисляем общую стоимость

total\_price = float(price) \* quantity

# Вставляем запись в таблицу sales

self.cursor.execute("""

INSERT INTO sales (order\_id, product\_id, staff\_id, sale\_date, quantity, total\_price)

VALUES (%s, %s, %s, %s, %s, %s)

""", (None, product\_id, staff\_id, sale\_date, quantity, total\_price))

self.connection.commit()

# Очищаем корзину

self.cart\_table.setRowCount(0)

self.total\_value\_label.setText("0")

# Обновляем таблицу продаж

self.load\_sales()

QMessageBox.information(self, "Успех", "Заказ успешно создан и добавлен в продажи.")

logging.info("Заказ добавлен в продажи.")

except Exception as e:

self.connection.rollback()

QMessageBox.warning(self, "Ошибка", f"Не удалось создать заказ: {e}")

logging.error(f"Ошибка создания заказа: {e}")

def load\_sales(self):

try:

self.cursor.execute("""

SELECT s.id, p.name, s.quantity, s.total\_price, s.sale\_date, st.last\_name, st.first\_name

FROM sales s

JOIN products p ON s.product\_id = p.id

JOIN staff st ON s.staff\_id = st.id

""")

sales = self.cursor.fetchall()

self.sales\_table.setRowCount(0)

self.sales\_table.setColumnCount(7)

self.sales\_table.setHorizontalHeaderLabels(

["ID", "Товар", "Количество", "Сумма", "Дата продажи", "Фамилия сотрудника", "Имя сотрудника"])

for sale in sales:

row\_position = self.sales\_table.rowCount()

self.sales\_table.insertRow(row\_position)

self.sales\_table.setItem(row\_position, 0, QTableWidgetItem(str(sale[0])))

self.sales\_table.setItem(row\_position, 1, QTableWidgetItem(sale[1]))

self.sales\_table.setItem(row\_position, 2, QTableWidgetItem(str(sale[2])))

self.sales\_table.setItem(row\_position, 3, QTableWidgetItem(str(sale[3])))

self.sales\_table.setItem(row\_position, 4, QTableWidgetItem(sale[4].strftime("%Y-%m-%d")))

self.sales\_table.setItem(row\_position, 5, QTableWidgetItem(sale[5]))

self.sales\_table.setItem(row\_position, 6, QTableWidgetItem(sale[6]))

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось загрузить данные о продажах: {e}")

logging.error(f"Ошибка загрузки продаж: {e}")

return # Добавили return здесь!

def create\_catalog\_content(self):

# Добавляем кнопки в первом окне

self.create\_navigation\_buttons() # Добавляем навигационные кнопки

# Добавляем дополнительные кнопки

self.order\_button = QPushButton('Создать заказ', self)

self.order\_button.setGeometry(1000, 300, 180, 50) # Задаем позицию

self.order\_button.clicked.connect(self.on\_order\_button\_click)

self.search\_line = QtWidgets.QLineEdit(self)

self.search\_line.setGeometry(210, 100, 770, 30)

self.search\_line.setPlaceholderText("Поиск товара...")

self.product\_catalog = QListWidget(self)

self.product\_catalog.setGeometry(210, 150, 770, 500)

self.load\_products\_to\_catalog() # Загружаем товары

def load\_products\_to\_catalog(self):

try:

self.cursor.execute("SELECT name FROM products;") # Извлекаем имена продуктов

products = self.cursor.fetchall() # Получаем все строки

self.product\_catalog.clear()

for product in products:

item = QListWidgetItem(product[0]) # имя продукта

item.setFlags(item.flags() | QtCore.Qt.ItemIsUserCheckable)

item.setCheckState(QtCore.Qt.Unchecked)

self.product\_catalog.addItem(item)

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось загрузить товары: {e}")

def load\_cart\_data(self):

self.cart\_table.setRowCount(0)

try:

self.cursor.execute("""

SELECT p.name, o.quantity, p.price

FROM orders o

JOIN products p ON o.product\_id = p.id;

""")

orders = self.cursor.fetchall()

if not orders:

QMessageBox.information(self, "Информация", "Корзина пуста")

for order in orders:

row\_position = self.cart\_table.rowCount()

self.cart\_table.insertRow(row\_position)

self.cart\_table.setItem(row\_position, 0, QTableWidgetItem(order[0])) # Имя товара

self.cart\_table.setItem(row\_position, 1, QTableWidgetItem(str(order[1]))) # Количество

self.cart\_table.setItem(row\_position, 2, QTableWidgetItem(str(order[2]))) # Цена

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Ошибка при загрузке данных корзины: {e}")

logging.error(f"Error loading cart {e}")

def calculate\_total(self):

*"""Вычисляет и отображает общую сумму заказа"""*

total\_price = 0

for row in range(self.cart\_table.rowCount()):

item\_name = self.cart\_table.item(row, 0).text()

item\_qty = int(self.cart\_table.item(row, 1).text())

# Получение данных о цене из базы данных или тестовых данных

try:

self.cursor.execute("SELECT price FROM products WHERE name = %s", (item\_name,))

product\_data = self.cursor.fetchone()

if product\_data:

product\_price = float(product\_data[0])

total\_price += product\_price \* item\_qty

else:

QMessageBox.warning(self, "Ошибка", f"Не найдена цена товара: {item\_name}")

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось получить данные о цене товара: {e}")

self.total\_value\_label.setText(str(total\_price))

def load\_cart\_data\_from\_catalog(self, item\_name):

*"""Загружает данные о продукте в корзину из каталога"""*

row\_position = self.cart\_table.rowCount()

self.cart\_table.insertRow(row\_position) # Добавляем новую строку

item\_widget = QTableWidgetItem(item\_name)

item\_widget.setFlags(item\_widget.flags() & ~QtCore.Qt.ItemIsEditable)

self.cart\_table.setItem(row\_position, 0, item\_widget)

qty\_widget = QTableWidgetItem(str(1)) # 1 шт по умолчанию

qty\_widget.setFlags(qty\_widget.flags() & ~QtCore.Qt.ItemIsEditable)

self.cart\_table.setItem(row\_position, 1, qty\_widget)

def create\_product\_content(self):

self.create\_navigation\_buttons()

self.product\_table = QTableWidget(self)

self.product\_table.setGeometry(210, 150, 770, 500)

self.product\_table.setColumnCount(6) # 6 столбцов

self.product\_table.setHorizontalHeaderLabels(

["Наименование", "Характеристики", "Длительность лицензии", "Цена", "Количество", "Описание"])

# Кнопки для работы с товарами

self.add\_product\_button = QPushButton('Добавить товар', self)

self.add\_product\_button.setGeometry(1000, 250, 180, 50)

self.add\_product\_button.clicked.connect(self.add\_product)

self.edit\_product\_button = QPushButton('Редактировать товар', self)

self.edit\_product\_button.setGeometry(1000, 350, 180, 50)

self.edit\_product\_button.clicked.connect(self.edit\_product)

self.delete\_product\_button = QPushButton('Удалить товар', self)

self.delete\_product\_button.setGeometry(1000, 450, 180, 50)

self.delete\_product\_button.clicked.connect(self.delete\_product)

# Кнопка "Добавить в корзину"

self.add\_to\_cart\_button = QPushButton("Добавить в корзину", self)

self.add\_to\_cart\_button.setGeometry(1000, 550, 180, 50)

self.add\_to\_cart\_button.clicked.connect(self.add\_to\_cart)

self.load\_products()

def add\_to\_cart(self):

current\_row = self.product\_table.currentRow()

if current\_row < 0:

QMessageBox.warning(self, "Ошибка", "Пожалуйста, выберите товар для добавления в корзину.")

return

product\_name = self.product\_table.item(current\_row, 0).text() # Получаем имя товара

cart\_window = self.windows[1] # Получаем окно корзины

cart\_window.load\_cart\_data\_from\_catalog(product\_name) # Добавляем товар в корзину

cart\_window.calculate\_total() # Пересчитываем итог

QMessageBox.information(self, "Успех", f"Товар '{product\_name}' добавлен в корзину.")

self.go\_to\_window(1) # Переходим в корзину

def create\_sales\_content(self):

self.create\_navigation\_buttons()

# Таблица для продаж

self.sales\_table = QTableWidget(self)

self.sales\_table.setGeometry(210, 150, 770, 500)

self.sales\_table.setColumnCount(7)

self.sales\_table.setHorizontalHeaderLabels(

["ID", "Товар", "Количество", "Сумма", "Дата продажи", "Фамилия сотрудника", "Имя сотрудника"])

# Кнопка для очистки таблицы продаж

self.clear\_sales\_button = QPushButton("Очистить продажи", self)

self.clear\_sales\_button.setGeometry(210, 670, 180, 50)

self.clear\_sales\_button.clicked.connect(self.clear\_sales\_table)

# Загружаем данные о продажах

self.load\_sales()

def clear\_sales\_table(self):

*"""Очищает таблицу продаж и данные из базы."""*

try:

self.cursor.execute("DELETE FROM sales")

self.connection.commit()

self.sales\_table.setRowCount(0)

QMessageBox.information(self, "Успех", "Таблица продаж и данные в базе данных очищены.")

logging.info("Sales table and data cleared successfully from DB.")

except Exception as e:

self.connection.rollback()

QMessageBox.warning(self, "Ошибка", f"Не удалось очистить продажи из базы данных: {e}")

logging.error(f"Error deleting sales data from DB: {e}")

def load\_products(self):

try:

self.cursor.execute("""

SELECT p.id, p.name, p.composition, p.price, p.expiration, p.quantity, p.description, s.software\_name

FROM products p

LEFT JOIN product\_software ps ON p.id = ps.product\_id

LEFT JOIN software s ON ps.software\_id = s.software\_id

""")

products = self.cursor.fetchall() # Получаем все строки

self.product\_table.setRowCount(0) # Очищаем предыдущие данные

for product in products:

row\_position = self.product\_table.rowCount()

self.product\_table.insertRow(row\_position)

self.product\_table.setItem(row\_position, 0, QTableWidgetItem(product[1])) # name

self.product\_table.setItem(row\_position, 1, QTableWidgetItem(product[2])) # composition

self.product\_table.setItem(row\_position, 2, QTableWidgetItem(str(product[3]))) # price

if isinstance(product[4], (date, datetime)):

self.product\_table.setItem(row\_position, 3,

QTableWidgetItem(product[4].strftime("%Y-%m-%d"))) # expiration

else:

self.product\_table.setItem(row\_position, 3, QTableWidgetItem(str(product[4]))) # expiration

self.product\_table.setItem(row\_position, 4, QTableWidgetItem(str(product[5]))) # quantity

self.product\_table.setItem(row\_position, 5, QTableWidgetItem(product[6])) # description

self.product\_table.setItem(row\_position, 6, QTableWidgetItem(str(product[7]) if product[7] else "")) # software

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось загрузить товары: {e}")

logging.error(f"Error loading products: {e}")

def add\_product(self):

dialog = ProductDialog(connection=self.connection) #передаем connection

if dialog.exec\_() == QDialog.Accepted:

product\_data = dialog.get\_product\_data()

logging.debug(f"Data from ProductDialog: {product\_data}")

# Валидация данных

if not product\_data["name"] or not product\_data["composition"] or not product\_data["price"] or not product\_data["expiration"] or not product\_data["quantity"] or not product\_data["description"]:

QMessageBox.warning(self, "Ошибка", "Все поля должны быть заполнены.")

logging.warning("Not all fields are filled.")

return

try:

price = float(product\_data["price"])

quantity = int(product\_data["quantity"])

expiration\_date = datetime.strptime(product\_data["expiration"], "%Y-%m-%d").date()

software\_id = product\_data["software\_id"]

except ValueError as e:

QMessageBox.warning(self, "Ошибка", f"Некорректные данные: {e}")

logging.warning(f"Invalid input {e}")

return

try:

row\_position = self.product\_table.rowCount()

self.product\_table.insertRow(row\_position)

self.product\_table.setItem(row\_position, 0, QTableWidgetItem(product\_data["name"]))

self.product\_table.setItem(row\_position, 1, QTableWidgetItem(product\_data["composition"]))

self.product\_table.setItem(row\_position, 2, QTableWidgetItem(str(price)))

self.product\_table.setItem(row\_position, 3, QTableWidgetItem(expiration\_date.strftime("%Y-%m-%d")))

self.product\_table.setItem(row\_position, 4, QTableWidgetItem(str(quantity)))

self.product\_table.setItem(row\_position, 5, QTableWidgetItem(product\_data["description"]))

self.product\_table.setItem(row\_position, 6, QTableWidgetItem(str(software\_id) if software\_id else ""))

self.cursor.execute("""

INSERT INTO products (name, composition, price, expiration, quantity, description)

VALUES (%s, %s, %s, %s, %s, %s)

RETURNING id;

""", (

product\_data["name"],

product\_data["composition"],

price,

expiration\_date,

quantity,

product\_data["description"]

))

product\_id = self.cursor.fetchone()[0]

if software\_id:

self.cursor.execute(

"INSERT INTO product\_software (product\_id, software\_id) VALUES (%s, %s)",

(product\_id, software\_id)

)

self.connection.commit()

QMessageBox.information(self, "Успех", "Товар успешно добавлен в базу данных.")

logging.info("Product added successfully to DB.")

self.load\_products()

except Exception as e:

self.connection.rollback()

QMessageBox.warning(self, "Ошибка", f"Не удалось добавить товар в базу данных: {e}")

logging.error(f"Error adding product to database: {e}")

def closeEvent(self, event):

# Закрываем соединение при закрытии приложения

self.cursor.close()

self.connection.close()

def edit\_product(self):

current\_row = self.product\_table.currentRow()

if current\_row < 0:

QMessageBox.warning(self, "Ошибка", "Пожалуйста, выберите товар для редактирования.")

return

try:

# Получаем ID товара из базы данных по имени товара

product\_name = self.product\_table.item(current\_row, 0).text()

self.cursor.execute("SELECT id FROM products WHERE name = %s", (product\_name,))

product\_id = self.cursor.fetchone()[0]

print(f"product\_id: {product\_id}")

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось получить ID товара: {e}")

logging.error(f"Ошибка получения ID товара: {e}")

return

try:

name = self.product\_table.item(current\_row, 0).text()

composition = self.product\_table.item(current\_row, 1).text()

price = self.product\_table.item(current\_row, 2).text()

expiration = self.product\_table.item(current\_row, 3).text()

quantity = self.product\_table.item(current\_row, 4).text()

description = self.product\_table.item(current\_row, 5).text()

product\_data = {

"id": product\_id,

"name": name,

"composition": composition,

"price": price,

"expiration": expiration,

"quantity": quantity,

"description": description

}

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось получить данные товара: {e}")

logging.error(f"Ошибка получения данных товара: {e}")

return

dialog = ProductDialog(product\_data, connection=self.connection) # передаем connection

if dialog.exec\_() == QDialog.Accepted:

updated\_data = dialog.get\_product\_data()

if updated\_data:

print(f"Данные из диалога: {updated\_data}")

try:

updated\_data["price"] = float(updated\_data["price"])

updated\_data["quantity"] = int(updated\_data["quantity"])

expiration\_date = datetime.strptime(updated\_data["expiration"], "%Y-%m-%d").date()

except ValueError as e:

QMessageBox.warning(self, "Ошибка", f"Некорректный формат данных: {e}")

logging.error(f"Ошибка преобразования данных: {e}")

return

try:

self.product\_table.setItem(current\_row, 0, QTableWidgetItem(updated\_data["name"]))

self.product\_table.setItem(current\_row, 1, QTableWidgetItem(updated\_data["composition"]))

self.product\_table.setItem(current\_row, 2, QTableWidgetItem(str(updated\_data["price"])))

self.product\_table.setItem(current\_row, 3, QTableWidgetItem(expiration\_date.strftime("%Y-%m-%d")))

self.product\_table.setItem(current\_row, 4, QTableWidgetItem(str(updated\_data["quantity"])))

self.product\_table.setItem(current\_row, 5, QTableWidgetItem(updated\_data["description"]))

self.product\_table.setItem(current\_row, 6, QTableWidgetItem(

str(updated\_data["software\_id"] if updated\_data["software\_id"] else "")))

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось установить данные товара в таблицу: {e}")

logging.error(f"Ошибка установки данных товара в таблицу: {e}")

return

try:

self.cursor.execute("""

UPDATE products SET name = %s, composition = %s, price = %s, expiration = %s, quantity = %s, description = %s

WHERE id = %s

""", (

updated\_data["name"],

updated\_data["composition"],

updated\_data["price"],

expiration\_date,

updated\_data["quantity"],

updated\_data["description"],

product\_id

))

self.cursor.execute("DELETE FROM product\_software WHERE product\_id = %s", (product\_id,))

if updated\_data["software\_id"]:

self.cursor.execute("INSERT INTO product\_software (product\_id, software\_id) VALUES (%s, %s)",

(product\_id, updated\_data["software\_id"]))

self.connection.commit()

QMessageBox.information(self, "Успех", "Товар успешно обновлён.")

except Exception as e:

self.connection.rollback()

QMessageBox.warning(self, "Ошибка", f"Не удалось обновить товар в базе данных: {e}")

logging.error(f"Ошибка обновления товара в БД: {e}")

self.load\_products()

else:

print("Данные не получены")

else:

print("Диалог отменен")

def delete\_product(self):

current\_row = self.product\_table.currentRow()

if current\_row < 0: # Проверка, выбрана ли строка

QtWidgets.QMessageBox.warning(self, "Ошибка", "Пожалуйста, выберите товар для удаления.")

return

product\_name = self.product\_table.item(current\_row, 0).text()

logging.debug(f"Deleting product with name: {product\_name}")

reply = QMessageBox.question(self, 'Подтверждение удаления', f'Вы уверены, что хотите удалить {product\_name}?',

QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

if reply == QMessageBox.Yes:

# Удаление товара из базы данных

try:

self.cursor.execute("DELETE FROM products WHERE name = %s", (product\_name,))

self.connection.commit() # Сохраняем изменения

# Удаляем из таблицы

self.product\_table.removeRow(current\_row)

QMessageBox.information(self, "Успех", "Товар успешно удалён.")

logging.info("Product deleted successfully from DB.")

except Exception as e:

self.connection.rollback() # Откатываем изменения в случае ошибки

QMessageBox.warning(self, "Ошибка", f"Не удалось удалить товар из базы данных: {e}")

logging.error(f"Error deleting product from database: {e}")

def create\_staff\_content(self):

self.create\_navigation\_buttons()

self.staff\_table = QTableWidget(self)

self.staff\_table.setGeometry(210, 150, 770, 500) # Задаем размер таблицы

self.staff\_table.setColumnCount(7) # 6 столбцов

self.staff\_table.setHorizontalHeaderLabels(

["Фамилия", "Имя", "Отчество", "Дата Рождения", "Телефон", "Логин", "Пароль"])

# Кнопки для работы с персоналом

self.add\_staff\_button = QPushButton('Добавить сотрудника', self)

self.add\_staff\_button.setGeometry(1000, 250, 180, 50) # Задаем позицию

self.add\_staff\_button.clicked.connect(self.add\_staff)

self.edit\_staff\_button = QPushButton('Редактировать сотрудника', self)

self.edit\_staff\_button.setGeometry(1000, 350, 180, 50) # Задаем позицию

self.edit\_staff\_button.clicked.connect(self.edit\_staff)

self.delete\_staff\_button = QPushButton('Удалить сотрудника', self)

self.delete\_staff\_button.setGeometry(1000, 450, 180, 50) # Задаем позицию

self.delete\_staff\_button.clicked.connect(self.delete\_staff)

self.load\_staff()

def load\_staff(self):

try:

self.cursor.execute(

"SELECT id, last\_name, first\_name, middle\_name, birth\_date, phone, login, password FROM staff;")

staff = self.cursor.fetchall()

self.staff\_table.setRowCount(0)

self.staff\_table.setColumnCount(8) # 8 столбцов

self.staff\_table.setHorizontalHeaderLabels(

["Фамилия", "Имя", "Отчество", "Дата Рождения", "Телефон", "Логин", "Пароль", "ID"])

for staff in staff:

row\_position = self.staff\_table.rowCount()

self.staff\_table.insertRow(row\_position)

self.staff\_table.setItem(row\_position, 0, QTableWidgetItem(staff[1]))

self.staff\_table.setItem(row\_position, 1, QTableWidgetItem(staff[2]))

self.staff\_table.setItem(row\_position, 2, QTableWidgetItem(staff[3]))

self.staff\_table.setItem(row\_position, 3,

QTableWidgetItem(staff[4].strftime("%Y-%m-%d")))

self.staff\_table.setItem(row\_position, 4, QTableWidgetItem(staff[5]))

self.staff\_table.setItem(row\_position, 5, QTableWidgetItem(staff[6]))

self.staff\_table.setItem(row\_position, 6, QTableWidgetItem(staff[7]))

self.staff\_table.setItem(row\_position, 7, QTableWidgetItem(str(staff[0]))) # id

except Exception as e:

QMessageBox.warning(self, "Ошибка", f"Не удалось загрузить персонал: {e}")

def add\_staff(self):

dialog = StaffDialog()

if dialog.exec\_() == QDialog.Accepted:

staff\_data = dialog.get\_staff\_data()

# Валидация номера телефона

if not is\_valid\_phone(staff\_data["phone"]):

QMessageBox.warning(self, "Ошибка ввода", "Некорректный номер телефона. Формат: +7 (XXX) XXX XX XX")

return

row\_position = self.staff\_table.rowCount()

self.staff\_table.insertRow(row\_position) # Добавляем новую строку

self.staff\_table.setItem(row\_position, 0, QTableWidgetItem(staff\_data["last\_name"]))

self.staff\_table.setItem(row\_position, 1, QTableWidgetItem(staff\_data["first\_name"]))

self.staff\_table.setItem(row\_position, 2, QTableWidgetItem(staff\_data["middle\_name"]))

self.staff\_table.setItem(row\_position, 3, QTableWidgetItem(staff\_data["birth\_date"]))

self.staff\_table.setItem(row\_position, 4, QTableWidgetItem(staff\_data["phone"])) # Номер телефона

self.staff\_table.setItem(row\_position, 5, QTableWidgetItem(staff\_data["login"]))

self.staff\_table.setItem(row\_position, 6, QTableWidgetItem(staff\_data["password"]))

# Запись данных в базу данных

try:

self.cursor.execute("""

INSERT INTO staff (last\_name, first\_name, middle\_name, birth\_date, phone, login, password)

VALUES (%s, %s, %s, %s, %s, %s, %s)

""", (

staff\_data["last\_name"],

staff\_data["first\_name"],

staff\_data["middle\_name"],

staff\_data["birth\_date"],

staff\_data["phone"], # Сохраняем номер телефона

staff\_data["login"],

staff\_data["password"]

))

self.connection.commit() # Сохраняем изменения

QMessageBox.information(self, "Успех", "Сотрудник успешно добавлен в базу данных.")

except Exception as e:

self.connection.rollback() # Откатываем изменения в случае ошибки

QMessageBox.warning(self, "Ошибка", f"Не удалось добавить сотрудник в базу данных: {e}")

def edit\_staff(self):

current\_row = self.staff\_table.currentRow()

if current\_row < 0:

QMessageBox.warning(self, "Ошибка", "Пожалуйста, выберите сотрудника для редактирования.")

return

staff\_id = self.staff\_table.item(current\_row, 7).text()

staff\_data = {

"last\_name": self.staff\_table.item(current\_row, 0).text(),

"first\_name": self.staff\_table.item(current\_row, 1).text(),

"middle\_name": self.staff\_table.item(current\_row, 2).text(),

"birth\_date": self.staff\_table.item(current\_row, 3).text(),

"phone": self.staff\_table.item(current\_row, 4).text(),

"login": self.staff\_table.item(current\_row, 5).text(),

"password": self.staff\_table.item(current\_row, 6).text(),

}

dialog = StaffDialog(staff\_data)

if dialog.exec\_() == QDialog.Accepted:

updated\_data = dialog.get\_staff\_data()

if not is\_valid\_phone(updated\_data["phone"]):

QMessageBox.warning(self, "Ошибка ввода", "Некорректный номер телефона. Формат: +7 (XXX) XXX XX XX")

return

self.staff\_table.setItem(current\_row, 0, QTableWidgetItem(updated\_data["last\_name"]))

self.staff\_table.setItem(current\_row, 1, QTableWidgetItem(updated\_data["first\_name"]))

self.staff\_table.setItem(current\_row, 2, QTableWidgetItem(updated\_data["middle\_name"]))

self.staff\_table.setItem(current\_row, 3, QTableWidgetItem(updated\_data["birth\_date"]))

self.staff\_table.setItem(current\_row, 4, QTableWidgetItem(updated\_data["phone"]))

self.staff\_table.setItem(current\_row, 5, QTableWidgetItem(updated\_data["login"]))

self.staff\_table.setItem(current\_row, 6, QTableWidgetItem(updated\_data["password"]))

try:

self.cursor.execute("""

UPDATE staff SET last\_name = %s, first\_name = %s, middle\_name = %s,

birth\_date = %s, phone = %s, login = %s, password = %s

WHERE id = %s

""", (

updated\_data["last\_name"],

updated\_data["first\_name"],

updated\_data["middle\_name"],

updated\_data["birth\_date"],

updated\_data["phone"],

updated\_data["login"],

updated\_data["password"],

staff\_id

))

self.connection.commit()

QMessageBox.information(self, "Успех", "Сотрудник успешно обновлён.")

except Exception as e:

self.connection.rollback()

QMessageBox.warning(self, "Ошибка", f"Не удалось обновить сотрудника в базе данных: {e}")

logging.error(f"Error editing staff {e}")

def delete\_staff(self):

current\_row = self.staff\_table.currentRow()

if current\_row < 0:

QMessageBox.warning(self, "Ошибка", "Пожалуйста, выберите сотрудника для удаления.")

return

staff\_id = self.staff\_table.item(current\_row, 7).text()

staff\_name = f"{self.staff\_table.item(current\_row, 0).text()} {self.staff\_table.item(current\_row, 1).text()}"

reply = QMessageBox.question(self, 'Подтверждение удаления',

f'Вы уверены, что хотите удалить сотрудника {staff\_name}?',

QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

if reply == QMessageBox.Yes:

try:

self.cursor.execute("DELETE FROM staff WHERE id = %s", (staff\_id,))

self.connection.commit()

self.staff\_table.removeRow(current\_row)

QMessageBox.information(self, "Успех", "Сотрудник успешно удалён.")

except Exception as e:

self.connection.rollback()

QMessageBox.warning(self, "Ошибка", f"Не удалось удалить сотрудника из базы данных: {e}")

logging.error(f"Error deleting staff {e}")

def on\_order\_button\_click(self):

# Получение выбранных товаров и перенос в корзину

selected\_items = self.product\_catalog.selectedItems()

if selected\_items:

cart\_window = self.windows[1]

cart\_window.cart\_table.setRowCount(0) # Очищаем корзину

for item in selected\_items:

cart\_window.load\_cart\_data\_from\_catalog(item.text())

cart\_window.calculate\_total() # Пересчитываем итог

self.go\_to\_window(1) # Переход к окну корзины

def go\_to\_window(self, idx):

self.hide() # Скрываем текущее окно

self.windows[idx].show() # Показываем следующее окно

def set\_other\_windows(self, windows):

self.windows = windows # Сохраняем ссылки на другие окна

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

total\_windows = 6

windows = [Window(i, total\_windows) for i in range(total\_windows)]

# Устанавливаем ссылки на другие окна

for w in windows:

w.set\_other\_windows(windows)

windows[0].show() # Показываем первое окно

sys.exit(app.exec\_())