Министерство образования Республики Беларусь

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

К ЗАЩИТЕ ДОПУСТИТЬ

_____ *А. М. Ковальчук*

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
ИНФОРМАЦИОННАЯ СИСТЕМА СУПЕРМАРКЕТ

БГУИР КП 1–40 02 01 215 ПЗ

Студент: Копейкина В.А.

Руководитель: Ассистент кафедры ЭВМ
Богдан Е.В.

Минск 2023

Учреждение образования

«Белорусский государственный университет информатики

и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____
(подпись)
_____2023 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Копейкиной Виктории Анатольевне*

Тема проекта «*Система заказов в интернет-магазине*»

2. Срок сдачи студентом законченного проекта *15 декабря 2023 г.*

3. Исходные данные к проекту *Язык программирования – C++, среда разработки – Qt-Creator*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

1. Лист задания.
2. Введение.
3. Обзор литературы.
   3.1. Обзор методов и алгоритмов решения поставленной задачи.
4. Функциональное проектирование.
   4.1. Структура входных и выходных данных.
   4.2. Разработка диаграммы классов.
   4.3. Описание классов.
5. Разработка программных модулей.
   5.1. Разработка схем алгоритмов (два наиболее важных метода).
   5.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов).

6. Результаты работы.
7. Заключение
8. Литература
9. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Диаграмма классов.

2. Схема метода on_load().

3. Схема метода sell_order().

6. Консультант по проекту (с обозначением разделов проекта) Е.В. Богдан

7. Дата выдачи задания *15.09.2023 г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

1. Выбор задания. Разработка содержания пояснительной записки.

Перечень графического материала – 15 %;

разделы 2, 3 – 10 %;

разделы 4 к – 20 %;

разделы 5 к – 35 %;

раздел 6,7,8 – 5 %;

раздел 9 к – 5%;

оформление пояснительной записки и графического материала к 15.12.22 – 10 %

Защита курсового проекта с 21.12 по 28.12.23г.

РУКОВОДИТЕЛЬ Е.В. Богдан _____

(подпись)

Задание принял к исполнению _____ Копейкина В.А.

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

C++ позволяет с легкостью работать с памятью, управлять ресурсами и использовать современные подходы к разработке ПО. С несколькими десятилетиями истории и активной поддержкой со стороны сообщества разработчиков, C++ остается эффективным выбором для разработки прикладного программного обеспечения. Небольшое описание современных возможностей, доступных при разработке приложений с использованием C++:

- Стандартная библиотека: C++ поставляется с богатой стандартной библиотекой, которая включает в себя контейнеры, алгоритмы, ввод/вывод, многопоточность и многие другие компоненты. Это позволяет разработчикам создавать приложения более эффективно, используя готовые решения.
- Высокая производительность: C++ известен своей высокой производительностью благодаря близкому к аппаратному уровню управлению памятью и оптимизациям, которые предоставляют компиляторы.
- Поддержка множества платформ: C++ поддерживается на различных операционных системах и архитектурах, что делает его универсальным выбором для разработки.
- Современный стандарт языка: Последние стандарты C++ внесли множество улучшений и новых возможностей в язык, включая умные указатели, диапазоны и многое другое.
- Безопасность: Современные стандарты C++ также уделяют внимание безопасности, предоставляя средства для уменьшения риска ошибок в коде, такие как проверка границ массивов.

Описанные выше возможности позволяют программистам разрабатывать сложное и производительное ПО с использованием C++.

Qt Creator является мощной интегрированной средой разработки (IDE), специально разработанной для создания приложений с использованием фреймворка Qt.

Qt Creator предоставляет разработчикам удобное и эффективное рабочее окружение, которое объединяет в себе множество инструментов для разработки, отладки и профилирования приложений. Он поддерживает различные языки программирования, включая C++, QML и Python, что позволяет создавать кросс-платформенные приложения с привлекательным пользовательским интерфейсом.

1 ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать функциональную систему заказов для интернет-магазина, которая предоставит пользователям возможность заказывать товары онлайн, обеспечивая удобство и эффективность процесса. Программа “Система заказов в интернет-магазине” должна включать следующий набор функций:

- Регистрация в личном кабинете. Создание личного аккаунта с уникальным логином и паролем.
- Поиск товара по фильтру и названию. Поиск товаров с использованием фильтров, таких как категории, цена и другие параметры, а также по названию товара.
- Добавление и удаление товаров из корзины. Зарегистрированные пользователи могут добавлять товары в корзину и удалять их по своему усмотрению.
- Расчет общей стоимости заказа. Система должна автоматически вычислять общую стоимость заказа на основе добавленных товаров и их количества.

Разработка и внедрение данной системы заказов интернет-магазина обеспечит следующие преимущества для пользователей и владельцев магазина:

- Удобство для покупателей. Пользователи смогут легко находить нужные товары, управлять корзиной, выбирать способ доставки, и видеть общую сумму заказа, что сделает процесс покупок более удобным и прозрачным.
- Эффективность обслуживания. Владельцы магазина смогут автоматизировать процесс обработки заказов, расчета скидок и учета товаров в корзине, что позволит им предоставлять более эффективное обслуживание клиентов.
- Увеличение продаж. Удобный процесс заказа могут стимулировать клиентов к совершению покупок и увеличению среднего чека.

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Обзор методов и алгоритмов решения поставленной задачи

Для уникальности объекта используется шаблон template < class T>. Этот шаблон имеет метод класса, возвращающий ссылку на единый экземпляр объекта типа T, реализующий создание и возвращение единственного экземпляра объекта. Также данный класс содержит оператор присваивания, чтобы предотвратить присваивание экземпляров.

Листинг кода программы находится в приложении А.

2.2 Анализ существующих аналогов

Тема курсового проекта была выбрана с целью освоения навыков разработки интернет-магазина с использованием фреймворка Qt и языка SQL для работы с базой данных.

Интернет-магазины являются актуальными и востребованными в современном мире, поскольку все больше людей предпочитают делать покупки онлайн. Для создания корректно работающего интернет-магазина с использованием Qt и SQL необходимо иметь представление о существующих аналогах.

Нужно также проанализировать существующие решения интернет-магазинов, реализованных с использованием Qt и SQL, чтобы определить, какие функциональности и особенности могут быть полезны в создаваемом приложении.

2.2.1 Приложение Wildberries

Wildberries — многофункциональное приложение для мобильных устройств, предоставляющее удобную платформу для покупок онлайн. С помощью приложения Wildberries пользователи могут легко находить и заказывать различные товары из широкого ассортимента.

Приложение Wildberries предлагает множество преимуществ. Оно обеспечивает удобный интерфейс и понятную навигацию, что делает процесс покупок простым и приятным. Пользователи могут легко просматривать товары по категориям, использовать фильтры для уточнения результатов поиска и сортировать товары по различным параметрам.

Данное приложение также предлагает удобные способы оплаты, включая онлайн-платежи и оплату при получении товара. Это обеспечивает гибкость и удобство при совершении покупок.

Синхронизация файлов и данных между устройствами на базе Android является еще одним преимуществом приложения Wildberries. Это позволяет пользователям сохранять свои предпочтения, историю заказов и корзину

покупок на всех своих устройствах, обеспечивая безопасность и удобство использования.

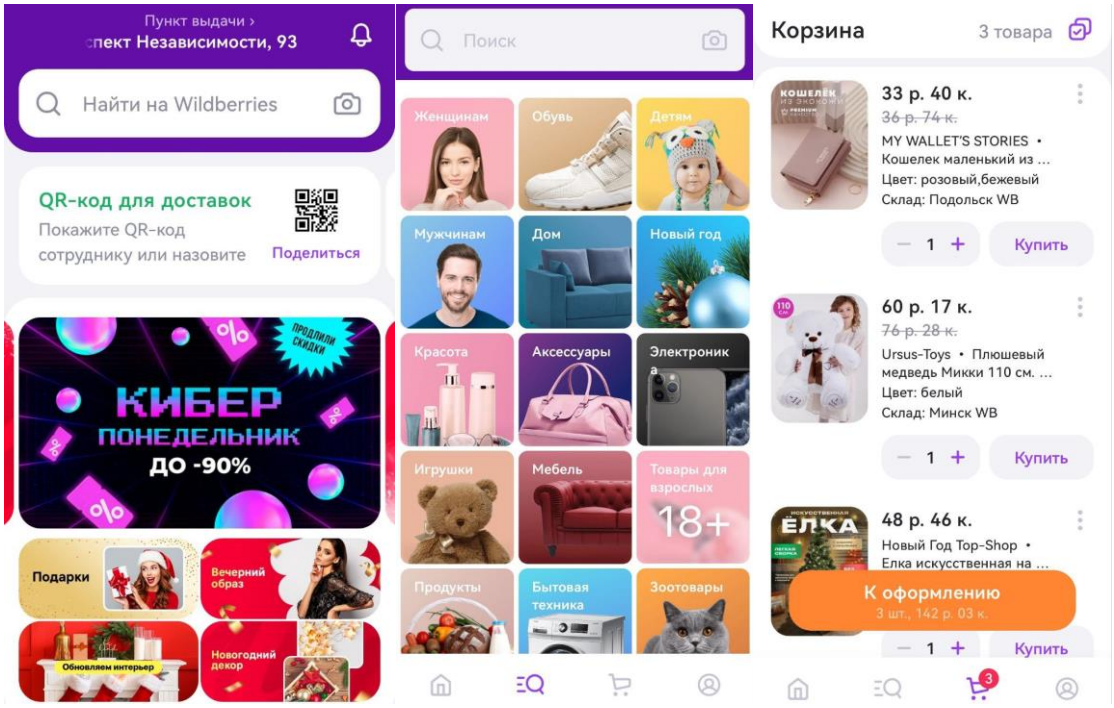

Рисунок 2.1 – Скриншоты Wildberries

2.2.2 Приложение Ozon

Ozon — инновационное приложение для мобильных устройств, предоставляющее широкий выбор товаров и удобную платформу для онлайн-покупок. С помощью приложения Ozon пользователи могут легко находить и заказывать разнообразные товары из различных категорий.

Приложение Ozon предлагает удобный и интуитивно понятный интерфейс, который обеспечивает легкую навигацию и приятный опыт покупок. Пользователи могут использовать удобные функции фильтрации, чтобы уточнить результаты поиска и быстро найти нужные товары. Кроме того, функциональность "Рекомендации" помогает пользователям открывать новые товары и предложения, основываясь на их предпочтениях и истории покупок.

Одним из ключевых преимуществ Ozon является простой и безопасный процесс оформления заказа и оплаты. Пользователи могут выбирать удобные способы оплаты, включая онлайн-платежи и оплату при получении товара. Кроме того, Ozon предлагает надежную систему доставки, что обеспечивает оперативное и надежное получение заказанных товаров.

Данное приложение также предлагает программу лояльности, которая позволяет пользователям получать дополнительные бонусы, скидки и специальные предложения. Это создает стимул для регулярных покупок и обеспечивает дополнительные выгоды для пользователей.

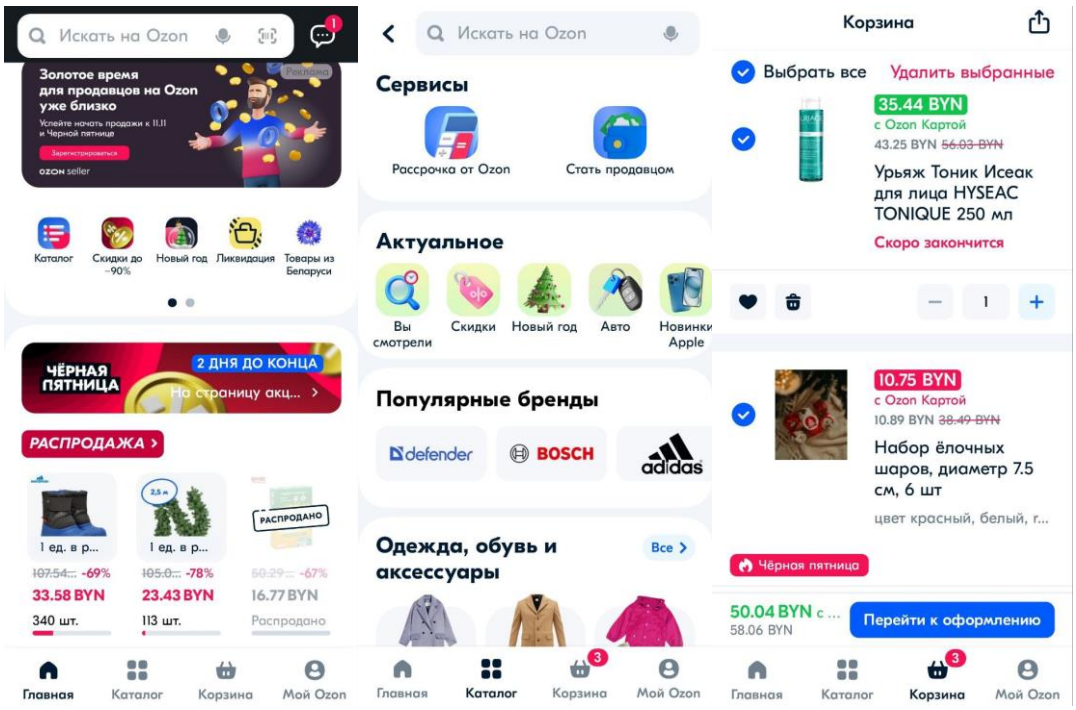

Рисунок 2.2 – Скриншоты Ozon

2.2.3 Приложение OZ.by

OZ.by — это интернет-магазин, предлагающий широкий ассортимент товаров, включая книги, игры, косметику, товары для дома, творчества, подарки и продукты. Он также предлагает доставку по Беларуси. OZ.by является надежным и удобным местом для покупки товаров онлайн.

OZ.by предлагает удобный интерфейс и простую загрузку книг и других товаров. Вы можете легко найти и приобрести нужные товары, выбрав из широкого ассортимента. OZ.by также предлагает синхронизацию файлов для Android-устройств, что делает использование интернет-магазина еще более удобным.

В интернет-магазине OZ.by вы найдете разнообразные книги, включая художественную литературу, нехудожественную литературу, бизнес-литературу и детскую литературу. Вы можете выбрать книги различных жанров, включая зарубежную и русскую литературу, фантастику, детективы и многое другое. OZ.by предлагает новинки книжного мира и топовые книги, чтобы удовлетворить любые литературные предпочтения.

Кроме книг, OZ.by также предлагает широкий выбор товаров для дома. Вы можете приобрести посуду, кухонные принадлежности, предметы интерьера, товары для сада, хозяйственные товары и многое другое. OZ.by предлагает различные бренды и варианты товаров, чтобы удовлетворить потребности в домашнем обустройстве.

В интернет-магазине OZ.by вы также найдете разнообразные товары косметики и парфюмерии. Здесь представлены средства для ухода за лицом, телом, волосами, руками и ногами. Вы можете выбрать декоративную

косметику, парфюмерию и другие товары, чтобы подчеркнуть свою красоту и заботиться о себе.

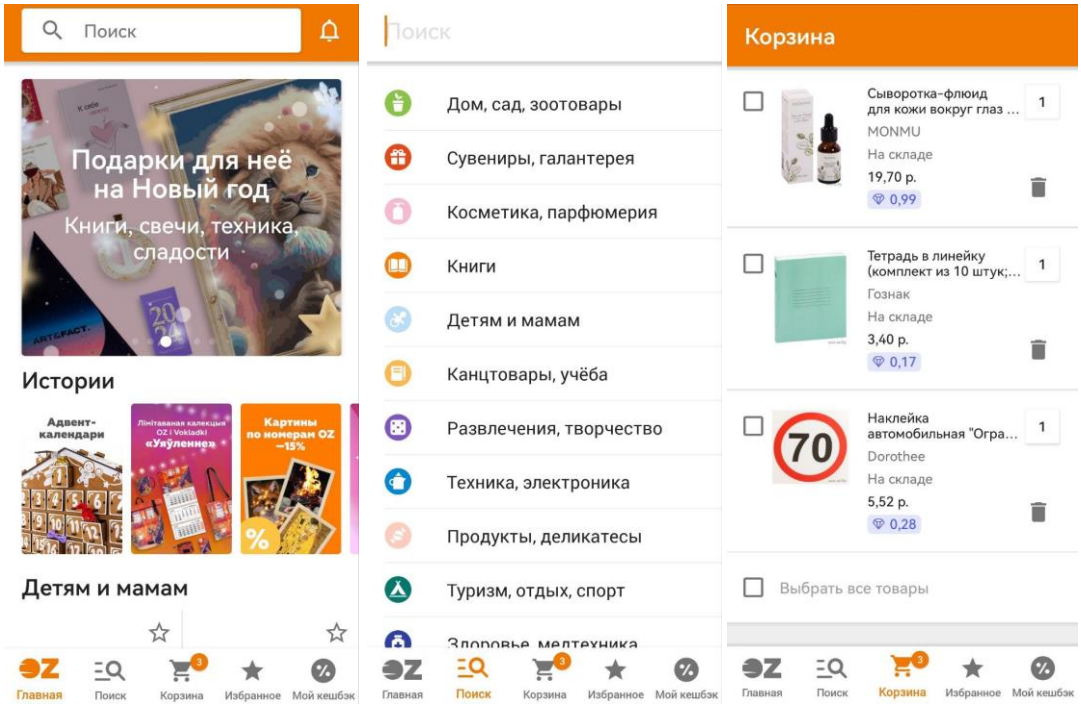

Рисунок 2.3 – Скриншоты OZ.by

2.3 Требования к работе программы

После рассмотрения аналогов различных интернет-магазинов становится понятно, что подобного рода приложения обычно включают в себя основные функции:

- Отображение каталога товаров, представленных в интернет-магазине.
- Обработка и отображение информации о каждом товаре, включая название, описание, цену, изображения и другие характеристики.
- Предоставление возможности добавления товаров в корзину для последующего оформления заказа.
- Отображение информации о доступности товаров, скидках или акциях.
- Предоставление информации о способах оплаты, доставке и возврате товаров.
- Различные способы коммуникации с поддержкой клиентов, включая онлайн-чат, электронную почту или телефон.
- Адаптивный дизайн, который обеспечивает удобство использования интернет-магазина на различных устройствах, включая компьютеры, планшеты и смартфоны.

Для реализации интернет-магазина был выбран язык программирования C++ с использованием фреймворка Qt и языка SQL.

Преимущества C++ включают его высокую производительность и широкие возможности. Благодаря низкоуровневым возможностям языка, C++ обеспечивает более прямой доступ к системным ресурсам и позволяет управлять памятью и процессорными ресурсами более эффективно. Это особенно важно для интернет-магазина, который может иметь большой объем данных и высокую нагрузку.

Фреймворк Qt предоставляет набор инструментов для разработки графического интерфейса пользователя. Он обеспечивает кросс-платформенность, что позволяет создавать приложения, работающие на различных операционных системах, таких как Windows, macOS и Linux. Qt также предоставляет удобные средства для работы с сетью, базами данных и другими функциональными возможностями, необходимыми для реализации интернет-магазина.

Язык SQL (Structured Query Language) используется для работы с базами данных. Он позволяет создавать, изменять и управлять данными в базе данных. SQL является стандартным языком запросов для множества баз данных, что делает его универсальным и мощным инструментом для работы с информацией в интернет-магазине.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВНИЕ

3.1 Структура входных и выходных данных

Входными данными в приложении являются файлы формата db, которые мы загружаем из памяти компьютера.

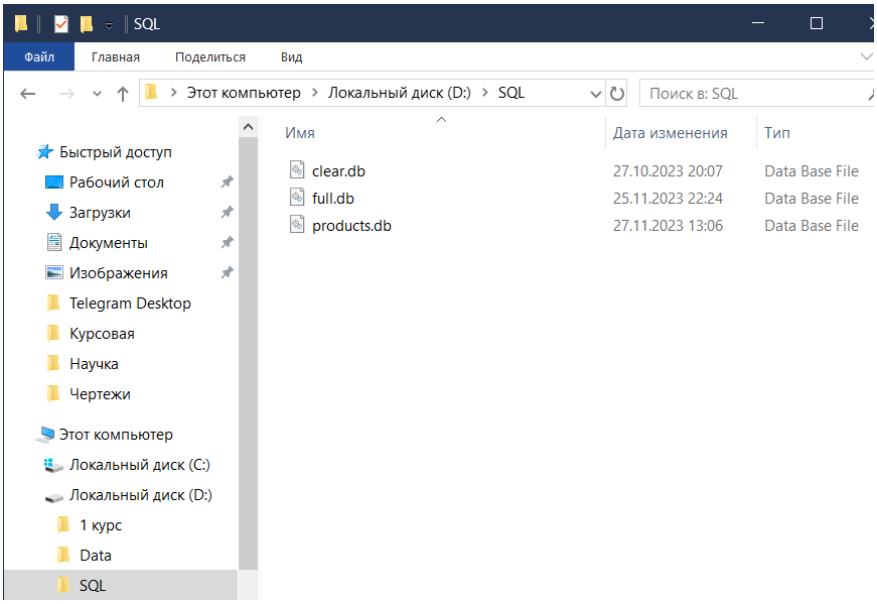

Рисунок 3.1 – Входные файлы формата db

Дальше данные, содержащиеся в файле, записываются в SQLite в таком виде:

Таблица 3.1 – структура информации о товаре в базе данных

users	products	solt_products	products_order
kojom23 vichka25 lerka18	Перец Молоко Чай	Вода Платье Пудра	Игрушка Книга

3.2 Разработка диаграммы классов

Диаграмма классов — это структурный тип диаграммы, используемый в объектно-ориентированном программировании (ООП), чтобы показать классы, их атрибуты и взаимодействия между ними. Она является основным инструментом анализа и проектирования системы в методологии Unified Modeling Language (UML).

Диаграмма классов состоит из прямоугольников, которые представляют классы. Внутри прямоугольников указываются название класса, его атрибуты и методы. Атрибуты — это переменные, которые хранят данные, связанные с классом, а методы — это функции или операции, которые могут быть выполнены классом. Диаграмма классов представлена в приложении Б.

3.3 Описание классов

3.3.1 Каталог покупателя

Класс `CatalogBuyer` представляет собой главное окно каталога товаров для покупателя. Этот класс унаследован от `` `QMainWindow` ``, что делает его основным окном приложения. Данный класс служит основным контроллером для взаимодействия покупателя с каталогом товаров, обеспечивая управление данными, заказами и интерфейсом.

Каталог товаров является обязательной частью любого магазина, он может использоваться для составления различных отчетов и формирования списка востребованных, но заканчивающихся товаров, например. Электронный каталог товаров предполагает хранение соответствующей информации в электронном виде, что позволяет более эффективно обрабатывать ее средствами ЭВМ. Такой каталог обязателен при электронной коммерции, то есть любых продаж товаров и услуг с использованием сети Интернет.

Метод `on_login()` обрабатывает событие входа пользователя в систему.

Данный метод вызывает диалоговое окно для ввода учетных данных и запускает процесс аутентификации.

Метод `auth_ok(User::Role)` предназначен для обработки успешной аутентификации пользователя. Данный метод выполняет действия, связанные с уровнем доступа пользователя (ролью), например, настройка интерфейса в зависимости от его прав.

Метод `on_updated()` используется для обновления данных в каталоге. Данный метод обращается к базе данных, чтобы получить актуальную информацию о товарах.

Метод `on_add()` обрабатывает событие добавления продуктов в заказ.

Данный метод вызывает диалоговое окно для выбора продуктов и добавляет их в соответствующий заказ.

Метод `on_order()` предназначен для отображения заказа покупателя.

Данный метод открывает новое окно для просмотра текущего состояния заказа.

Также имеется указатель `ui` на объект класса `CatalogBuyer`, что предоставляет доступ к элементам пользовательского интерфейса. Это позволяет программно управлять элементами интерфейса, изменять их свойства и обновлять отображаемую информацию.

Указатель `order_buyer` связан с объектом класса `OrderBuyer`, представляющим заказ покупателя. Это позволяет взаимодействовать с заказом, добавлять в него продукты, отслеживать его состояние и, возможно, редактировать информацию о заказе.

3.3.2 Каталог менеджера

Класс `CatalogManagement` предназначен для централизованного управления каталогом, включая управление пользователями, добавление/удаление продуктов, управление заказами и генерацию отчетов о продажах. Он использует другие классы и объекты для конкретных функциональностей, таких как `Users`, `AddProduct`, `OrderManagement` и `SalesReport`. Класс разработан для взаимодействия с пользовательским интерфейсом (`Ui::CatalogManagement`) для графического представления операций управления каталогом.

Метод `on_user_management`() вызывается при нажатии кнопки управления пользователями. Он отображает окно для управления пользователями.

Метод `on_report()` вызывается при нажатии кнопки отображения отчетов. Он отображает окно для просмотра отчетов.

Метод `on_product_remove()` вызывается при нажатии кнопки удаления товаров. Он получает выбранные элементы из таблицы и удаляет соответствующие товары из базы данных.

Метод `on_product_add()` вызывается при нажатии кнопки добавления товаров. Он отображает окно для добавления новых товаров.

Метод `on_order_management()` вызывается при нажатии кнопки управления заказами. Он отображает окно для управления заказами.

Метод `on_product_added(Product product)` вызывается при добавлении нового товара. Он проверяет, существует ли товар с таким же артикулом в базе данных, и если нет, то добавляет новый товар.

Метод `on_updated()` вызывается при обновлении данных в базе данных продуктов. Он получает список продуктов из базы данных и обновляет таблицу в пользовательском интерфейсе с информацией о продуктах.

В классе `CatalogManagement` используются указатели на объекты пользовательского интерфейса (`Ui::CatalogManagement`), объекты для управления пользователями (`Users`), добавления продукта (`AddProduct`), управления заказами (`OrderManagement`) и отчетов (`SalesReport`). Указатели используются для создания экземпляров этих классов и управления их отображением и взаимодействием с другими компонентами приложения.

3.3.3 Обработка заказов

Класс `OrderManagement` является подклассом `QWidget` и представляет собой компонент приложения для управления заказами. Данный класс предоставляет удобный и понятный интерфейс для управления заказами в приложении. Он позволяет загружать, удалять, продавать и получать выбранные заказы, обеспечивая эффективное управление заказами и повышая общий уровень удобства использования приложения.

Метод `on_load()` загружает список заказов из базы данных и отображает их в таблице на пользовательском интерфейсе. Он вызывается при запуске приложения и при обновлении списка заказов.

Метод `on_remove()` удаляет выбранные заказы из базы данных и обновляет список заказов на пользовательском интерфейсе. Он вызывается при нажатии на кнопку "Удалить".

Метод `on_sell()` помечает выбранные заказы как проданные в базе данных и обновляет список заказов на пользовательском интерфейсе. Он вызывается при нажатии на кнопку "Продать".

Метод `selected_orders()` возвращает список выбранных заказов из таблицы на пользовательском интерфейсе. Он используется в методах `on_remove()` и `on_sell()` для получения выбранных заказов и выполнения соответствующих операций с ними.

3.3.4 Оформление заказа

Класс `OrderBuyer` является компонентом приложения, отвечающим за функциональность покупки товаров и создание заказов. Данный класс `OrderBuyer` предоставляет функциональность для покупки товаров и создания заказов. Он позволяет загружать список доступных продуктов, создавать заказы, удалять продукты из заказа и получать информацию о текущем состоянии заказа.

Метод `load(set<string> product_articles)` загружает набор артикулов продуктов, которые представляют доступные продукты, которые могут быть добавлены в заказ. Он добавляет артикулы продуктов в набор `product_articles` и обновляет таблицу в пользовательском интерфейсе для отображения загруженных продуктов.

Метод `current_order()` возвращает текущий набор артикулов продуктов в заказе. Он используется для получения информации о выбранных продуктах в заказе.

Слот `create_order()` вызывается, когда пользователь нажимает кнопку "`ok`" в пользовательском интерфейсе. Он получает ввод электронной почты пользователя, проверяет его на корректность, получает текущую дату, извлекает выбранные продукты из набора `product_articles` и добавляет заказ в базу данных продуктов.

Слот `remove_product()` вызывается, когда пользователь нажимает кнопку "`Удалить`" в пользовательском интерфейсе. Он получает выбранные строки из таблицы, извлекает соответствующие артикулы продуктов и удаляет их из набора `product_articles`. Затем он загружает обновленный набор продуктов в таблицу.

Указатель `Ui::OrderBuyer *ui` используется для доступа к элементам пользовательского интерфейса и их изменения.

Поле `set<string> product_articles` хранит артикулы продуктов, которые в настоящее время находятся в заказе. Оно используется для отслеживания выбранных продуктов в заказе.

В целом, класс `OrderBuyer` предоставляет удобный интерфейс для покупателя, позволяющий ему выбирать и управлять продуктами в заказе. Он интегрируется с базой данных продуктов и обеспечивает взаимодействие покупателя с системой заказов.

3.3.5 Авторизация

Класс `AuthWidget` представляет собой виджет, который используется для аутентификации пользователей. Данный класс предоставляет пользовательский интерфейс для аутентификации пользователей и проверки их учетных данных. Он позволяет пользователям вводить свои учетные данные, проверяет их правильность и уведомляет о успешной аутентификации, передавая роль пользователя.

Слот `enter()` вызывается при нажатии на кнопку "`Войти`" (`ui->enter`). В этом методе происходит получение введенного логина и пароля, проверка правильности введенных данных и отправка сигнала `auth_ok` с информацией о роли пользователя, если данные верны.

Сигнал `void auth_ok(User::Role)` отправляется при успешной аутентификации пользователя. Передает информацию о роли пользователя (`User::Role`).

3.3.6 Добавление товара

Класс `AddProduct` представляет собой виджет, который используется для добавления нового продукта. Данный класс предоставляет пользовательский интерфейс для добавления нового продукта и отправки информации о нем. Он позволяет пользователям вводить информацию о продукте, проверяет ее на корректность и отправляет сигнал с информацией о продукте для дальнейшей обработки.

Сигнал `product(Product product)` отправляет сигнал с информацией о добавленном продукте.

Слот `on_add()` вызывается при нажатии на кнопку "`Добавить`". Он проверяет заполнены ли все поля ввода, создает объект `Product` с данными из полей ввода и отправляет сигнал `product` с этим объектом.

3.3.7 Отчет по продажам

Класс `SalesReport` представляет собой виджет, который используется для отображения отчетов о продажах. Он является подклассом `QWidget` и предоставляет пользовательский интерфейс для взаимодействия с данными отчета.

Метод on_calc() вызывается при нажатии на кнопку "Рассчитать". Он получает значения даты начала и окончания периода из соответствующих полей ввода и использует их для вызова методов count() и sum() класса PRODUCT_DB. Метод count() возвращает количество продуктов, проданных в указанном периоде, а метод sum() возвращает сумму продаж за этот период. Затем полученные значения количества и суммы отображаются в соответствующих полях на пользовательском интерфейсе.

3.3.8 Управление пользователями

Класс Users представляет собой виджет, который используется для управления списком пользователей. Он наследуется от класса QWidget и предоставляет пользовательский интерфейс для добавления, удаления и обновления пользователей. Класс Users предоставляет пользовательский интерфейс для взаимодействия с данными о пользователях, таких как их имена, адреса электронной почты и другая информация. Он может содержать элементы интерфейса, такие как таблицы, поля ввода и кнопки, для удобного управления списком пользователей.

Метод on_remove() получает список выбранных элементов в таблице пользователей; извлекает номера строк выбранных элементов и сохраняет их в множество selected_rows; для каждой выбранной строки получает логин пользователя и добавляет его в множество removed_users; проходит по каждому логину в removed_users и удаляет соответствующего пользователя из базы данных.

Метод on_add() получает логин и пароль нового пользователя из соответствующих полей ввода; проверяет, что логин и пароль не пустые; проверяет, что логин не занят другим пользователем в базе данных; определяет роль нового пользователя в зависимости от состояния флажка "Суперпользователь"; добавляет нового пользователя в базу данных с указанными логином, паролем и ролью.

Метод on_updated() получает список всех пользователей из базы данных; очищает таблицу пользователей; для каждого пользователя создает новую строку в таблице и заполняет ячейки данными о логине и роли пользователя.

3.3.9 Модель пользователя в системе

Класс User представляет собой модель пользователя в системе. Он содержит информацию о логине, пароле и роли пользователя. Основная задача класса User - предоставить функционал для работы с данными пользователя и его ролью в системе.

Метод on_remove() обрабатывает событие нажатия на кнопку "Удалить"; получает список выбранных элементов в таблице пользователей; извлекает информацию о выбранных пользователях и добавляет их во

множество `removed_users` проверяет, является ли выбранный пользователь текущим пользователем. Если да, то пропускает его удаление. Также удаляет выбранных пользователей из базы данных.

Метод `on_add()` обрабатывает событие нажатия на кнопку "`Добавить`"; получает введенные значения логина и пароля из текстовых полей; проверяет, что логин и пароль не пустые; проверяет, что логин не занят другим пользователем в базе данных; определяет роль пользователя в зависимости от выбранного флажка "`isSuper`"; добавляет нового пользователя в базу данных.

Метод `on_updated()` обрабатывает событие обновления базы данных пользователей; получает список всех пользователей из базы данных; очищает таблицу пользователей; добавляет каждого пользователя в таблицу.

3.3.10 Модель товара

Класс `Product` представляет собой модель товара и содержит информацию о его характеристиках.

3.3.11 Дата продажи продукта

Класс `SoltProduct` является производным классом от класса `Product`. Он добавляет новое поле `realization_date` (дата реализации или продажи продукта) и наследует все остальные поля и методы класса `Product`.

3.3.12 Модель заказа

Класс `Order` представляет собой модель заказа, которая содержит информацию о дате заказа, электронной почте заказчика и списке идентификаторов продуктов в заказе.

3.3.13 Фасад базы данных

Класс `DBFacade` представляет собой фасад базы данных и выполняет роль промежуточного слоя между приложением и базой данных. Он предоставляет удобный интерфейс для работы с базой данных и скрывает детали реализации.

Метод `exec` выполняет SQL-запрос к базе данных. Принимает строку с SQL-запросом в качестве параметра. Если выполнение запроса не удалось, выбрасывается исключение `ExecException`.

Метод `qs(QString)` принимает строку в качестве параметра и возвращает эту строку, заключенную в одинарные кавычки. Этот метод используется для правильного форматирования строковых значений в SQL-запросах.

Метод `qs(std::string)` принимает `std::string` в качестве параметра, преобразует его в `QString` и вызывает метод `qs(QString)` для форматирования строки.

Сигнал `updated` испускается, когда база данных обновляется.

Класс `DBFacade` также содержит два пользовательских исключения.

Класс `OpenDBException` - это исключение выбрасывается, если не удалось открыть соединение с базой данных. Оно принимает имя базы данных в качестве параметра.

Класс `ExecException` - это исключение выбрасывается, если выполнение SQL-запроса не удалось. Оно принимает сам запрос в качестве параметра.

3.3.14 Интерфейс для работы с базой данных

Данный класс `ProductDB` является подклассом класса `DBFacade` и представляет собой интерфейс для работы с базой данных продуктов. Основная задача класса - предоставление методов для добавления, удаления и получения информации о пользователях, продуктах и заказах из базы данных.

Метод `is_login_busy(QString login)` проверяет, занят ли указанный логин пользователем.

Метод `add_user(User user)` добавляет нового пользователя в базу данных.

Метод `users()` извлекает и возвращает список пользователей из базы данных.

Метод `remove_user(string login)` удаляет пользователя из базы данных на основе указанного логина.

Метод `get_user(QString login)` получает информацию о пользователе из базы данных на основе логина.

Метод `products()` извлекает и возвращает список продуктов из базы данных.

Метод `add_product(Product product)` добавляет новый продукт в базу данных.

Метод `product_by_article(string article, Product &product)` извлекает информацию о продукте из базы данных на основе артикула.

Метод `product_by_id(int id, Product &product)` извлекает информацию о продукте из базы данных на основе ID.

Метод `remove_product(string article)` удаляет продукт из таблицы на основе заданного артикула.

Метод `add_order(string date, string email, vector<Product> products)` добавляет заказы в корзину.

Метод `orders()` извлекает и возвращает список заказов из базы данных.

Метод `remove_order(string date, string email)` удаляет заказы, которые имеют определенную дату и электронную почту.

Метод `sell_order(string date, string email)` обрабатывает заказы на продажу товаров.

Метод `count(QString from, QString to)` подсчитывает количество товаров, которые были проданы в заданном промежутке времени.

Метод `sum(QString from, QString to)` вычисляет сумму стоимости товаров, проданных в заданном промежутке времени.

Метод `createTables()` создает четыре таблицы в базе данных.

3.3.15 Шаблон для единственного экземпляра

Класс `Singleton` предназначен для создания и использования единственного экземпляра объекта. Основная задача этого класса - гарантировать наличие только одного экземпляра класса и предоставлять глобальную точку доступа к этому экземпляру.

Метод `instance()` является статическим методом и реализует создание и возвращение единственного экземпляра объекта. При первом вызове метода `instance()` экземпляр будет создан, а при последующих вызовах будет возвращаться ссылка на этот уже созданный экземпляр. Единственный экземпляр класса создается только внутри самого класса и не может быть изменен или создан другими классами.

Использование данного класса позволяет создавать единственные объекты, которые могут быть доступны и используемы из любого места программы через вызов статического метода `instance()`.

Общая цель этого шаблона `Singleton` состоит в том, чтобы гарантировать, что у класса есть только один экземпляр, который может быть доступен из любой части программы. Это полезно, когда требуется глобальный доступ к одному и тому же объекту.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка схем алгоритмов

Схема метода on_load () приведена в приложении Г. – метод для загрузки заказов из базы данных и отображения информации о заказах в виде таблицы в пользовательском интерфейсе.

Схема метода sell_order () приведена в приложении В. – метод для обработки заказов на продажу товаров.

4.2 Разработка алгоритмов

4.2.1 Алгоритм загрузки информации о заказах из баз данных и отображение их в таблице

Для алгоритма по шагам рассмотрен метод on_load класса OrderManagement.

Шаг 1. Получение списка заказов из базы данных;

Шаг 2. Очистка таблицы перед загрузкой новых данных;

Шаг 3. Получение текущего индекса строки таблицы;

Шаг 4. Вставка новой строки в таблицу;

Шаг 5. Заполнение ячейки таблицы с датой заказа;

Шаг 6. Заполнение ячейки таблицы с электронной почтой заказчика;

Шаг 7. Создание строки со списком идентификаторов продуктов заказа;

Шаг 8. Содержимое sstr считывается в переменную ids_str с помощью getline().Это позволяет получить строку, содержащую идентификаторы продуктов, разделенные пробелами;

Шаг 9. Заполнение ячейки таблицы со списком идентификаторов продуктов;

Шаг 10. Завершение выполнения метода.

4.2.2 Алгоритм функционала по обработке операций продажи продуктов и взаимодействию с базой данных продуктов

Для алгоритма по шагам рассмотрен метод sell_order класса ProductDB.

Шаг 1. Получение текущей даты в формате ISODate и преобразование ее в строку с форматом "гггг.мм.дд";

Шаг 2. Построение запроса для выборки идентификаторов продуктов из таблицы products_order, где дата равна переданной дате и электронной почте равной переданной;

Шаг 3. Выполнение запроса к базе данных;

Шаг 4. Создание вектора для хранения идентификаторов проданных продуктов;

Шаг 5. Перебор результатов запроса и добавление идентификаторов в вектор;

Шаг 6. Для каждого идентификатора проданного продукта выполнение следующих действий;

Шаг 7. Получение информации о продукте по его идентификатору;

Шаг 8. Построение запроса для добавления информации о проданном продукте в таблицу sold_products;

Шаг 9. Выполнение запроса к базе данных;

Шаг 10. Построение запроса для удаления продукта из таблицы products по его идентификатору;

Шаг 11. Выполнение запроса к базе данных;

Шаг 12. Удаление заказа из таблицы products_order по переданной дате и электронной почте;

Шаг 13. Генерация сигнала об обновлении данных;

Шаг 14. Завершение выполнения метода.

5 РЕЗУЛЬТАТ РАБОТЫ

5.1 Использование приложения

Для запуска программы необходим открыть файлы исходного кода в Qt Creator и собрать проект. Информация по созданию проекта находится в файле product.db. После этого откроется окно программы с каталогом покупателя, где будет отображаться список добавленных товаров (рисунок 5.1).

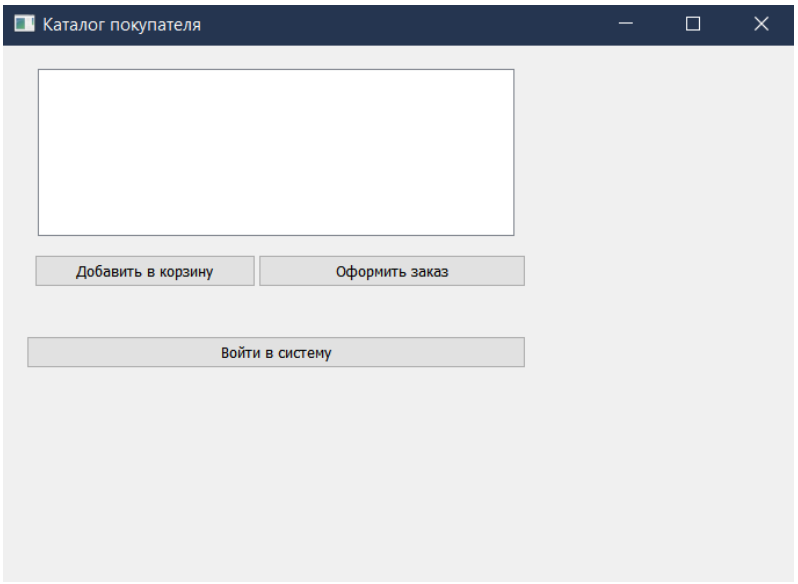

Рисунок 5.1 – Каталог покупателя

При нажатии на кнопку входа откроется окно, где будет запрошен логин и пароль пользователя (система проверяет является ли пользователь администратором или простым покупателем) (рисунок 5.2).

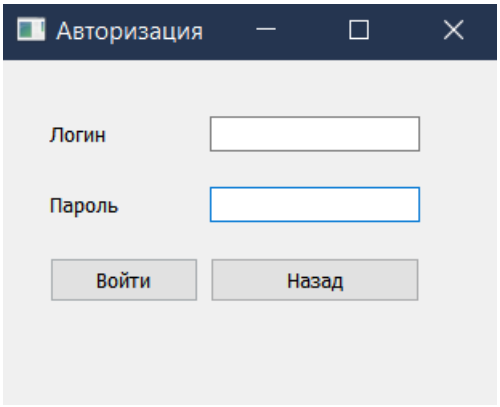

Рисунок 5.2 – Авторизация

После входа администратора открывается каталог менеджера, в котором можно добавлять товары, обрабатывать заказы, удалять/добавлять товары, управлять пользователями и просматривать отчет (рисунок 5.3).

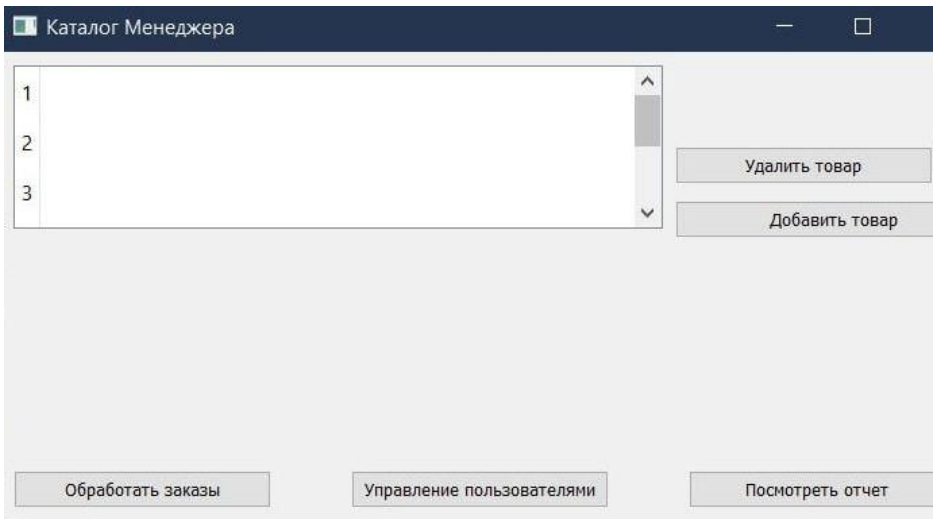

Рисунок 5.3 – Каталог менеджера

При нажатии на кнопку просмотра счёта возникает окно отчета по продажам (рисунок 5.4).

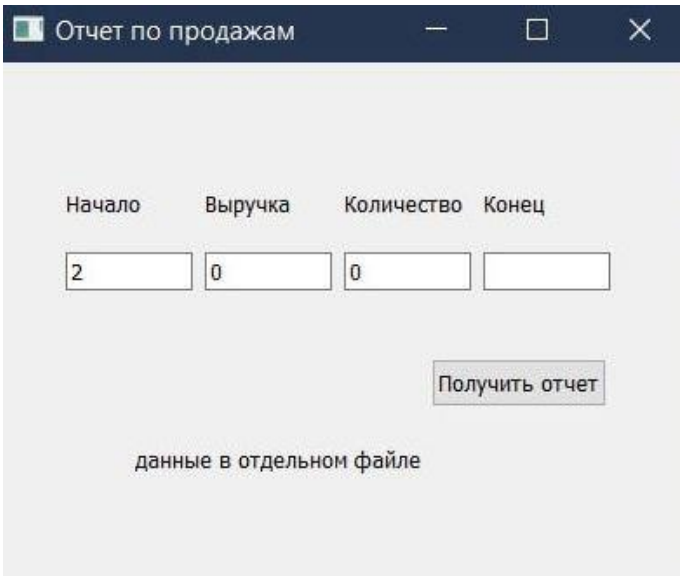

Рисунок 5.4 – Отчет по продажам

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта была изучена и успешно реализована «Система заказов в интернет-магазине» с использованием Qt Creator. Цель заключалась в создании надежной и эффективной системы, способной управлять процессом заказов, обеспечивать удобство покупателей и облегчать работу администраторов.

В процессе работы над проектом было использовано множество возможностей Qt Creator, таких как создание пользовательского интерфейса с помощью визуального редактора, организацию структуры кода, работу с базой данных, обработку событий и многое другое. Благодаря этим инструментам и функциональности Qt Creator, была разработана система заказов, которая соответствует современным требованиям интернет-магазинов.

Данный программный продукт не только обеспечивает функциональность для покупателей, но также помогает администраторам эффективно управлять процессом заказов и обеспечивать высокий уровень обслуживания клиентов.

Важно отметить, что приложение системы заказов в интернет-магазине было разработано с использованием языка программирования C++. Выбор C++ для разработки данного приложения обусловлен его преимуществами, такими как высокая производительность, эффективное использование ресурсов компьютера, богатый набор библиотек и широкие возможности для объектно-ориентированного программирования.

Qt Creator, как интегрированная среда разработки, предоставляет мощные инструменты и библиотеки, которые совместимы с C++ и помогли ускорить процесс разработки приложения. Также была использована функциональность Qt для создания пользовательского интерфейса, обработки событий, работы с базой данных и других задач, связанных с системой заказов в интернет-магазине.

Созданная система заказов представляет собой надежное и гибкое решение для интернет-магазинов, способное удовлетворить потребности как покупателей, так и администраторов.

СПИСОК ЛИТЕРАТУРЫ

1. Метод. Указания по К65 курсовому проектированию для студ. I-40 02 01 “Вычислительные машины, системы и сети” для всех форм обуч. / сост. А. В. Бушкевич, А. М. Ковальчук, И. В. Лукьянова. — Минск: БГУИР, 2009.

2. Учеб. пособие /Ю. А. Луцик , В. Н. Комличенко. – Минск: БГУИР, 2008.

3. QT Documentation – [Электронный ресурс] – Режим доступа: https://doc.qt.io/all-topics.html – Дата обращения: 10.11.2023.

4. Введение в SQL – [Электронный ресурс] – Режим доступа: https://www.academia.edu – Дата обращения: 10.11.2023.

5. C++ GUI Programming with Qt 4, — Jasmin Blanchette, Mark Summerfield, 2015 – Дата обращения: 10.11.2023.

6. The C Programming Language. 2nd Edition, — Dennis Ritchie, Brian Kernighan, 1978.

7. Работа с базами данных в Qt – [Электронный ресурс] – Режим доступа: https://habr.com/ru/articles/51650/ – Дата обращения: 10.11.2023.

8. Уроки Qt5 – [Электронный ресурс] – Режим доступа: https://ravesli.com/uroki-po-qt5/ – Дата обращения: 10.11.2023.

9. Qt для начинающих. Урок 1. Простейшее GUI-приложение и основные виджеты – [Электронный ресурс] – Режим доступа: http://knzsoft.ru/qt-bgr-ls1/ – Дата обращения: 10.11.2023.

ПРИЛОЖЕНИЕ А
Листинг кода

```
//main.cpp

#include "catalogbuyer.h"
#include "catalogmanagement.h"
#include "users.h"
#include <QApplication>

#include "productdb.h"

Q_DECLARE_METATYPE(Product)

int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  qRegisterMetaType<Product>();
  PRODUCT_DB.createTables();
  CURRENT_USER = "";
  CatalogBuyer w;
  w.show();
  return a.exec();
}


//addproduct.h

#ifndef ADDPRODUCT_H
#define ADDPRODUCT_H

#include <QWidget>
#include "product.h"

namespace Ui
{
class Addproduct;
}

class Addproduct : public QWidget
{
  Q_OBJECT

public:
  explicit Addproduct(QWidget *parent = nullptr);
  ~Addproduct();

signals:
  void product(Product product);
public slots:
  void on_add();
```

```
private:
  Ui::Addproduct *ui;
};

#endif // ADDPRODUCT_H


//addproduct.cpp

#include "addproduct.h"
#include "ui_addproduct.h"
#include <QIntValidator>
#include <QRegExpValidator>

Addproduct::Addproduct(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::Addproduct)
{
  ui->setupUi(this);
  ui->cost->setValidator(new QIntValidator(0, 9999999, this));
  ui->kind->setValidator(new QIntValidator(0, 10, this));
  QRegExpValidator*          dateValidator          =          new
QRegExpValidator(QRegExp("20\\d{2}\\.[01]?\\d\\.[0123]?\\d"),
this);
  ui->expiry_date->setValidator(dateValidator);
  ui->receipt_date->setValidator(dateValidator);
  connect(ui->add, SIGNAL(clicked()), SLOT(on_add()));
  connect(ui->back, SIGNAL(clicked()), SLOT(hide()));
}

Addproduct::~Addproduct()
{
  delete ui;
}

void Addproduct::on_add()
{
  auto all_fields = {ui->kind, ui->cost, ui->name, ui->article,
ui->name, ui->expiry_date, ui->receipt_date};
  for (auto field : all_fields)
  {
    if (field->text().isEmpty())
      return;
  }
  Product data;
  data.cost = ui->cost->text().toInt();
  data.kind = ui->kind->text().toInt();
  data.name = ui->name->text().toStdString();
  data.article = ui->article->text().toStdString();
  data.expiry_date = ui->expiry_date->text().toStdString();
  data.receipt_date = ui->receipt_date->text().toStdString();
  emit product(data);
```

```
}


//authwidget.h

#ifndef AUTHWIDGET_H
#define AUTHWIDGET_H

#include <QWidget>
#include "user.h"

namespace Ui
{
    class AuthWidget;
}

class AuthWidget : public QWidget
{
    Q_OBJECT
public:
    explicit AuthWidget(QWidget *parent = nullptr);
    ~AuthWidget();
private slots:
    void enter();
signals:
    void auth_ok(User::Role);
private:
    Ui::AuthWidget *ui;
};

#endif // AUTHWIDGET_H


//authwidget.cpp

#include "authwidget.h"
#include "ui_authwidget.h"
#include "productdb.h"
#include "user.h"

using namespace std;

AuthWidget::AuthWidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::AuthWidget)
{
    ui->setupUi(this);
    connect(ui->enter, SIGNAL(clicked()), SLOT(enter()));
    connect(ui->back, SIGNAL(clicked()), SLOT(close()));
}

AuthWidget::~AuthWidget()
{
```

```
    delete ui;
}

void AuthWidget::enter()
{
  string login = ui->login->text().toStdString();
  string password = ui->password->text().toStdString();
  ui->login->text().clear();
  ui->password->text().clear();
  if  (PRODUCT_DB.is_login_busy(QString::fromStdString(login))  ==
false)
    return;
  User user = PRODUCT_DB.get_user(QString::fromStdString(login));
  if (user.password == password)
  {
    CURRENT_USER = user.login;
    emit auth_ok(user.role);
    this->close();
  }
}

//catalogbuyer.h

#ifndef CATALOG_H
#define CATALOGBUYER_H

#include <QMainWindow>
#include "user.h"
#include <set>
#include "orderbuyer.h"

namespace Ui
{
class CatalogBuyer;
}

class CatalogBuyer : public QMainWindow
{
  Q_OBJECT

public:
  explicit CatalogBuyer(QWidget *parent = 0);
  ~CatalogBuyer();

public slots:
  void on_login();
  void on_auth_ok(User::Role);
  void on_updated();
  void on_add();
  void on_order();

private:
  Ui::CatalogBuyer *ui;
```

```
  OrderBuyer* order_buyer;
};

#endif // CATALOGBUYER_H


//catalogbuyer.cpp

#include "catalogbuyer.h"
#include "ui_catalogbuyer.h"
#include "authwidget.h"
#include "catalogmanagement.h"
#include "productdb.h"

CatalogBuyer::CatalogBuyer(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::CatalogBuyer)
{
  ui->setupUi(this);
  connect(ui->login, SIGNAL(clicked()), this, SLOT(on_login()));
  connect(ui->buy, SIGNAL(clicked()), this, SLOT(on_order()));
  connect(ui->add, SIGNAL(clicked()), this, SLOT(on_add()));
  on_updated();
  connect(&PRODUCT_DB, SIGNAL(updated()), SLOT(on_updated()));
  order_buyer = new OrderBuyer(nullptr);
  order_buyer->hide();
}

CatalogBuyer::~CatalogBuyer()
{
  delete ui;
}

void CatalogBuyer::on_login()
{
  AuthWidget* auth = new AuthWidget(nullptr);
  connect(auth,                        SIGNAL(auth_ok(User::Role)),
SLOT(on_auth_ok(User::Role)));
  auth->show();
}

void CatalogBuyer::on_auth_ok(User::Role role)
{
  this->hide();
  CatalogManagement   *catalog   =   new   CatalogManagement(role,
nullptr);
  catalog->show();
}

void CatalogBuyer::on_updated()
{
  vector<Product> products = PRODUCT_DB.products();
```

```
  ui->table->setRowCount(0);
  for (auto product : products)
  {
    int index = ui->table->rowCount();
    ui->table->insertRow(index);
    ui->table->setItem(index,                0,                new
QTableWidgetItem(QString::fromStdString(product.name)));
    ui->table->setItem(index,                1,                new
QTableWidgetItem(QString::fromStdString(product.article)));
    ui->table->setItem(index,                2,                new
QTableWidgetItem(QString::number(product.cost)));
    ui->table->setItem(index,                3,                new
QTableWidgetItem(QString::number(product.kind)));
    ui->table->setItem(index,                4,                new
QTableWidgetItem(QString::fromStdString(product.expiry_date)));
    ui->table->setItem(index,                5,                new
QTableWidgetItem(QString::fromStdString(product.receipt_date)));
  }
}

void CatalogBuyer::on_add()
{
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
  set<int> selected_rows;
  auto product_articles = order_buyer->current_order();
  for (auto item : selectedItems)
  {
    selected_rows.insert(item->row());
  }
  for (auto row : selected_rows)
  {
    auto article = ui->table->item(row, 1)->text().toStdString();
    product_articles.insert(article);
  }
  order_buyer->load(product_articles);
}

void CatalogBuyer::on_order()
{
  order_buyer->show();
}


//catalogmanagement.h

#ifndef CATALOGMANAGEMENT_H
#define CATALOGMANAGEMENT_H

#include <QWidget>
#include "user.h"
#include "users.h"
#include "addproduct.h"
```

```
#include "ordermanagement.h"
#include "salesreport.h"

namespace Ui
{
class CatalogManagement;
}

class CatalogManagement : public QWidget
{
  Q_OBJECT

public:
  CatalogManagement(User::Role role, QWidget *parent = nullptr);
  ~CatalogManagement();

public slots:
  void on_user_management();
  void on_report();
  void on_product_remove();
  void on_product_add();
  void on_order_management();
  void on_product_added(Product product);
  void on_updated();

private:
  Ui::CatalogManagement *ui;
  Users* users;
  Addproduct* add_product;
  OrderManagement* order_management;
  SalesReport *report;
};

#endif // CATALOGMANAGEMENT_H


//catalogmanagement.cpp

#include "catalogmanagement.h"
#include "ui_catalogmanagement.h"
#include "productdb.h"
#include <set>

CatalogManagement::CatalogManagement(User::Role   role,   QWidget
*parent) :
  QWidget(parent),
  ui(new Ui::CatalogManagement) {
  ui->setupUi(this);

  bool superEnabled = role == User::Role::CompanyManagement;

  ui->users_control->setEnabled(superEnabled);
  ui->report->setEnabled(superEnabled);
```

```
  connect(ui->add, SIGNAL(clicked()), SLOT(on_product_add()));
  connect(ui->remove,                          SIGNAL(clicked()),
SLOT(on_product_remove()));
  connect(ui->users_control,                   SIGNAL(clicked()),
SLOT(on_user_management()));
  connect(ui->report, SIGNAL(clicked()), SLOT(on_report()));
  connect(ui->orders,                          SIGNAL(clicked()),
SLOT(on_order_management()));

  users = new Users(nullptr);
  users->hide();

  add_product = new Addproduct(nullptr);
  add_product->hide();

  order_management = new OrderManagement(nullptr);
  order_management->hide();

  report = new SalesReport(nullptr);
  report->hide();

  connect(add_product,                  SIGNAL(product(Product)),
SLOT(on_product_added(Product)));
  connect(&PRODUCT_DB, SIGNAL(updated()), SLOT(on_updated()));

  on_updated();
}

CatalogManagement::~CatalogManagement() {
  delete ui;
}

void CatalogManagement::on_user_management() {
  users->show();
}

void CatalogManagement::on_report() {
  report->show();
}

void CatalogManagement::on_product_remove() {
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
  set<int> selected_rows;
  set<QString> removed_articles;

  for (auto item : selectedItems) {
    selected_rows.insert(item->row());
  }

  for (auto row : selected_rows) {
    auto article = ui->table->item(row, 1)->text();
```

```
    removed_articles.insert(article);
  }

  for (auto article : removed_articles) {
    PRODUCT_DB.remove_product(article.toStdString());
  }
}

void CatalogManagement::on_product_add() {
  add_product->show();
}

void CatalogManagement::on_order_management() {
  order_management->show();
}

void CatalogManagement::on_product_added(Product product) {
  Product other;
  if  (false  ==  PRODUCT_DB.product_by_article(product.article,
other))
    PRODUCT_DB.add_product(product);
}

void CatalogManagement::on_updated() {
  vector<Product> products = PRODUCT_DB.products();

  ui->table->setRowCount(0);
  for (auto product : products) {
    int index = ui->table->rowCount();
    ui->table->insertRow(index);

    ui->table->setItem(index,               0,               new
QTableWidgetItem(QString::fromStdString(product.name)));
    ui->table->setItem(index,               1,               new
QTableWidgetItem(QString::fromStdString(product.article)));
    ui->table->setItem(index,               2,               new
QTableWidgetItem(QString::number(product.cost)));
    ui->table->setItem(index,               3,               new
QTableWidgetItem(QString::number(product.kind)));
    ui->table->setItem(index,               4,               new
QTableWidgetItem(QString::fromStdString(product.expiry_date)));
    ui->table->setItem(index,               5,               new
QTableWidgetItem(QString::fromStdString(product.receipt_date)));
  }
}


//dbfacade.h

#ifndef DBFACADE_H
#define DBFACADE_H

#include <QObject>
```

```
#include <QtSql/QtSql>
#include <exception>

class DBFacade : public QObject
{
    Q_OBJECT
public:
    explicit DBFacade(QString databasename, QObject *parent = 0);
    ~DBFacade();

signals:
    void updated();

protected:
    void exec(QString);
    QString qs(QString);
    QString qs(std::string);
    QSqlDatabase m_db;
    QSqlQuery *m_query;
};

class OpenDBException: public std::exception
{
public:
    OpenDBException(const char* dbName) : m_dbname(dbName) {}

private:
    virtual const char* what() const throw()
    {
        return m_dbname;
    }
    const char* m_dbname;
};

class ExecException: public std::exception
{
public:
    ExecException(const char* request) : m_request(request) {}

private:
    virtual const char* what() const throw()
    {
        return m_request;
    }
    const char* m_request;
};

#endif // DBFACADE_H


//dbfacade.cpp

#include "dbfacade.h"
```

```
DBFacade::DBFacade(QString databasename, QObject *parent) :
QObject(parent)
{
    m_db = QSqlDatabase::addDatabase("QSQLITE", databasename);
    m_db.setDatabaseName(databasename);
    if (false == m_db.open())
        throw OpenDBException(databasename.toLatin1());
    m_query = new QSqlQuery(m_db);
}

DBFacade::~DBFacade()
{
    delete m_query;
}

QString DBFacade::qs(QString str)
{
    return "'" + str + "'";
}

QString DBFacade::qs(std::string str)
{
    return qs(QString::fromStdString(str));
}

void DBFacade::exec(QString str)
{
    if (false == m_query->exec(str))
        throw ExecException(str.toLatin1());
}


//order.h

#ifndef ORDER_H
#define ORDER_H

#include <string>
#include <vector>

using namespace std;
class Order
{
public:
  string date;
  string email;
  vector<int> product_ids;
};

#endif // ORDER_H
```

```
//orderbuyer.h

#ifndef ORDERBUYER_H
#define ORDERBUYER_H

#include <QWidget>
#include <set>
#include <string>
using namespace std;

namespace Ui
{
class OrderBuyer;
}

class OrderBuyer : public QWidget
{
  Q_OBJECT

public:
  explicit OrderBuyer(QWidget *parent = nullptr);
  ~OrderBuyer();

  void load(set<string> product_articles);
  set<string> current_order();

public slots:
  void create_order();
  void remove_product();

private:
  Ui::OrderBuyer *ui;
  set<string> product_articles;
};

#endif // ORDERBUYER_H


//orderbuyer.cpp

#include "orderbuyer.h"
#include "ui_orderbuyer.h"
#include "product.h"
#include "productdb.h"

OrderBuyer::OrderBuyer(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::OrderBuyer)
{
  ui->setupUi(this);
  connect(ui->back, SIGNAL(clicked()), this, SLOT(hide()));
  connect(ui->ok, SIGNAL(clicked()), this, SLOT(create_order()));
  connect(ui->remove, SIGNAL(clicked()), SLOT(remove_product()));
```

```
}

OrderBuyer::~OrderBuyer()
{
  delete ui;
}

void OrderBuyer::load(set<string> additional_products)
{
  for (auto product : additional_products)
  {
    product_articles.insert(product);
  }
  ui->table->setRowCount(0);
  for (auto article : product_articles)
  {
    Product product;
    PRODUCT_DB.product_by_article(article, product);
    int index = ui->table->rowCount();
    ui->table->insertRow(index);
    ui->table->setItem(index,                0,                new
QTableWidgetItem(QString::fromStdString(product.name)));
    ui->table->setItem(index,                1,                new
QTableWidgetItem(QString::fromStdString(product.article)));
    ui->table->setItem(index,                2,                new
QTableWidgetItem(QString::number(product.cost)));
    ui->table->setItem(index,                3,                new
QTableWidgetItem(QString::number(product.kind)));
    ui->table->setItem(index,                4,                new
QTableWidgetItem(QString::fromStdString(product.expiry_date)));
    ui->table->setItem(index,                5,                new
QTableWidgetItem(QString::fromStdString(product.receipt_date)));
  }
}

set<string> OrderBuyer::current_order()
{
  return product_articles;
}

void OrderBuyer::create_order()
{
  QString email = ui->email->text();
  if  (email.contains('.')  ==  false  ||  email.contains('@')  ==
false)
    return;
  QString                        date_string                        =
QDate::currentDate().toString(Qt::ISODate);
  date_string.truncate(10);
  date_string.replace("-", ".");
  vector<Product> products;
  for (auto article : product_articles)
  {
```

```
    Product product;
    PRODUCT_DB.product_by_article(article, product);
    products.push_back(product);
  }
  PRODUCT_DB.add_order(date_string.toStdString(),
                       email.toStdString(),
                       products);
  hide();
}

void OrderBuyer::remove_product()
{
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
  set<int> selected_rows;
  for (auto item : selectedItems)
  {
    selected_rows.insert(item->row());
  }
  for (auto row : selected_rows)
  {
    auto article = ui->table->item(row, 1)->text().toStdString();
    product_articles.erase(article);
  }
  load(product_articles);
}


//ordermanagement.h

#ifndef ORDERMANAGEMENT_H
#define ORDERMANAGEMENT_H

#include <QWidget>
#include <vector>
#include "order.h"

using namespace std;

namespace Ui
{
class OrderManagement;
}

class OrderManagement : public QWidget
{
  Q_OBJECT

public:
  explicit OrderManagement(QWidget *parent = nullptr);
  ~OrderManagement();

public slots:
```

```
  void on_load();
  void on_remove();
  void on_sell();

  vector<Order> selected_orders();

private:
  Ui::OrderManagement *ui;
};

#endif // ORDERMANAGEMENT_H


//ordermanagement.cpp

#include "ordermanagement.h"
#include "ui_ordermanagement.h"
#include "productdb.h"
#include <sstream>

OrderManagement::OrderManagement(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::OrderManagement)
{
  ui->setupUi(this);
  connect(ui->back, SIGNAL(clicked()), this, SLOT(hide()));
  connect(ui->sell, SIGNAL(clicked()), this, SLOT(on_sell()));
  connect(ui->remove,          SIGNAL(clicked()),          this,
SLOT(on_remove()));
  on_load();
}

OrderManagement::~OrderManagement()
{
  delete ui;
}

void OrderManagement::on_load()
{
  vector<Order> orders = PRODUCT_DB.orders();
  ui->table->setRowCount(0);
  for (auto order : orders)
  {
    int index = ui->table->rowCount();
    ui->table->insertRow(index);
    ui->table->setItem(index,               0,               new
QTableWidgetItem(QString::fromStdString(order.date)));
    ui->table->setItem(index,               1,               new
QTableWidgetItem(QString::fromStdString(order.email)));
    stringstream sstr;
    for (auto id : order.product_ids)
    {
      sstr << id << "  ";
```

```
    }
    string ids_str;
    getline(sstr, ids_str);
    ui->table->setItem(index,                2,                new
QTableWidgetItem(QString::fromStdString(ids_str)));
  }
}

vector<Order> OrderManagement::selected_orders()
{
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
  set<int> selected_rows;
  vector<Order> orders;
  for (auto item : selectedItems)
  {
    selected_rows.insert(item->row());
  }
  for (auto row : selected_rows)
  {
    auto date = ui->table->item(row, 0)->text().toStdString();
    auto email = ui->table->item(row, 1)->text().toStdString();
    auto   it_order   =   find_if(orders.begin(),   orders.end(),
[=](Order order)
    {
      return order.date == date && order.email == email;
    });
    if (it_order == orders.end())
    {
      Order order;
      order.date = date;
      order.email = email;
      orders.push_back(order);
    }
  }
  return orders;
}

void OrderManagement::on_remove()
{
  vector<Order> orders = selected_orders();
  for (auto order : orders)
  {
    PRODUCT_DB.remove_order(order.date, order.email);
  }
  on_load();
}

void OrderManagement::on_sell()
{
  vector<Order> orders = selected_orders();
  for (auto order : orders)
  {
```

```
    PRODUCT_DB.sell_order(order.date, order.email);
  }
  on_load();
}


//product.h

#ifndef PRODUCT_H
#define PRODUCT_H

#include <string>
using namespace std;

class Product
{
public:
  int id;
  string name;
  string article;
  int cost;
  int kind;
  string expiry_date;
  string receipt_date;
};
#endif // PRODUCT_H


//productdb.h

#ifndef PRODUCTDB_H
#define PRODUCTDB_H

#include "dbfacade.h"
#include "singleton.h"
#include <QObject>
#include <QList>
#include "order.h"
#include "product.h"
#include "soltproduct.h"
#include "user.h"
#include <set>

class ProductDB : public DBFacade
{
    Q_OBJECT
public:
    explicit ProductDB(QString dbFIlename = "products.sqlite",
QObject *parent = 0);
    bool is_login_busy(QString login);
    void add_user(User user);
    vector<User> users();
    void remove_user(string login);
```

```
    User get_user(QString login);
    vector<Product> products();
    void add_product(Product product);
    bool product_by_article(string article, Product &product);
    bool product_by_id(int id, Product &product);
    void remove_product(string article);
    void  add_order(string  date,  string  email,  vector<Product>
products);
    vector<Order> orders();
    void remove_order(string date, string email);
    void sell_order(string date, string email);
    int count(QString from, QString to);
    int sum(QString from, QString to);
    void createTables();

public slots:
};

#define PRODUCT_DB Singleton<ProductDB>::instance()

#endif // PRODUCTDB_H


//productdb.cpp

#include "productdb.h"
#include <QMap>

ProductDB::ProductDB(QString dbFIlename, QObject *parent)
    : DBFacade(dbFIlename, parent)
{
}

void ProductDB::createTables()
{
    if (false == m_db.tables().contains("users"))
    {
      exec("CREATE TABLE users"
           "("
              "login TEXT PRIMARY KEY, "
              "password TEXT NOT NULL, "
              "role TEXT"
           ");"
      );
    }

    if (false == m_db.tables().contains("products"))
    {
      exec("CREATE TABLE products"
           "("
              "id INTEGER PRIMARY KEY AUTOINCREMENT, "
              "name TEXT NUT NULL, "
              "article TEXT NUT NULL, "
```

```
                "cost INTEGER NUT NULL, "
                "kind INTEGER NUT NULL, "
                "expiry_date TEXT NOT NULL, "
                "receipt_date TEXT NOT NULL "
            ");"
        );
    }

    if (false == m_db.tables().contains("solt_products"))
    {
        exec("CREATE TABLE solt_products"
             "("
                "id INTEGER PRIMARY KEY, "
                "name TEXT NUT NULL, "
                "article TEXT NUT NULL, "
                "cost INTEGER NUT NULL, "
                "kind INTEGER NUT NULL, "
                "expiry_date TEXT NOT NULL, "
                "receipt_date TEXT NOT NULL, "
                "realization_date TEXT NOT NULL"
            ");"
        );
    }

    if (false == m_db.tables().contains("products_order"))
    {
        exec("CREATE TABLE products_order"
             "("
                "id_product INTEGER NOT NULL,"
                "date TEXT NOT NULL,"
                "email TEXT NOT NULL"
            ");"
        );
    }
}

bool ProductDB::is_login_busy(QString login)
{
  QString query = tr("SELECT login FROM users WHERE login = ") +
qs(login);
  exec(query);
  return m_query->first();
}

void ProductDB::add_user(User user)
{
  QString query = tr("INSERT INTO users(login, password, role)
VALUES (")
                        +  qs(QString::fromStdString(user.login))
+ ","
                        +
qs(QString::fromStdString(user.password)) + ","
```

```
                          +
qs(QString::fromStdString(user.role_to_string(user.role))) + ")";
  exec(query);
  emit updated();
}

vector<User> ProductDB::users()
{
  vector<User> users;
  QString query = tr("SELECT login, password, role FROM users ");
  exec(query);
  while (true == m_query->next())
  {
    QString login = m_query->value(0).toString();
    QString password = m_query->value(1).toString();
    QString role = m_query->value(2).toString();
    users.push_back(User(login.toStdString(),
                         password.toStdString(),

User::string_to_role(role.toStdString())));
  }
  return users;
}

User ProductDB::get_user(QString login)
{
  QString query = tr("SELECT password, role FROM users WHERE login
= ") + qs(login);
  exec(query);
  if (false == m_query->next())
  {
    throw string("wrong login");
  }
  QString password = m_query->value(0).toString();
  QString role = m_query->value(1).toString();
  return    User(login.toStdString(),    password.toStdString(),
User::string_to_role(role.toStdString()));
}

void ProductDB::remove_user(string login)
{
  QString query = tr("DELETE FROM users WHERE login = ") +
qs(login);
  exec(query);
  emit updated();
}

void ProductDB::add_product(Product product)
{
  QString query = tr("INSERT INTO products(name, article, cost,
kind, expiry_date, receipt_date) VALUES (")
                         + qs(product.name) + ","
                         + qs(product.article) + ","
```

```
                        + QString::number(product.cost) + ","
                        + QString::number(product.kind) + ","
                        + qs(product.expiry_date) + ","
                        + qs(product.receipt_date) + ")";
  exec(query);
  emit updated();
}

vector<Product> ProductDB::products()
{
  vector<Product> products;
  QString query = tr("SELECT id, name, article, cost, kind,
expiry_date, receipt_date FROM products ");
  exec(query);
  while (true == m_query->next())
  {
    Product data;
    data.id = m_query->value(0).toInt();
    data.name = m_query->value(1).toString().toStdString();
    data.article = m_query->value(2).toString().toStdString();
    data.cost = m_query->value(3).toInt();
    data.kind = m_query->value(4).toInt();
    data.expiry_date                    =                  m_query-
>value(5).toString().toStdString();
    data.receipt_date                    =                  m_query-
>value(6).toString().toStdString();

    products.push_back(data);
  }
  return products;
}

bool   ProductDB::product_by_article(string   article,   Product&
product)
{
  QString   query   =   tr("SELECT   name,   article,   cost,   kind,
expiry_date, receipt_date, id "
                     "FROM   products   WHERE   article   =   ")   +
qs(article);
  exec(query);
  if (false == m_query->next())
  {
    return false;
  }
  product.name = m_query->value(0).toString().toStdString();
  product.article = m_query->value(1).toString().toStdString();
  product.cost = m_query->value(2).toInt();
  product.kind = m_query->value(3).toInt();
  product.expiry_date                    =                  m_query-
>value(4).toString().toStdString();
  product.receipt_date                    =                  m_query-
>value(5).toString().toStdString();
  product.id = m_query->value(6).toInt();
```

```
  return true;
}

bool ProductDB::product_by_id(int id, Product& product)
{
  QString query = tr("SELECT name, article, cost, kind,
expiry_date, receipt_date, id "
                     "FROM products WHERE id = ") +
QString::number(id);
  exec(query);
  if (false == m_query->next())
  {
    return false;
  }
  product.name = m_query->value(0).toString().toStdString();
  product.article = m_query->value(1).toString().toStdString();
  product.cost = m_query->value(2).toInt();
  product.kind = m_query->value(3).toInt();
  product.expiry_date = m_query-
>value(4).toString().toStdString();
  product.receipt_date = m_query-
>value(5).toString().toStdString();
  product.id = m_query->value(6).toInt();
  return true;
}

void ProductDB::remove_product(string article)
{
  QString query = tr("DELETE FROM products WHERE article = ") +
qs(article);
  exec(query);
  emit updated();
}

void ProductDB::add_order(string date, string email,
vector<Product> products)
{
  for (auto product : products)
  {
    QString query = tr("INSERT INTO products_order(date, email,
id_product) VALUES (")
                       + qs(date) + ","
                       + qs(email) + ","
                       + QString::number(product.id) + ")";
    exec(query);
  }
}

vector<Order> ProductDB::orders()
{
  vector<Order> orders;
  QString query = tr("SELECT date, email, id_product FROM
products_order ");
```

```
  exec(query);
  while (true == m_query->next())
  {
    string date = m_query->value(0).toString().toStdString();
    string email = m_query->value(1).toString().toStdString();
    int id = m_query->value(2).toInt();
    auto   it_order   =   find_if(orders.begin(),   orders.end(),
[=](Order order){
      return order.date == date && order.email == email;
    });
    if (it_order != orders.end())
    {
      (*it_order).product_ids.push_back(id);
    }
    else
    {
      Order order;
      order.date = date;
      order.email = email;
      order.product_ids.push_back(id);
      orders.push_back(order);
    }
  }
  return orders;
}

void ProductDB::remove_order(string date, string email)
{
  QString query = tr("DELETE FROM products_order WHERE ") +
                      " date = " + qs(date) +
                      "AND email = " + qs(email);
  exec(query);
  emit updated();
}

void ProductDB::sell_order(string date, string email)
{
  QString                          date_string                          =
QDate::currentDate().toString(Qt::ISODate);
  date_string.truncate(10);
  date_string.replace("-", ".");
  QString query = tr("SELECT id_product FROM products_order WHERE
") +
                      " date = " + qs(date) +
                      "AND email = " + qs(email);
  exec(query);
  vector<int> sold_ids;
  while (true == m_query->next())
  {
    int id = m_query->value(0).toInt();
    sold_ids.push_back(id);
  }
  for (auto id : sold_ids)
```

```
  {
    Product product;
    product_by_id(id, product);
    query = tr("INSERT INTO solt_products")
         + tr(" (id, name, article, cost, kind, expiry_date,
receipt_date, realization_date)")
         + tr(" VALUES (")
         + QString::number(product.id) + ","
         + qs(product.name) + ","
         + qs(product.article) + ","
         + QString::number(product.cost) + ","
         + QString::number(product.kind) + ","
         + qs(product.expiry_date) + ","
         + qs(product.receipt_date) + ","
         + qs(date_string) + ")";
    exec(query);
    QString query = tr("DELETE FROM products WHERE id = ") +
QString::number(id);
    exec(query);
  }
  remove_order(date, email);
  emit updated();
}

int ProductDB::count(QString from, QString to)
{
    exec(
          tr("SELECT COUNT(*) FROM solt_products WHERE ") +
          tr("realization_date >= ") + qs(from) +
          tr(" AND realization_date <= ") + qs(to));
    m_query->next();
    return m_query->value(0).toInt();
}

int ProductDB::sum(QString from, QString to)
{
    exec(
          tr("SELECT SUM(cost) FROM solt_products WHERE ") +
          tr("realization_date >= ") + qs(from) +
          tr(" AND realization_date <= ") + qs(to));
    m_query->next();
    return m_query->value(0).toInt();
}


//salesreport.h

#ifndef SALESREPORT_H
#define SALESREPORT_H

#include <QWidget>

namespace Ui
```

```
{
class SalesReport;
}

class SalesReport : public QWidget
{
  Q_OBJECT

public:
  explicit SalesReport(QWidget *parent = nullptr);
  ~SalesReport();

public slots:
  void on_calc();

private:
  Ui::SalesReport *ui;
};

#endif // SALESREPORT_H


//salesreport.cpp

#include "salesreport.h"
#include "ui_salesreport.h"
#include "productdb.h"

SalesReport::SalesReport(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::SalesReport)
{
  ui->setupUi(this);
  connect(ui->calc, SIGNAL(clicked()), SLOT(on_calc()));
  QRegExpValidator*          dateValidator          =          new
QRegExpValidator(QRegExp("20\\d{2}\\.[01]?\\d\\.[0123]?\\d"),
this);
  ui->start->setValidator(dateValidator);
  ui->end->setValidator(dateValidator);
}

SalesReport::~SalesReport()
{
  delete ui;
}

void SalesReport::on_calc()
{
  QString from = ui->start->text();
  QString to = ui->end->text();
  int count = PRODUCT_DB.count(from, to);
  int sum = PRODUCT_DB.sum(from, to);
  ui->count->setText(QString::number(count));
```

```
  ui->sum->setText(QString::number(sum));
}


//singleton.h

#ifndef SINGLETON_H
#define SINGLETON_H

#include <QObject>

template <class T>
class Singleton
{
public:
  static T& instance()
  {
    static T instance;
    return instance;
  }

private:
  Singleton();
  ~Singleton();
  Singleton(const Singleton &);
  Singleton& operator=(const Singleton &);
};

#endif


//soltproduct.h

#ifndef SOLTPRODUCT_H
#define SOLTPRODUCT_H

#include "product.h"

class SoltProduct : public Product
{
public:
  string  realization_date;
};

#endif // SOLTPRODUCT_H
//user.h

#ifndef USER_H
#define USER_H

#include <string>

using namespace std;
```

```
class User {
public:
  enum Role {
    ProductManagement, CompanyManagement
  };

  User(string _login, string _password, Role _role);

  static string role_to_string(Role role);
  static Role string_to_role(string);

  string login, password;
  Role role;
};

#define CURRENT_USER Singleton<string>::instance()

#endif // USER_H


//user.cpp

#include "user.h"

User::User(string _login, string _password, Role _role)
  : login(_login), password(_password), role(_role) {
}


string User::role_to_string(Role role) {
  if (role == ProductManagement) {
    return "Product";
  }
  return "Company";
}

User::Role User::string_to_role(string str) {
  if (str == "Product")
    return Role::ProductManagement;
  if (str == "Company")
    return Role::CompanyManagement;
  throw string("wrong role");
}

//users.h

#ifndef USERS_H
#define USERS_H

#include <QWidget>

namespace Ui
```

```
{
  class Users;
}

class Users : public QWidget
{
  Q_OBJECT

public:
  explicit Users(QWidget *parent = nullptr);
  ~Users();

public slots:
  void on_remove();a
  void on_add();
  void on_updated();

private:
  Ui::Users *ui;
};

#endif // USERS_H


//users.cpp

#include "users.h"
#include "productdb.h"
#include "ui_users.h"
#include "user.h"
#include <set>
using namespace std;

Users::Users(QWidget *parent) :
  QWidget(parent),
  ui(new Ui::Users)
{
  ui->setupUi(this);
  on_updated();
  connect(ui->add, SIGNAL(clicked()), SLOT(on_add()));
  connect(ui->remove, SIGNAL(clicked()), SLOT(on_remove()));
  connect(&PRODUCT_DB, SIGNAL(updated()), SLOT(on_updated()));
}

Users::~Users()
{
  delete ui;
}

void Users::on_remove()
{
  QList<QTableWidgetItem*>     selectedItems     =     ui->table-
>selectedItems();
```

```
  set<int> selected_rows;
  set<QString> removed_users;
  for (auto item : selectedItems)
  {
    selected_rows.insert(item->row());
  }
  for (auto row : selected_rows)
  {
    auto login = ui->table->item(row, 0)->text();
    removed_users.insert(login);
  }
  for (auto login : removed_users)
  {
    User user = PRODUCT_DB.get_user(login);
    if (user.login == CURRENT_USER)
    {
      continue;
    }
    PRODUCT_DB.remove_user(user.login);
  }
}

void Users::on_add()
{
  QString login = ui->login->text();
  QString password = ui->password->text();
  if (login.isEmpty() || password.isEmpty())
    return;
  if (PRODUCT_DB.is_login_busy(login))
    return;
  bool isSuper = ui->isSuper->isChecked();
  User::Role role;
  if (false == isSuper)
  {
    role = User::Role::ProductManagement;
  }
  else
  {
    role = User::Role::CompanyManagement;
  }
  PRODUCT_DB.add_user(User(login.toStdString(),
password.toStdString(), role));
}

void Users::on_updated()
{
  vector<User> users = PRODUCT_DB.users();
  ui->table->setRowCount(0);
  for (auto user : users)
  {
    int index = ui->table->rowCount();
    ui->table->insertRow(index);
```

```
    ui->table->setItem(index,                0,                new
QTableWidgetItem(QString::fromStdString(user.login)));
    ui->table->setItem(index,                1,                new
QTableWidgetItem(QString::fromStdString(User::role_to_string(use
r.role))));
  }
}
```

ПРИЛОЖЕНИЕ Б

Диаграмма классов

ПРИЛОЖЕНИЕ В

Схема метода on_load()

ПРИЛОЖЕНИЕ Г

Схема метода sell_order()

ПРИЛОЖЕНИЕ Д

Ведомость документов