

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ НИЖЕГОРОДСКОЙ ОБЛАСТИ

Государственное бюджетное профессиональное образовательное учреждение

НИЖЕГОРОДСКИЙ РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Специальность 09.02.07 Информационные системы и программирование

Квалификация Программист

КУРСОВАЯ РАБОТА

МДК 11.01 Технология разработки и защиты баз данных

Тема: «Проектирование и разработка базы данных для магазина автозапчастей»

Выполнил студент группы

ЗИСиП-21-1

Кузенкова Анна Павловна

Проверил преподаватель
Гутянская Е.М.

Проект защищен с оценкой

Дата защиты _____

Подпись _____

Нижний Новгород, 2023

Оглавление

Введение	2
1. Теоретические основы разрабатываемой темы	4
1.1 Анализ проектируемой системы.....	4
2. Проектирование базы данных информационной системы	7
2.1 Информационно-логическая модель базы данных.....	7
2.2 Словарь данных.....	7
2.3 Ограничения ссылочной целостности.....	12
2.4 Обоснование выбора СУБД.....	14
2.5 Триггеры и хранимые процедуры.....	15
3. Разработка информационной системы.....	20
3.1 Описание структуры системы	20
3.2 Разработка класса подключения к базе данных.....	23
3.3 Разработка функционала главного окна	41
3.4 Разработка функционала авторизации пользователя.....	43
3.5 Разработка капчи	47
3.6 Разработка функционала регистрации пользователя.....	51
3.7 Разработка раздела «Аккаунт»	53
3.8 Разработка функционала к аккаунту сотрудника	74
3.9 Разработка функционала к аккаунту клиента	86
4. Руководство пользователя	91
4.1 Страница главного окна	91
4.2 Окна авторизации, регистрации, капчи	91
4.3 Главная страница сотрудника	92
4.4 Главная страница клиента	92
Заключение	93
Список литературы.....	94
Приложения	95

Введение

В настоящее время автозапчасти являются неотъемлемой частью жизни автовладельцев и автомобильных сервисов. Постоянно растущий спрос на автозапчасти ставит перед магазинами автозапчастей задачу эффективного управления складскими операциями, учетом товаров и обслуживанием клиентов. Для достижения этих целей важным элементом является наличие и правильное функционирование базы данных, специально разработанной для магазина автозапчастей.

Целью данной курсовой работы является разработка базы данных и информационной системы для магазина автозапчастей. Это позволит оптимизировать управление складскими операциями, автоматизировать процессы учета товаров и обслуживания клиентов, улучшить эффективность работы и повысить удовлетворенность клиентов.

Задачи работы:

1. Анализ требований и потребностей магазина автозапчастей.
2. Проектирование структуры базы данных с учетом особенностей магазина автозапчастей.
3. Разработка функциональной части информационной системы для автоматизации бизнес-процессов.
4. Реализация и интеграция базы данных и информационной системы.
5. Анализ эффективности использования базы данных и информационной системы в работе магазина автозапчастей.

Ожидаемый результат работы:

Готовая функционирующая база данных и информационная система для магазина автозапчастей.

В процессе выполнения данной курсовой работы будут применены современные методы и инструменты проектирования баз данных и

информационных систем, что позволит создать удобную, надежную и эффективную систему управления информацией для магазина автозапчастей. Исследование и анализ лучших практик в области проектирования и разработки баз данных и информационных систем для магазинов автозапчастей позволят учесть все необходимые функции и требования, обеспечить оптимальное использование ресурсов и повысить конкурентоспособность магазина автозапчастей на рынке.

1. Теоретические основы разрабатываемой темы

1.1 Анализ проектируемой системы

В рамках разработанной системы для магазина автозапчастей будут реализованы следующие функции:

1. Многопользовательский доступ
2. Разграничение функционала по ролям
3. Клиент и сотрудник магазина автозапчастей могут осуществлять следующие действия:
 - a. получать оперативную информацию о наличии, описании и стоимости автозапчасти;
 - b. просматривать несколько фото выбранной автозапчасти;
 - c. осуществлять подбор запчастей по критериям;
 - d. устанавливать фильтр на производителя автозапчастей;
 - e. выбирать пункт доставки из имеющихся. (только для сотрудника, так как именно он будет оформлять заказ)
4. Сотрудник магазина может осуществлять следующие действия:
 - a. вести справочники (добавление, удаление, редактирование);
 - b. оформлять заказ на автозапчасти (в одном заказе может быть несколько различных автозапчастей в разном количестве);
 - c. стоимость заказа рассчитывается динамически.

Данные в системе будут храниться в структурированном формате в базе данных:

- Пункт выдачи. В этой таблице хранится информация обо всех пунктах выдачи, доступных клиентам.
- Персонал. В этой таблице хранится информация о сотрудниках магазина, включая их контактную информацию и данные для авторизации.

- Роль. В этой таблице хранится информация о ролях.
- Категория. В этой таблице хранится информация о категории того или иного товара.
- Товар. В этой таблице хранится информация обо всех товарах, доступных для продажи, включая описание, цену, кол-во на складе и статус.
- Дополнительные фото товара. В этой таблице хранится информация о дополнительных фотографиях товаров, доступных для продажи.
- Заказы. В этой таблице хранится информация о заказах, включая цену и дату доставки.
- Модель автомобиля. В этой таблице хранится информация о моделях автомобилей.
- Производитель. В этой таблице хранится информация о производителях автозапчастей, продаваемых в магазине.
- Клиенты. В этой таблице хранится информация о клиентах магазина, включая их контактную информацию.
- Отзывы. В этой таблице хранится информация об отзывах, полученных от клиентов, включая их оценки и комментарии.

Рассмотрим функции, которые будут реализованы в нашем приложении:

1. Многопользовательский доступ

Многопользовательский доступ — это возможность одновременного доступа нескольких пользователей к одному приложению. Для этого используются пул соединения — это метод оптимизации работы с базой данных, при котором определенное количество соединений к базе данных создается заранее и затем используется многократно различными запросами от разных пользователей. Это позволяет избежать множества подключений и отключений к базе данных, что улучшает производительность и экономит ресурсы.

2. Разграничение функционала по ролям

Разграничение функционала по ролям — это метод управления доступом, при котором права доступа к различным функциям системы предоставляются на основе ролей, которые назначаются пользователям. Каждая роль определяет свой набор прав доступа. Этот подход позволяет эффективно управлять правами доступа и обеспечивает безопасность информации в системе.

В нашей программе сотрудник магазина будет иметь доступ ко всем функциям и данным, в то время как клиенты будут иметь ограниченный доступ к определенным функциям или данным.

Функции, доступные для сотрудника магазина:

- Получение информации о наличии, описании и стоимости автозапчасти;
- Просматривать и добавлять несколько фото выбранной автозапчасти;
- Осуществлять поиск и сортировку продуктов;
- Вести справочники (добавление, удаление, редактирование);
- Оформлять заказ на автозапчасти (в одном заказе может быть несколько различных автозапчастей в разном количестве);

Функции, доступные для клиента:

- Получение информации о наличии, описании и стоимости автозапчасти;
- Просматривать несколько фото выбранной автозапчасти;
- Осуществлять поиск и сортировку продуктов;

2. Проектирование базы данных информационной системы

2.1 Информационно-логическая модель базы данных

Информационно-логическая модель отображает данные предметной области в виде совокупности информационных объектов и связей между ними. Эта модель представляет данные, подлежащие хранению в базе данных.

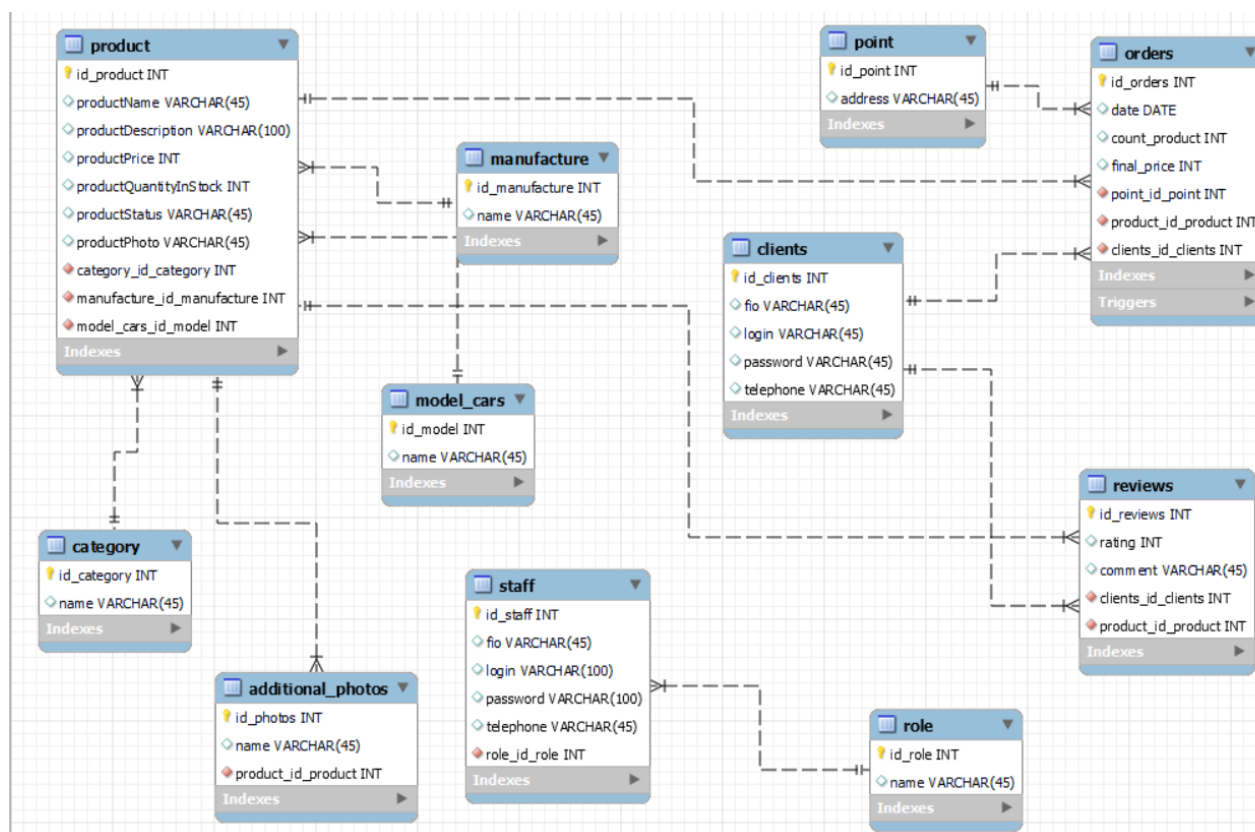


Рисунок 1. Информационно-логическая модель базы данных

2.2 Словарь данных

В приведённых ниже таблицах описаны поля всех таблиц базы данных, используемых в разработанной информационной системе.

additional_photos			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_photos	Идентификатор фото продукта	INT	PK

name	Наименование фото	VARCHAR (45)	
product_id_product	Идентификатор товара	INT	FK (product.id_ product)

point			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_point	Идентификатор пункта выдачи	INT	PK
address	Адрес	VARCHAR (45)	

role			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_role	Идентификатор роли	INT	PK
name	Наименование роли	VARCHAR (45)	

staff			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_staff	Идентификатор персонала	INT	PK
fio	ФИО персонала	VARCHAR (45)	
login	Логин	VARCHAR (100)	
password	Пароль	VARCHAR (100)	
telephone	Номер телефона	VARCHAR (45)	

role_id_role	Идентификатор роли	INT	FK (role.id_role)
--------------	-----------------------	-----	-------------------

category			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_category	Идентификатор категории	INT	PK
name	Наименование категории	VARCHAR (45)	

model_cars			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_model	Идентификатор модели автомобиля	INT	PK
name	Наименование модели автомобиля	VARCHAR (45)	

product			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_product	Идентификатор товара	INT	PK
productName	Наименование продукта	VARCHAR (45)	

productDescription	Описание	VARCHAR (100)	
productPrice	Цена	INT	
productQuantityInStock	Кол-во на складе	INT	
productStatus	Статус	VARCHAR (45)	
productPhoto	Фото продукта	VARCHAR (45)	
category_id_category	Идентификатор категории	INT	FK (category.id_cat egory)
manufacture_id_manufa cture	Идентификатор производителя	INT	FK (manufacture.id_ manufacture)
model_cars_id_model	Идентификатор модели автомобиля	INT	FK (model_cars.id_m odel)

orders			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_orders	Идентификатор заказа	INT	PK
date	Дата заказа	DATE	
count_product	Кол-во товара	INT	
final_price	Конечная цена	INT	
point_id_point	Идентификатор пункта выдачи	INT	FK (point.id_ point)

product_id_product	Идентификатор товара	INT	FK (product.id_ product)
client_id_client	Идентификатор клиента	INT	FK (client.id_ client)

manufacture			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_manufacture	Идентификатор производителя	INT	PK
name	Наименование производителя	VARCHAR (45)	

reviews			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id_reviews	Идентификатор отзывов	INT	PK
rating	Оценка	INT	
comment	Комментарии	VARCHAR (45)	
client_id_client	Идентификатор клиента	INT	FK (client.id_ client)
product_id_product	Идентификатор товара	INT	FK (product.id_ product)

clients			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)

id_clients	Идентификатор персонала	INT	PK
fio	ФИО персонала	VARCHAR (45)	
login	Логин	VARCHAR (45)	
password	Пароль	VARCHAR (45)	
telephone	Номер телефона	VARCHAR (45)	

2.3 Ограничения ссылочной целостности

1. Ограничение на таблице «staff»:

Идентификатор роли (role_id_role) является внешним ключом, связанным с таблицей «role» по полю идентификатора роли (id_role). Это ограничение гарантирует, что каждый сотрудник будет иметь действительную роль, представленную в таблице «role». Такая связь помогает контролировать доступ и права сотрудников.

2. Ограничение на таблице «product»:

Идентификатор категории (category_id_category) является внешним ключом, связанным с таблицей «category» по полю идентификатора роли (id_category). Это ограничение гарантирует, что каждый товар ссылается на свою категорию в таблице «category». Такая связь позволяет отслеживать к какой категории относится тот или иной товар.

Идентификатор производителя (manufacture_id_manufacture) является внешним ключом, связанным с таблицей «manufacture» по полю идентификатора роли (id_manufacture). Это ограничение гарантирует, что каждый товар ссылается своего производителя в таблице «manufacture». Такая связь позволяет отслеживать к какому производителю относится тот или иной товар.

Идентификатор модели автомобиля (model_cars_id_model) является внешним ключом, связанным с таблицей «model_cars» по полю идентификатора роли (id_model). Это ограничение гарантирует, что каждый товар ссылается на

свою модель автомобиля в таблице «model_cars». Такая связь позволяет отслеживать к какой модели автомобиля относится тот или иной товар.

3. Ограничение на таблице «reviews»:

Идентификатор отзывов (clients_id_clients) является внешним ключом, связанным с таблицей «clients» по полю идентификатора роли (id_clients). Это ограничение гарантирует, что каждый отзыв будет принадлежать действительному клиенту, представленного в таблице «clients». Такая связь позволяет отслеживать от какого клиента поступил тот или иной отзыв.

Идентификатор продукта (product_id_product) является внешним ключом, связанным с таблицей «product» по полю идентификатора роли (id_product). Это ограничение гарантирует, что каждый отзыв будет принадлежать действительному товару, представленному в таблице «product». Такая связь позволяет отслеживать к какому товару поступил тот или иной отзыв.

4. Ограничение на таблице «orders»:

Идентификатор пункта выдачи (point_id_point) является внешним ключом, связанным с таблицей «point» по полю идентификатора роли (id_point). Это ограничение гарантирует, что каждый заказ имеет свой пункт выдачи, представленный в таблице «point». Такая связь позволяет отслеживать заказы, которые выдаются в том или ином пункте выдачи.

Идентификатор клиента (clients_id_clients) является внешним ключом, связанным с таблицей «clients» по полю идентификатора роли (id_clients). Это ограничение гарантирует, что каждый заказ имеет своего действительного клиента, представленный в таблице «clients». Такая связь позволяет отслеживать заказы, относящиеся к тому или иному клиенту.

Идентификатор продукта (product_id_product) является внешним ключом, связанным с таблицей «product» по полю идентификатора роли (id_product). Это ограничение гарантирует, что в состав каждого заказа будут входить

действительные товары, представленные в таблице «product». Такая связь позволяет отслеживать состав того или иного заказа.

5. Ограничение на таблице «additional_photos»:

Идентификатор продукта (product_id_product) является внешним ключом, связанным с таблицей «product» по полю идентификатора роли (id_product). Это ограничение гарантирует, что каждому продукту приходится действительные фото, представленные в таблице «additional_photos».

2.4 Обоснование выбора СУБД

Система управления базами данных (СУБД) – это набор программ, которые управляют структурой БД и контролируют доступ к данным, хранящимся в БД. СУБД служит посредником между пользователем и БД.

В качестве СУБД для моей информационной системы я выбрала MySQL. Так как она имеет ряд преимуществ по сравнению с другими СУБД:

1. Простота использования: MySQL предлагает интуитивно понятный и простой язык запросов SQL, что делает его легким в освоении даже для новичков.

2. Высокая производительность: MySQL обладает эффективным механизмом хранения и обработки данных, обеспечивая высокую производительность при выполнении запросов и обработке больших объемов данных.

3. Надежность и стабильность: MySQL известен своей стабильностью и надежностью, обеспечивая минимальное время простоя и высокую доступность данных.

4. Безопасность: MySQL предлагает многоуровневую систему безопасности, включая функции шифрования, аутентификации и контроля доступа, что обеспечивает защиту данных от несанкционированного доступа и повреждения.

5. Хорошая поддержка сообщества: MySQL имеет широкое и активное сообщество разработчиков и пользователей, что обеспечивает доступ к обширной документации, руководствам и онлайн-ресурсам, а также возможность общаться и получать поддержку от опытных пользователей.

6. Частые обновления и поддержка: MySQL активно разрабатывается и поддерживается командой разработчиков Oracle, что гарантирует появление новых функций, исправлений ошибок и обновлений безопасности.

2.5 Триггеры и хранимые процедуры

Триггеры и хранимые процедуры для базы данных магазина автозапчастей могут быть созданы, чтобы автоматизировать определенные операции или проверки при выполнении изменений в базе данных.

Процедуры к базе данных:

Процедуры для добавления клиента проверяют, существует ли клиент с таким же логином. Если существует, то выводит сообщение об ошибке, иначе добавляет нового клиента. (проверяются две таблицы: clients и staff, чтобы исключить того, чтобы клиент смог зарегистрироваться под логином сотрудника)

SQL код:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `AddClient`(  
    IN new_fio VARCHAR(45),  
    IN new_login VARCHAR(45),  
    IN new_password VARCHAR(45),  
    IN new_telephone VARCHAR(45)  
)  
BEGIN  
    DECLARE existing_count INT;  
    -- Проверяем существующую запись с таким же логином  
    SELECT COUNT(*)  
    INTO existing_count  
    FROM clients
```



```

WHERE login = new_login;

-- Если клиент с таким логином уже существует, выводим сообщение об
ошибке

IF existing_count > 0 THEN

    SELECT 'Ошибка: клиент с таким логином уже существует' as Success;

ELSE

    -- Иначе добавляем нового клиента

    INSERT INTO clients (fio, login, password, telephone) VALUES
(new_fio, new_login, new_password, new_telephone);

    SELECT 'Новый клиент успешно добавлен' as Success;

END IF;

END

```

SQL код:

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `AddStaff` (

    IN new_fio VARCHAR(45),

    IN new_login VARCHAR(45),

    IN new_password VARCHAR(45),

    IN new_telephone VARCHAR(45)

)

BEGIN

    DECLARE existing_count INT;

    -- Проверяем существующую запись с таким же логином

    SELECT COUNT(*)

    INTO existing_count

    FROM staff

    WHERE login = new_login;

    -- Если клиент с таким логином уже существует, выводим сообщение об
ошибке

    IF existing_count > 0 THEN

```

```

        SELECT 'Ошибка: клиент с таким логином уже существует' as Success;
ELSE
    -- Иначе добавляем нового клиента
    INSERT INTO clients (fio, login, password, telephone) VALUES
(new_fio, new_login, new_password, new_telephone);
    SELECT 'Новый клиент успешно добавлен' as Success;
END IF;
END

```

Триггер для таблицы «orders»:

Триггер будет активироваться после каждой вставки новой записи в таблицу «orders». Он будет обновлять количество товара на складе – поле «productQuantityInStock» из таблицы product, вычитая количество, указанное во вставленной записи – то, что мы впишем в поле count_product.

SQL код:

```

CREATE DEFINER=`root`@`localhost` TRIGGER `updateProductQuantity2`
AFTER INSERT ON `orders` FOR EACH ROW BEGIN
    DECLARE newQuantity INT;
    SELECT productQuantityInStock - NEW.count_product INTO newQuantity
    FROM product
    WHERE id_product = NEW.product_id_product;
    IF newQuantity < 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Ошибка:
Недостаточно количества на складе';
    ELSE
        UPDATE product
        SET productQuantityInStock = newQuantity
        WHERE id_product = NEW.product_id_product;
    END IF;

```

Триггеры для таблицы «product»:

Триггер будет проверять, есть ли товар в заказах перед удалением. Если такой товар найден в заказах, триггер вызовет ошибку и предотвратит удаление продукта.

SQL код:

DELIMITER //

CREATE TRIGGER prevent_product_deletion

BEFORE DELETE ON product

FOR EACH ROW

BEGIN

DECLARE products_count INT;

SELECT COUNT(*) INTO products_count FROM orders WHERE
product_id_product = OLD.id_product;

IF products_count > 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Этот товар
нельзя удалить, пока он есть в заказе';

END IF;

END; //

Триггер будет менять поле статус «productStatus» на «Присутствует», если количество товара на складе больше нуля, иначе поле «productStatus» будет хранить значение «Отсутствует».

SQL код:

DELIMITER //

CREATE TRIGGER UpdateProductStatus

BEFORE UPDATE ON product

FOR EACH ROW

BEGIN

IF NEW.productQuantityInStock > 0 THEN

SET NEW.productStatus = 'Присутствует';

```
ELSE  
  SET NEW.productStatus = 'Отсутствует';  
END IF;  
END; //
```

3. Разработка информационной системы

3.1 Описание структуры системы

Приложение состоит из 10 окон, взаимосвязанных между собой: «Главное окно», «Авторизация», «Капча», «Регистрация», «Профиль», «Добавление товара», «Редактирование товара», «Добавление дополнительных фото», «Просмотр дополнительных фото», «Оформление заказа», «Оставить отзыв».

Переходы между страницами осуществляются при помощи кнопок и контекстного меню.

На странице главного меню находится название магазина, логотип и кнопка авторизации.

На странице авторизации пользователь должен ввести логин и пароль. При необходимости, есть возможность увидеть пароль, а также зарегистрироваться, если аккаунта ещё нет. При нажатии кнопки войти, когда поля ещё не заполнены, выведется предупреждение «Заполните все поля». Если поля заполнены, но таких данных нет в базе данных, то на экране появится ошибка «Произошла ошибка при входе в личный кабинет» и появится капча.

Капча является модальным окном, оно блокирует работу пользователя с приложением. Если ввести неправильно символы капчи, то выведется предупреждение «Проверка не пройдена. Попробуйте ещё раз.» и новая капча. При верном введении символов, появится информация «Проверка пройдена» и доступ к окну авторизации.

Если у пользователя ещё нет аккаунта, то его можно создать, нажав на кнопку зарегистрироваться. При нажатии, появляется окно, где нужно вписать необходимые данные: ФИО, логин, пароль, телефон. При нажатии кнопки зарегистрироваться, когда не были заполнены все данные, выведется предупреждение «Заполните все поля». При нажатии кнопки, когда все поля заполнены, выведется информация, что новый клиент успешно зарегистрирован, через 2 секунды окно с регистрацией закроется и пользователю вновь предоставится окно авторизации. Если в базе данных уже существует

пользователь с логином, который ввёл незарегистрированный клиент, то выведется предупреждение «Клиент с таким логином уже существует».

После успешной авторизации через окно «Авторизация» пользователю выведется информация, что авторизация прошла успешно и откроется окно «Профиль». В верхнем правом углу будет информация о роли пользователя и его ФИО.

Если пользователь вошёл под ролью «Клиент», то ему будет представлена информация о существующих для продажи продуктов и информации о них, а именно: наименование товара, его описание, производитель, категория, наличие на складе и цена. Клиент может найти нужный товар, используя поиск. Так же есть возможность отсортировать список доступных для покупки продуктов по марке автомобиля, по производителю, по категории и по статусу. Для того, чтобы отсортировать товары по марке автомобиля, необходимо нажать на значок того или иного автомобиля. Сортировать можно отдельно по поставщикам, выбрав нужного из выпадающего списка, товары автоматически отсортируются. Так же, можно произвести сортировку отдельно по категории или по статусу, или же по обоим критериям, нажав на значок поиска. Чтобы вернуть список товаров к начальному виду, можно нажать на значок «по умолчанию».

Выбрав товар и нажав по нему правой кнопкой мыши, можно выбрать «Просмотр дополнительных фото». При нажатии открывается окно, где показаны дополнительные фото, выбранного товара. Пролистать фотографии можно с помощью кнопок.

Для клиента доступна опция «Оставить отзыв». Нажав на соответствующую кнопку, открывается окно, где необходимо выбрать продукт, поставить оценку от 1 до 5 включительно и, по желанию написать комментарий. Если клиент вводит оценку отличную от доступного интервала, выводится предупреждение «Оцените нас от 1 до 5». Без выбора продукта и оценки отзыв не отправится и выведется предупреждение «Заполните все данные», однако, если продукт выбран и поле «оценка» заполнена корректно, то отправить отзыв

можно и без заполнения строки «Комментарий». Когда отзыв успешно отправлен, на экран выводится благодарность клиенту.

Чтобы вернуться к окну авторизации, клиенту нужно нажать на значок дома.

Если пользователь вошёл под ролью «Администратор» или «Сотрудник», пользователь имеет те же функции, что и «Клиент» с некоторыми добавлениями.

У сотрудника магазина есть доступ к добавлению товара, его редактированию, удалению, добавлению или просмотру дополнительных фото, а также доступ к оформлению заказа.

Выбрав товар и нажав по нему правой кнопкой мыши, сотруднику предоставляется выбор.

При выборе «Редактировать», открывается окно, где сотрудник может внести правки для конкретного товара. Если нажать кнопку «Редактировать», когда не все поля заполнены, то выведется ошибка «Заполните все поля или проверьте корректность данных.», также эта ошибка появляется, когда сотрудник пытается ввести отрицательное значение или ноль для поля «цена», а также, если введено отрицательное значение для поля «количество на складе».

При выборе «Удалить», происходит удаление выбранного товара. Для того, чтобы изменения пришли в силу необходимо нажать на кнопку «Обновить».

При выборе «Добавить дополнительные фото», открывается окно, где сотрудник может добавить фотографию или фотографии выбранного товара. Если сотрудник при нажатии на кнопку «Добавить» не заполнил хотя бы одно поле, то выведется соответствующая ошибка. Если заполнено хотя бы одно поле, то фотография или фотографии будут успешно добавлены.

Так же сотрудник при нажатии на кнопку «Добавить новый товар» получает доступ к окну добавления. Необходимо заполнить все поля корректными значениями. Допускается оставить пустым поле «Фото», тогда у продукта будет фото-заглушка «Фото временно отсутствует». Если нажать кнопку «Добавить», когда не все поля заполнены, то выведется предупреждение «Заполните все поля или проверьте корректность данных.», также это

предупреждение появляется, когда сотрудник пытается ввести отрицательное значение или нуль для поля «цена», а также, если введено отрицательное значение для поля «количество на складе». Для того, чтобы изменения пришли в силу необходимо нажать на кнопку «Обновить».

При нажатии на кнопку «Оформить заказ», сотрудник получает доступ к окну добавления заказа, где необходимо указать клиента, продукт, его количество, пункт выдачи и нажать на кнопку «Добавить в заказ». Цена каждого продукта и конечная цена рассчитывается динамически. Пока в заказ не добавлен хотя бы один товар, кнопка «Оформить» будет недоступной. Если заказ не удовлетворяет сотрудника, он может его очистить кнопкой «Очистить». Если ввести в поле «Количество товара» нуль или отрицательное значение, то выведется предупреждение «Заполните данные корректно». Не заполнив все необходимые поля, то также выведется предупреждение. При верном заполнении и нажав на кнопку «Оформить», заказ успешно добавляется и выводится соответствующее сообщение.

3.2 Разработка класса подключения к базе данных

Листинг класса Main (главный класс, с которого начинается работа программы)

```
package com.example.coursework;
```

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;
```

```
import javax.swing.*;
import java.io.IOException;
public class Main extends Application {
```

```
    @Override
```

```
    public void start(Stage stage) throws IOException {
```

```
        FXMLLoader fxmlLoader = new
```

```
FXMLLoader(com.example.coursework.Main.class.getResource("homePage.fxml"));
```

```
        Scene scene = new Scene(fxmlLoader.load(), 764, 573);
```



```

        stage.setScene(scene);
        stage.show();
        stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
    }

    public static void main(String[] args) {
        launch();
    }
}

```

Класс Database реализует подключение к базе данных, благодаря чему, основная программа может обращаться к подключению и отсылать запросы к базе данных.

Листинг класса Database.

```

package com.example.coursework;

import java.sql.*;
import java.util.ArrayList;

public class Database {

    private final String host = "127.0.0.1";
    private final String port = "3306";
    private final String dbName = "coursework";
    private final String login = "root";
    private final String password = "vanessa2020k";

    private static int max, id;

    private Connection getDbConnection() throws ClassNotFoundException,
SQLException {
        String connStr = "jdbc:mysql://" + host + ":" + port + "/" + dbName +
"?characterEncoding=UTF8";
        Class.forName("com.mysql.cj.jdbc.Driver");

        return DriverManager.getConnection(connStr, login, password);
    }

    public ArrayList<String> getPrice(String name, String count) throws
SQLException, ClassNotFoundException {

        String sql = "SELECT productPrice * " + count + " as productPrice FROM

```

```
product where productName="" + name + " ";
```

```
Statement statement = getDbConnection().createStatement();  
ResultSet resultSet = statement.executeQuery(sql);
```

```
ArrayList<String> productPrice = new ArrayList<>();  
while (resultSet.next())  
    productPrice.add(resultSet.getString("productPrice"));  
return productPrice;  
}
```

```
public int getPriceInt(String name) throws SQLException,  
ClassNotFoundException {  
    String sql = "SELECT productPrice FROM product where productName="" +  
name + """;  
    PreparedStatement statement = getDbConnection().prepareStatement(sql);  
    ResultSet resultSet = statement.executeQuery();  
    int count = 0;  
    while (resultSet.next()) {  
        count = resultSet.getInt(1);  
    }  
    return count;  
}
```

```
public int getStaff(String log, String pas) throws SQLException,  
ClassNotFoundException {  
    String sql = "SELECT count(*) as n FROM staff where login=? and  
password=?";  
    PreparedStatement statement = getDbConnection().prepareStatement(sql);  
    statement.setString(1, log);  
    statement.setString(2, pas);  
    ResultSet resultSet = statement.executeQuery();  
    int count = 0;  
    while (resultSet.next()) {  
        count = resultSet.getInt(1);  
    }  
    return count;  
}
```

```
public int getClients(String log, String pas) throws SQLException,  
ClassNotFoundException {  
    String sql = "SELECT count(*) as n FROM clients where login=? and  
password=?";  
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
```

```

statement.setString(1, log);
statement.setString(2, pas);
ResultSet resultSet = statement.executeQuery();
int count = 0;
while (resultSet.next()) {
    count = resultSet.getInt(1);
}
return count;
}

```

```

public String getRole(String log) throws SQLException, ClassNotFoundException
{
    String sql = "SELECT name FROM role join staff on
role.id_role=staff.role_id_role where login='" + log + "' ";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet resultSet = statement.executeQuery();
    String role = "";
    while (resultSet.next()) {
        role = resultSet.getString("name");
    }
    return role;
}

```

```

public String getFioClient(String log) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT fio FROM clients where login='" + log + "' ";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet resultSet = statement.executeQuery();
    String name = "";
    while (resultSet.next()) {
        name = resultSet.getString("fio");
    }
    return name;
}

```

```

public String getFioStaff(String log) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT fio FROM staff where login='" + log + "' ";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet resultSet = statement.executeQuery();
    String name = "";
    while (resultSet.next()) {
        name = resultSet.getString("fio");
    }
}

```

```

    return name;
}

```

```

public String clientAdd(String fio, String log, String pas, String tel) throws
SQLException, ClassNotFoundException {

```

```

    String vivod = "";
    CallableStatement proc = getDbConnection().prepareCall("CALL AddClient(?,
?, ?, ?)");
    proc.setString(1, fio);
    proc.setString(2, log);
    proc.setString(3, pas);
    proc.setString(4, tel);

    ResultSet res =proc.executeQuery();
    while (res.next()){
        vivod = res.getString("Success");
    }
    return vivod;
}

```

```

public ArrayList<ProductData> getProduct() throws SQLException,
ClassNotFoundException {
    String sql = "SELECT * FROM product ORDER BY `id_product`";

```

```

    Statement statement = getDbConnection().createStatement();
    ResultSet res = statement.executeQuery(sql);

```

```

    ArrayList<ProductData> product = new ArrayList<>();
    while (res.next())
        product.add(new ProductData(res.getString("productName"),
res.getString("productDescription"), res.getInt("productPrice"),
res.getInt("productQuantityInStock"), res.getString("productStatus"),
res.getString("productPhoto"), res.getInt("category_id_category"),
res.getInt("manufacture_id_manufacture"), res.getInt("model_cars_id_model")));
    return product;
}

```

```

public ArrayList getProductSearch(String name) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT * FROM product where productName like ?";

```

```

    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    statement.setString(1, "%" + name + "%");

```

```
ResultSet res = statement.executeQuery();
```

```
ArrayList<ProductData> product = new ArrayList<>();
```

```
while (res.next())  
    product.add(new ProductData(res.getString("productName"),  
res.getString("productDescription"), res.getInt("productPrice"),  
res.getInt("productQuantityInStock"), res.getString("productStatus"),  
res.getString("productPhoto"), res.getInt("category_id_category"),  
res.getInt("manufacture_id_manufacture"), res.getInt("model_cars_id_model")));  
    return product;  
}
```

```
public ArrayList<ProductData> getProductSortMan(Integer man) throws  
SQLException, ClassNotFoundException {  
    String sql = "SELECT * FROM product where manufacture_id_manufacture ="  
+ man + "";  
    ;
```

```
Statement statement = getDbConnection().createStatement();  
ResultSet res = statement.executeQuery(sql);
```

```
ArrayList<ProductData> product = new ArrayList<>();  
while (res.next())  
    product.add(new ProductData(res.getString("productName"),  
res.getString("productDescription"), res.getInt("productPrice"),  
res.getInt("productQuantityInStock"), res.getString("productStatus"),  
res.getString("productPhoto"), res.getInt("category_id_category"),  
res.getInt("manufacture_id_manufacture"), res.getInt("model_cars_id_model")));  
    return product;  
}
```

```
public ArrayList<ProductData> getProductModel(Integer model) throws  
SQLException, ClassNotFoundException {  
    String sql = "SELECT * FROM product where model_cars_id_model =" +  
model + "";
```

```
Statement statement = getDbConnection().createStatement();  
ResultSet res = statement.executeQuery(sql);
```

```
ArrayList<ProductData> product = new ArrayList<>();  
while (res.next())  
    product.add(new ProductData(res.getString("productName"),  
res.getString("productDescription"), res.getInt("productPrice"),
```

```

res.getInt("productQuantityInStock"), res.getString("productStatus"),
res.getString("productPhoto"), res.getInt("category_id_category"),
res.getInt("manufacture_id_manufacture"), res.getInt("model_cars_id_model"))));
    return product;
}

```

```

public ArrayList<String> getCategoryMain() throws SQLException,
ClassNotFoundException {
    String sql = "SELECT * FROM category ORDER BY `id_category`";

    Statement statement = getDbConnection().createStatement();
    ResultSet resultSet = statement.executeQuery(sql);

    ArrayList<String> address = new ArrayList<>();
    while (resultSet.next())
        address.add(resultSet.getString("name"));

    return address;
}

```

```

public ArrayList<String> getModelMain() throws SQLException,
ClassNotFoundException {
    String sql = "SELECT * FROM model_cars ORDER BY `id_model`";

    Statement statement = getDbConnection().createStatement();
    ResultSet resultSet = statement.executeQuery(sql);

    ArrayList<String> address = new ArrayList<>();
    while (resultSet.next())
        address.add(resultSet.getString("name"));

    return address;
}

```

```

public ArrayList<String> getManufactureMain() throws SQLException,
ClassNotFoundException {
    String sql = "SELECT * FROM manufacture ORDER BY `id_manufacture`";

    Statement statement = getDbConnection().createStatement();
    ResultSet resultSet = statement.executeQuery(sql);

    ArrayList<String> address = new ArrayList<>();
    while (resultSet.next())
        address.add(resultSet.getString("name"));
}

```

```
    return address;
}
```

```
public ArrayList<String> getPoint() throws SQLException,
ClassNotFoundException {
    String sql = "SELECT * FROM point ORDER BY `id_point`";

    Statement statement = getDbConnection().createStatement();
    ResultSet resultSet = statement.executeQuery(sql);

    ArrayList<String> address = new ArrayList<>();
    while (resultSet.next())
        address.add(resultSet.getString("address"));

    return address;
}
```

```
public ArrayList<String> getClient() throws SQLException,
ClassNotFoundException {
    String sql = "SELECT * FROM clients ORDER BY `id_clients`";

    Statement statement = getDbConnection().createStatement();
    ResultSet resultSet = statement.executeQuery(sql);

    ArrayList<String> address = new ArrayList<>();
    while (resultSet.next())
        address.add(resultSet.getString("fio"));

    return address;
}
```

```
public ArrayList<String> getProductMain() throws SQLException,
ClassNotFoundException {
    String sql = "SELECT * FROM product ORDER BY `id_product`";

    Statement statement = getDbConnection().createStatement();
    ResultSet resultSet = statement.executeQuery(sql);

    ArrayList<String> address = new ArrayList<>();
    while (resultSet.next())
        address.add(resultSet.getString("productName"));

    return address;
}
```

```
}
```

```
public ArrayList<Integer> getCategory(String category) throws SQLException,  
ClassNotFoundException {
```

```
    String sql = "select id_product FROM product join category on  
product.category_id_category=category.id_category where category.name = ?";  
    PreparedStatement statement = getDbConnection().prepareStatement(sql);  
    statement.setString(1, category);
```

```
    ResultSet res = statement.executeQuery();  
    ArrayList<Integer> stud = new ArrayList<>();  
    while (res.next())  
        stud.add(res.getInt("id_product"));  
    return stud;
```

```
}
```

```
public ArrayList<Integer> getStatus(String status) throws SQLException,  
ClassNotFoundException {
```

```
    String sql = "select id_product FROM product where productStatus = ?";  
    PreparedStatement statement = getDbConnection().prepareStatement(sql);  
    statement.setString(1, status);
```

```
    ResultSet res = statement.executeQuery();  
    ArrayList<Integer> stud = new ArrayList<>();  
    while (res.next())  
        stud.add(res.getInt("id_product"));  
    return stud;
```

```
}
```

```
public ArrayList<Integer> getProductStatNo() throws SQLException,  
ClassNotFoundException {
```

```
    String sql = "SELECT id_product FROM product where productStatus =  
'Отсутствует' ";  
    PreparedStatement statement = getDbConnection().prepareStatement(sql);  
    ResultSet res = statement.executeQuery();  
    ArrayList<Integer> stud = new ArrayList<>();  
    while (res.next())  
        stud.add(res.getInt("id_product"));  
    return stud;
```

```
}
```

```
public ArrayList<Integer> getProductStat() throws SQLException,  
ClassNotFoundException {
```

```
    String sql = "SELECT id_product FROM product where productStatus =
```



```

'Присутствует' ";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    ArrayList<Integer> stud = new ArrayList<>();
    while (res.next())
        stud.add(res.getInt("id_product"));
    return stud;
}

public ArrayList<ProductData> getProductWithId(int n) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT * FROM product where id_product = ?";

    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    statement.setInt(1, n);

    ResultSet res = statement.executeQuery();
    ArrayList<ProductData> product = new ArrayList<>();

    while (res.next())
        product.add(new ProductData(res.getString("productName"),
res.getString("productDescription"), res.getInt("productPrice"),
res.getInt("productQuantityInStock"), res.getString("productStatus"),
res.getString("productPhoto"), res.getInt("category_id_category"),
res.getInt("manufacture_id_manufacture"), res.getInt("model_cars_id_model")));
    return product;
}

public String getManufactureString(Integer id) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT name FROM manufacture where id_manufacture=" + id
+ """;
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet resultSet = statement.executeQuery();
    String name = "";
    while (resultSet.next()) {
        name = resultSet.getString("name");
    }
    return name;
}

public String getCategoryString(Integer id) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT name FROM category where id_category=" + id + """;

```

```

PreparedStatement statement = getDbConnection().prepareStatement(sql);
ResultSet resultSet = statement.executeQuery();
String name = "";
while (resultSet.next()) {
    name = resultSet.getString("name");
}
return name;
}

```

```

public Integer getIdProduct(String name) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT id_product FROM product where productName =? ";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    statement.setString(1, name);
    ResultSet res = statement.executeQuery();
    id = 0;
    while (res.next()) {
        id = res.getInt(1);
    }
    return id;
}

```

```

public int getOneProductPrice(String id) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT productPrice FROM product where id_product=?";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    statement.setString(1, id);
    ResultSet res = statement.executeQuery();

    int tasks = 0;
    while (res.next()) {
        tasks = res.getInt("productPrice");
    }
    return tasks;
}

```

```

public String getOneProductPhoto(String id) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT productPhoto FROM product where id_product=?";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    statement.setString(1, id);
    ResultSet res = statement.executeQuery();

    String tasks = "";

```

```

while (res.next()) {
    tasks = res.getString("productPhoto");
}
return tasks;
}

```

```

public String getOneProductDescription(String id) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT productDescription FROM product where id_product=?";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    statement.setString(1, id);
    ResultSet res = statement.executeQuery();

    String tasks = "";
    while (res.next()) {
        tasks = res.getString("productDescription");
    }
    return tasks;
}

```

```

public String getOneProductName(String id) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT productName FROM product where id_product=?";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    statement.setString(1, id);
    ResultSet res = statement.executeQuery();
    String tasks = "";
    while (res.next()) {
        tasks = res.getString("productName");
    }
    return tasks;
}

```

```

public int getOneProductQuantityInStock(String id) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT productQuantityInStock FROM product where
id_product=?";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    statement.setString(1, id);
    ResultSet res = statement.executeQuery();

    int tasks = 0;
    while (res.next()) {
        tasks = res.getInt("productQuantityInStock");
    }
}

```

```

    }
    return tasks;
}

```

public void insertProduct(String name, String desc, Integer price, Integer stock, String stat, String photo, Integer cat, Integer man, Integer mod) throws

```

SQLException, ClassNotFoundException {
    String sql = "INSERT INTO product VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
    PreparedStatement prSt = getDbConnection().prepareStatement(sql);
    prSt.setInt(1, maxProduct());
    prSt.setString(2, name);
    prSt.setString(3, desc);
    prSt.setInt(4, price);
    prSt.setInt(5, stock);
    prSt.setString(6, stat);
    prSt.setString(7, photo);
    prSt.setInt(8, cat);
    prSt.setInt(9, man);
    prSt.setInt(10, mod);

    prSt.executeUpdate();
}

```

```

public Integer maxProduct() throws SQLException, ClassNotFoundException {
    String sql = "SELECT max(id_product) as max FROM product";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    max = 0;
    while (res.next()) {
        max = res.getInt(1);
    }
    max += 1;
    return max;
}

```

```

public Integer maxOrder() throws SQLException, ClassNotFoundException {
    String sql = "SELECT max(id_orders) as max FROM orders";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    max = 0;
    while (res.next()) {
        max = res.getInt(1);
    }
    max += 1;
}

```

```

    return max;
}

```

```

public Integer getCategoryForInsert(String name) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT id_category FROM category where name = '" + name +
""";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    int id_cat = 0;
    while (res.next()) {
        id_cat = res.getInt(1);
    }
    return id_cat;
}

```

```

public Integer getManufactureForInsert(String name) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT id_manufacture FROM manufacture where name = '" +
name + """;
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    int id_cat = 0;
    while (res.next()) {
        id_cat = res.getInt(1);
    }
    return id_cat;
}

```

```

public Integer getModelForInsert(String name) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT id_model FROM model_cars where name = '" + name +
""";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    int id_cat = 0;
    while (res.next()) {
        id_cat = res.getInt(1);
    }
    return id_cat;
}

```

```

public void insertOrder(Integer count, Integer finalPrice, Integer point, Integer
product, Integer client) throws SQLException, ClassNotFoundException {

```

```

String sql = "INSERT INTO orders VALUES (?, current_date, ?, ?, ?, ?, ?)";
PreparedStatement prSt = getDbConnection().prepareStatement(sql);
prSt.setInt(1, maxOrder());
prSt.setInt(2, count);
prSt.setInt(3, finalPrice);
prSt.setInt(4, point);
prSt.setInt(5, product);
prSt.setInt(6, client);

prSt.executeUpdate();
}

```

```

public Integer getPointForInsert(String address) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT id_point FROM point where address = '" + address + "'";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    int id_cat = 0;
    while (res.next()) {
        id_cat = res.getInt(1);
    }
    return id_cat;
}

```

```

public Integer getProductForInsert(String name) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT id_product FROM product where productName = '" +
name + "'";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    int id_cat = 0;
    while (res.next()) {
        id_cat = res.getInt(1);
    }
    return id_cat;
}

```

```

public Integer getClientForInsert(String fio) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT id_clients FROM clients where fio = '" + fio + "'";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    int id_cat = 0;
    while (res.next()) {

```

```

        id_cat = res.getInt(1);
    }
    return id_cat;
}

```

```

    public void deleteProduct(Integer id) throws SQLException,
ClassNotFoundException {
        String sql = "delete from product where id_product = ?";
        PreparedStatement prSt = getDbConnection().prepareStatement(sql);
        prSt.setInt(1, id);

        prSt.executeUpdate();
    }
    public void deleteAdditionalPhotot(Integer id) throws SQLException,
ClassNotFoundException {
        String sql = "delete from additional_photos where product_id_product = ?";
        PreparedStatement prSt = getDbConnection().prepareStatement(sql);
        prSt.setInt(1, id);

        prSt.executeUpdate();
    }

```

```

    public void updateProduct(String name, String desc, Integer price, Integer stock,
String stat, String photo, Integer cat, Integer man, Integer mod, Integer idProd)
throws SQLException, ClassNotFoundException {
        String sql = "update product set productName=?, productDescription=?,
productPrice=?, productQuantityInStock=?, productStatus=?, productPhoto=?,
category_id_category=?, manufacture_id_manufacture=?, model_cars_id_model=?
where id_product=?";
        PreparedStatement prSt = getDbConnection().prepareStatement(sql);
        prSt.setString(1, name);
        prSt.setString(2, desc);
        prSt.setInt(3, price);
        prSt.setInt(4, stock);
        prSt.setString(5, stat);
        prSt.setString(6, photo);
        prSt.setInt(7, cat);
        prSt.setInt(8, man);
        prSt.setInt(9, mod);
        prSt.setInt(10, idProd);

        prSt.executeUpdate();
    }

```

```

public Integer maxReviews() throws SQLException, ClassNotFoundException {
    String sql = "SELECT max(id_reviews) as max FROM reviews";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    max = 0;
    while (res.next()) {
        max = res.getInt(1);
    }
    max += 1;
    return max;
}

```

```

public void insertReviews(String rating, String comment, Integer client, Integer
product) throws SQLException, ClassNotFoundException {
    String sql = "INSERT INTO reviews VALUES (?, ?, ?, ?, ?)";
    PreparedStatement prSt = getDbConnection().prepareStatement(sql);
    prSt.setInt(1, maxReviews());
    prSt.setString(2, rating);
    prSt.setString(3, comment);
    prSt.setInt(4, client);
    prSt.setInt(5, product);

    prSt.executeUpdate();
}

```

```

public Integer getClientForReview(String log) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT id_clients FROM clients where login = '" + log + "'";
    PreparedStatement statement = getDbConnection().prepareStatement(sql);
    ResultSet res = statement.executeQuery();
    int id_client = 0;
    while (res.next()) {
        id_client = res.getInt(1);
    }
    return id_client;
}

```

```

public ArrayList<String> getPhoto(Integer id) throws SQLException,
ClassNotFoundException {
    String sql = "SELECT name FROM additional_photos WHERE
product_id_product='" + id + "' ";

    Statement statement = getDbConnection().createStatement();
    ResultSet resultSet = statement.executeQuery(sql);
}

```



```

        ArrayList<String> photo = new ArrayList<>();
        while (resultSet.next())
            photo.add(resultSet.getString("name"));
        return photo;
    }

```

```

    public String getProductName(String id) throws SQLException,
        ClassNotFoundException {
        String sql = "SELECT productName FROM product where id_product=?";
        PreparedStatement statement = getDbConnection().prepareStatement(sql);
        statement.setString(1, id);
        ResultSet res = statement.executeQuery();
        String tasks = "";
        while (res.next()) {
            tasks = res.getString("productName");
        }
        return tasks;
    }

```

```

    public Integer maxAdditionalPhotos() throws SQLException,
        ClassNotFoundException {
        String sql = "SELECT max(id_photos) as max FROM additional_photos";
        PreparedStatement statement = getDbConnection().prepareStatement(sql);
        ResultSet res = statement.executeQuery();
        max = 0;
        while (res.next()) {
            max = res.getInt(1);
        }
        max += 1;
        return max;
    }

```

```

    public void insertAdditionalPhotos(String name, Integer product) throws
        SQLException, ClassNotFoundException {
        String sql = "INSERT INTO additional_photos VALUES (?, ?, ?)";
        PreparedStatement prSt = getDbConnection().prepareStatement(sql);
        prSt.setInt(1, maxAdditionalPhotos());
        prSt.setString(2, name);
        prSt.setInt(3, product);

        prSt.executeUpdate();
    }

```

```

    public ArrayList<ProductData> getCategoryForSorting(Integer idCat) throws
SQLException, ClassNotFoundException {
        String sql = "SELECT * FROM product where category_id_category = ?";

        PreparedStatement statement = getDbConnection().prepareStatement(sql);
        statement.setInt(1, idCat);

        ResultSet res = statement.executeQuery();
        ArrayList<ProductData> product = new ArrayList<>();

        while (res.next())
            product.add(new ProductData(res.getString("productName"),
res.getString("productDescription"), res.getInt("productPrice"),
res.getInt("productQuantityInStock"), res.getString("productStatus"),
res.getString("productPhoto"), res.getInt("category_id_category"),
res.getInt("manufacture_id_manufacture"), res.getInt("model_cars_id_model")));
        return product;
    }
    public ArrayList<ProductData> getStatusForSorting(String status) throws
SQLException, ClassNotFoundException {
        String sql = "SELECT * FROM product where productStatus = ?";

        PreparedStatement statement = getDbConnection().prepareStatement(sql);
        statement.setString(1, status);

        ResultSet res = statement.executeQuery();
        ArrayList<ProductData> product = new ArrayList<>();

        while (res.next())
            product.add(new ProductData(res.getString("productName"),
res.getString("productDescription"), res.getInt("productPrice"),
res.getInt("productQuantityInStock"), res.getString("productStatus"),
res.getString("productPhoto"), res.getInt("category_id_category"),
res.getInt("manufacture_id_manufacture"), res.getInt("model_cars_id_model")));
        return product;
    }
}

```

3.3 Разработка функционала главного окна

Данный функционал разработан с целью ознакомления пользователя с домашней страницы приложения. (Рисунок 2)

Листинг главного класса приложения, с которого начинается «жизнь»

приложения — HomePage:

```
package com.example.coursework;
```

```
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;
```

```
public class HomePage {
```

```
    @FXML
```

```
    private Button buttonAuthorization;
```

```
    @FXML
```

```
    public void close() {
```

```
        Stage stage = (Stage) buttonAuthorization.getScene().getWindow();
```

```
        stage.close();
```

```
    }
```

```
    @FXML
```

```
    void initialize() {
```

```
        buttonAuthorization.addEventHandler(MouseEvent.MOUSE_CLICKED, new
        EventHandler<MouseEvent>() {
```

```
            @Override
```

```
            public void handle(MouseEvent mouseEvent) {
```

```
                try {
```

```
                    FXMLLoader fxmLoader = new
                    FXMLLoader(Main.class.getResource("authorization.fxml"));
```

```
                    Scene scene = new Scene(fxmLoader.load(), 409, 494);
```

```
                    Stage stage = new Stage();
```

```
                    stage.setScene(scene);
```

```
                    stage.show();
```

```
                    stage.getIcons().add(new
                    Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
```

```
                    close();
```

```
                } catch (Exception e) {
```

```

        e.printStackTrace();
    }
}
});
}
}

```

3.4 Разработка функционала авторизации пользователя

Данный функционал разработан с целью обеспечения разграничения доступа клиентов и сотрудников к функционалу приложения. Методика заключается в сравнении данных, указанных пользователем на форме авторизации со значениями, хранящимися в базе и, в случае их совпадения, предоставление пользователю доступа к его личному кабинету. В случае, если пользователь ввёл некорректные данные для входа — ему высветиться соответствующее окно об ошибке и капча. (Рисунок 3-4)

Класс Authorization отвечает за функционал окна авторизации.

Листинг класса Authorization.

```

package com.example.coursework;

import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.input.MouseEvent;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

import java.util.Objects;

public class Authorization {
    @FXML
    private Button buttonEnter, buttonRegistration, buttonSee;

```

```
@FXML
private TextField textLogin, textPasswordSee;
```

```
@FXML
private PasswordField textPassword;
```

```
int k;
String fio, role;
static int id_client;
```

```
Database database = new Database();
```

```
public static void showAlertError(String title, String content) {
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(content);
    Stage stage = (Stage) alert.getDialogPane().getScene().getWindow();
    stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
    alert.showAndWait();
}
public static void showAlert(String title, String content) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(content);
    Stage stage = (Stage) alert.getDialogPane().getScene().getWindow();
    stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
    alert.showAndWait();
}
```

```
@FXML
public void close() {
    Stage stage = (Stage) buttonEnter.getScene().getWindow();
    stage.close();
}
```

```
@FXML
void initialize() {
```

```
    textPasswordSee.setVisible(false);
```

```
    buttonSee.addEventHandler(MouseEvent.MOUSE_CLICKED, new
```

```
EventHandler<MouseEvent>() {
```

```
    @Override
    public void handle(MouseEvent mouseEvent) {
        if (textPassword.getText().isEmpty()) {
            textPassword.setText(textPassword.getText());
        }
    }
});
```

```
buttonSee.addEventHandler(MouseEvent.MOUSE_CLICKED, new
EventHandler<MouseEvent>() {
```

```
    @Override
    public void handle(MouseEvent mouseEvent) {
        k++;
        if (!textPassword.getText().isEmpty() & (k % 2 == 1)) {
            textPasswordSee.setText(textPassword.getText());
            textPasswordSee.setVisible(true);
            textPassword.setText(textPasswordSee.getText());
        } else {
            textPassword.setText(textPasswordSee.getText());
            textPasswordSee.setVisible(false);
        }
    }
});
```

```
buttonEnter.addEventHandler(MouseEvent.MOUSE_CLICKED, new
EventHandler<MouseEvent>() {
```

```
    @Override
    public void handle(MouseEvent mouseEvent) {
        try {
            if (!textLogin.getText().trim().isEmpty() &&
!textPassword.getText().trim().isEmpty()) {
                int n, n1, m, m1;
                n = database.getStaff(textLogin.getText(), textPassword.getText());
                m = database.getStaff(textLogin.getText(),
textPasswordSee.getText());
                n1 = database.getClients(textLogin.getText(), textPassword.getText());
                m1 = database.getClients(textLogin.getText(),
textPasswordSee.getText());
                if (n != 0 || m != 0 || n1 != 0 || m1 != 0) {
                    showAlert("", "Авторизация прошла успешно.");
                    id_client = database.getClientForReview(textLogin.getText());
                    close();
                }
            }
        } catch (Exception e) {
            showAlert("", "Ошибка авторизации.");
        }
    }
});
```

```

        if (Objects.equals(database.getRole(textLogin.getText()),
"Сотрудник")
        || Objects.equals(database.getRole(textLogin.getText()),
"Администратор")) {

            FXMLLoader fxmLoader = new
FXMLLoader(Main.class.getResource("MainAccount.fxml"));
            Scene scene = new Scene(fxmLoader.load(), 1093, 826);
            Stage stage = new Stage();
            stage.setScene(scene);
            stage.show();
            stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));

            MainAccount mainAccount = fxmLoader.getController();
            fio = database.getFioStaff(textLogin.getText());
            mainAccount.labelFio.setText(fio);
            role = String.valueOf(database.getRole(textLogin.getText()));
            mainAccount.labelRole.setText(role);
        } else {

            FXMLLoader fxmLoader = new
FXMLLoader(Main.class.getResource("account.fxml"));
            Scene scene = new Scene(fxmLoader.load(), 1093, 826);
            Stage stage = new Stage();
            stage.setScene(scene);
            stage.show();
            stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));

            Account account = fxmLoader.getController();
            fio = database.getFioClient(textLogin.getText());
            account.labelFio.setText(fio);
            account.labelRole.setText("Клиент");
        }
    } else {
        showAlertError("Ошибка", "Произошла ошибка при входе в
личный кабинет.");
        FXMLLoader fxmLoader = new
FXMLLoader(Main.class.getResource("captcha.fxml"));
        Scene scene = new Scene(fxmLoader.load(), 300, 140);
        Stage stage = new Stage();
        stage.setScene(scene);
    }
}

```

```

        stage.initStyle(StageStyle.UNDECORATED);
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.show();
        stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
    }
    }else showAlertError("Ошибка","Заполните все поля.");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
});

buttonRegistration.addEventHandler(MouseEvent.MOUSE_CLICKED, new
EventHandler<MouseEvent>() {

    @Override
    public void handle(MouseEvent mouseEvent) {
        try {
            FXMLLoader fxmLoader = new
FXMLLoader(Main.class.getResource("registration.fxml"));
            Scene scene = new Scene(fxmLoader.load(), 375, 448);
            Stage stage = new Stage();
            stage.setScene(scene);
            stage.show();
            stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
}
}

```

3.5 Разработка капчи

Данный модуль разработан для защиты от автоматизированных программ и ботов, которые могут использоваться для спама, взлома аккаунтов или других нежелательных действий.

Класс `CaptchaGenerator` генерирует капчу.

Класс `Captcha` содержит методы для генерации капчи, проверки

правильности введенного текста и обработки событий кнопки для проверки капчи. Также он отвечает за отображение окна с капчей и обработку событий мыши для проверки правильности введенного текста. (Рисунок 4)

Листинг класса `CaptchaGenerator`.

```
package com.example.coursework;

import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;

import java.util.Random;

public class CaptchaGenerator {
    private static final char[] captchaSymbols = {
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
        'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
    };

    private final Canvas canvas;
    private final GraphicsContext gc;

    public CaptchaGenerator(Canvas canvas) {
        this.canvas = canvas;
        this.gc = canvas.getGraphicsContext2D();
    }

    public static String text;
    public String generate(int length) {
        // Получаем текст из рандомных символов
        text = generateCaptchaText(length);

        int width = (int) this.canvas.getWidth();
        int height = (int) this.canvas.getHeight();

        // Закрашиваем место белым
        gc.setFill(Color.WHITE);
        gc.fillRect(0, 0, width, height);

        Random random = new Random();

        gc.setFont(new Font("Arial", height / 2));
```

```

// Рисуем символы со случайным смещением по высоте
int symbolOffset = width / length;
for (int i = 0; i < length; i++) {
    gc.setFill(new Color(random.nextDouble() * 0.5f, random.nextDouble() *
0.5f, random.nextDouble() * 0.5f, 0.5f + random.nextDouble() * 0.25f));
    gc.fillText(String.valueOf(text.charAt(i)), i * symbolOffset + (symbolOffset
>> 1), height / 2 + random.nextInt(height / 2));
}

// Рисуем шумовые линии
gc.beginPath();
for (int i = 0; i < 3; i++) {
    gc.setFill(new Color(random.nextDouble() * 0.5f, random.nextDouble() *
0.5f, random.nextDouble() * 0.5f, 0.5f + random.nextDouble() * 0.25f));
    gc.moveTo(0, random.nextInt(height));
    gc.lineTo(width, random.nextInt(height));
}
gc.stroke();

// Рисуем шумовые круги
for (int i = 0; i < 7; i++) {
    gc.setFill(new Color(random.nextDouble() * 0.5f, random.nextDouble() *
0.5f, random.nextDouble() * 0.5f, 0.5f + random.nextDouble() * 0.25f));
    gc.fillOval(random.nextInt(width), random.nextInt(height), width / 10, width /
10);
}

return text;
}

private String generateCaptchaText(int length) {
    StringBuilder builder = new StringBuilder();

    Random random = new Random();
    for (int i = 0; i < length; i++)
builder.append(captchaSymbols[random.nextInt(captchaSymbols.length)]);

    return builder.toString();
}
}

```

Листинг класса Captcha.

```
package com.example.coursework;

import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.input.MouseEvent;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

import java.util.Objects;

public class Captcha {
    private static final int captchaLength = 5;

    @FXML
    private Canvas canvas;
    @FXML
    private TextField textInputField;
    @FXML
    private Button buttonExam;

    private CaptchaGenerator captchaGenerator;
    private String captchaText;

    @FXML
    public boolean validate() {
        if (!Objects.equals(this.captchaText, this.textInputField.getText())) {
            regenerateCaptcha();
            this.textInputField.setText("");
            return false;
        }
        return true;
    }

    @FXML
    private void initialize() {
```

```

this.captchaGenerator = new CaptchaGenerator(this.canvas);

regenerateCaptcha();

buttonExam.addEventHandler(MouseEvent.MOUSE_CLICKED, new
EventHandler<MouseEvent>() {

    @Override
    public void handle(MouseEvent mouseEvent) {
        try {
            if(CaptchaGenerator.text.equals(textInputField.getText())){
                Authorization.showAlert("", "Проверка пройдена.");
                Stage stage = (Stage) textInputField.getScene().getWindow();
                stage.close();
            }else {
                Authorization.showAlertError("Ошибка", "Проверка не пройдена.
Попробуйте ещё раз.");
                Stage stage = (Stage) textInputField.getScene().getWindow();
                stage.close();
                FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("captcha.fxml"));
                Scene scene = new Scene(fxmlLoader.load(), 300, 140);
                Stage stage2 = new Stage();
                stage2.initStyle(StageStyle.UNDECORATED);
                stage2.setScene(scene);
                stage2.initModality(Modality.APPLICATION_MODAL);
                stage2.show();
                stage2.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
}

private void regenerateCaptcha() {
    this.captchaText = this.captchaGenerator.generate(captchaLength);
}
}

```

3.6 Разработка функционала регистрации пользователя

Регистрация предоставляет функционал для создания новых учетных записей пользователей в приложении. (Рисунок 5)

Данный модуль включает в себя: форма регистрации (предоставляет пользователю форму для ввода своих персональных данных), хранение данных (после успешной регистрации модуль сохраняет данные нового пользователя в базе данных), уведомления (модуль отправляет уведомление пользователю о результате регистрации).

Класс Registration отвечает за функционал окна регистрации.

Листинг класса Registration.

```
package com.example.coursework;

import javafx.application.Platform;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

import java.sql.SQLException;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.TimeUnit;

public class Registration {
    @FXML
    private Button buttonRegistration;

    @FXML
    private TextField regFio, regLogin, regPassword, regTel;
    @FXML
    private Label labelInfo;

    Database database = new Database();

    @FXML
    public void close() {
        Stage stage = (Stage) buttonRegistration.getScene().getWindow();
        stage.close();
    }
}
```

```

@FXML
void initialize() {

    buttonRegistration.addEventHandler(MouseEvent.MOUSE_CLICKED, new
EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent mouseEvent) {
            try {
                if (!regFio.getText().isEmpty() && !regLogin.getText().isEmpty() &&
!regPassword.getText().isEmpty() && !regTel.getText().isEmpty()) {
                    labelInfo.setText(database.clientAdd(regFio.getText(),
regLogin.getText(), regPassword.getText(), regTel.getText()));
                    if (labelInfo.getText().equals("Новый клиент успешно добавлен")) {
                        Timer timer = new Timer();
                        timer.schedule(new TimerTask() {
                            @Override
                            public void run() {
                                Platform.runLater(() -> close());
                            }
                        }, 2000);
                    }
                } else {
                    Authorization.showAlertError("Ошибка", "Заполните все поля.");
                }
            } catch (SQLException | ClassNotFoundException e) {
                throw new RuntimeException(e);
            }
        }
    });
}
}

```

3.7 Разработка раздела «Аккаунт»

Данный модуль разработан с целью возможности просмотра доступных для продажи товаров, их сортировки, а также с целью использования других функционалов приложения.

Класс ProductData содержит данные о продукте, такие как название, описание, цена, наличие, статус, фото, категория, производитель и модель. Класс имеет конструктор для инициализации этих данных и методы для получения каждого из них.

Листинг класса ProductData.

```
package com.example.coursework;

public class ProductData {
    private String name, description, status, photo;
    private Integer price, stock, category, manufacture, model;

    public ProductData(String name, String description, Integer price, Integer stock,
String status, String photo, Integer category, Integer manufacture, Integer model) {
        this.name = name;
        this.description = description;
        this.price = price;
        this.stock = stock;
        this.status = status;
        this.photo = photo;
        this.category = category;
        this.manufacture = manufacture;
        this.model = model;
    }

    public String getName() {
        return this.name;
    }
    public String getDescription() {
        return this.description;
    }
    public Integer getPrice(){return this.price;}
    public Integer getStock() {
        return this.stock;
    }
    public String getStatus(){return this.status;}
    public String getPhoto(){return this.photo;}
    public Integer getCategory(){return this.category;}
    public Integer getManufacture(){return this.manufacture;}
    public Integer getModel(){return this.model;}
}
```

Класс Data отображает данные о продукте в ячейке списка, используя информацию из базы данных.

Листинг класса Data.

```

package com.example.coursework;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.Label;
import javafx.scene.control.ListCell;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.HBox;

import java.io.File;
import java.io.IOException;
import java.net.MalformedURLException;
import java.sql.SQLException;

public class Data extends ListCell<ProductData> {
    @FXML
    AnchorPane anchorPane;
    @FXML
    ImageView imageView;
    @FXML
    Label labelCategory, labelDescription, labelManufacturer, labelName, labelPrice,
    labelStock;
    Database database = new Database();
    private FXMLLoader mLLoader;

    @Override
    protected void updateItem(ProductData productData, boolean empty) {
        super.updateItem(productData, empty);

        if (empty || productData == null) {
            setText(null);
            setGraphic(null);
        } else {
            if (mLLoader == null) {
                mLLoader = new FXMLLoader(getClass().getResource("data.fxml"));
                mLLoader.setController(this);

                try {
                    mLLoader.load();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```



```

    }

    //Установка серого фона ячеек для товаров с отсутствующим статусом
    String status = productData.getStatus();
    if (status != null && status.equals("Отсутствует")) {
        anchorPane.setStyle("-fx-background-color: gray;");
    } else {
        anchorPane.setStyle(""); // Сброс стиля ячейки
    }

    try {
        File file = new File(productData.getPhoto());
        String urlImage = file.toURI().toURL().toString();
        Image image = new Image(urlImage);
        imageView.setImage(image);
    } catch (MalformedURLException e) {
        throw new RuntimeException(e);
    }
    setText(null);
    setGraphic(anchorPane);

    labelName.setText(productData.getName());
    labelDescription.setText(productData.getDescription());
    labelPrice.setText(String.valueOf(productData.getPrice()));
    labelStock.setText(String.valueOf(productData.getStock()));
    try {
        labelCategory.setText(String.valueOf(database.getCategoryString(productData.getCategory())));

        labelManufacturer.setText(String.valueOf(database.getManufactureString(productData.getManufacture())));
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}
}
}

```

Разграничение пользователей по ролям, означает возможность пользоваться разным доступом к функционалу. Для клиента открывается форма, за которую отвечает класс «Account», для сотрудника открывается форма, за

которую отвечает класс «MainAccount». (Рисунок 6-7)

Листинг класса Account.

```
package com.example.coursework;

import javafx.collections.FXCollections;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Account {
    @FXML
    ImageView imageHome, imageAudi, imageKia, imageMitsubishi, imageClear,
    imageSearch;

    @FXML
    TextField textSearch;

    @FXML
    Label labelFio, labelRole;

    @FXML
    ComboBox<String> comboCategory, comboManufacture, comboStatus;

    @FXML
    ListView<ProductData> listView;

    @FXML
    Button createReview;

    public static int model;
```

```
static String id_productInAccount;
```

```
Database database = new Database();
```

```
@FXML
```

```
public void close() {
```

```
    Stage stage = (Stage) imageHome.getScene().getWindow();
```

```
    stage.close();
```

```
}
```

```
private void search() {
```

```
    textSearch.textProperty().addListener((observable, oldValue, newValue) -> {  
        String filter = newValue.toLowerCase();
```

```
        if (newValue.isEmpty()) { // Если строка поиска пустая, то отображаем  
все данные
```

```
        try {
```

```
            listView.getItems().clear();
```

```
            List<ProductData> allData = database.getProduct();
```

```
            listView.getItems().addAll(allData);
```

```
        } catch (SQLException | ClassNotFoundException e) {
```

```
            throw new RuntimeException(e);
```

```
        }
```

```
    }
```

```
    while (!filter.isEmpty()) {
```

```
        try {
```

```
            listView.getItems().clear();
```

```
            List<ProductData> ls =
```

```
database.getProductSearch(textSearch.getText());
```

```
            listView.getItems().addAll(ls);
```

```
        } catch (SQLException | ClassNotFoundException e) {
```

```
            throw new RuntimeException(e);
```

```
        }
```

```
        break;
```

```
    }
```

```
});
```

```
}
```

```
@FXML
```

```
void initialize() throws SQLException, ClassNotFoundException {
```

```
    loadInfo();
```

```

search();

createReview.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent
-> {
    try {
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("review.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 530, 358);
        Stage stage = new Stage();
        stage.setScene(scene);
        stage.show();
        stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
});

imageHome.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -
> {
    try {
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("authorization.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 409, 494);
        Stage stage = new Stage();
        stage.setScene(scene);
        stage.show();
        stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    close();
});
imageClear.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent ->
{
    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProduct();
        listView.getItems().addAll(ls);
        openContextMenu();
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}

```

```

    }
    comboCategory.getSelectionModel().select("Категория");
    comboStatus.getSelectionModel().select("Статус");
    comboManufacture.getSelectionModel().select("Поставщики");
});

imageAudi.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent ->
{
    model = 1;

    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProductModel(model);
        listView.getItems().addAll(ls);
        openContextMenu();
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
});
imageKia.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -> {
    model = 2;

    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProductModel(model);
        listView.getItems().addAll(ls);
        openContextMenu();
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
});
imageMitsubishi.addEventHandler(MouseEvent.MOUSE_CLICKED,
mouseEvent -> {
    model = 3;

    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProductModel(model);
        listView.getItems().addAll(ls);
        openContextMenu();
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
});

```

```

        imageSearch.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -
> {
            if (((!(comboCategory.getValue() == null) || !(comboCategory.getValue() ==
"Категория"))
                && (comboStatus.getValue() == null || comboStatus.getValue() ==
"Статус"))) {
                try {
                    //сортируем по категории
                    ArrayList<Integer> category =
database.getCategory(String.valueOf(comboCategory.getValue()));
                    listView.getItems().clear();
                    // Создаем отсортированный список категорий для вывода
                    List<ProductData> sortedItems = new ArrayList<>();
                    for (int i = 0; i < category.size(); i++) {
                        List<ProductData> ls =
database.getCategoryForSorting(category.get(i));
                        sortedItems.addAll(ls);
                    }
                    Collections.sort(sortedItems,
Comparator.comparing(ProductData::getCategory));
                    listView.getItems().addAll(sortedItems);

                } catch (Exception e) {
                    // Обработка ошибок
                }
            } else if (((!(comboCategory.getValue() == null) || (comboCategory.getValue()
== "Категория"))
                && (!(comboStatus.getValue() == null) || !(comboStatus.getValue() ==
"Статус")))) {
                try {
                    //сортируем по статусу
                    listView.getItems().clear();
                    List<ProductData> sortedItemsStat = new ArrayList<>();
                    List<ProductData> l =
database.getStatusForSorting(comboStatus.getValue());
                    sortedItemsStat.addAll(l);
                    Collections.sort(sortedItemsStat,
Comparator.comparing(ProductData::getStatus));
                    listView.getItems().addAll(sortedItemsStat);
                } catch (Exception e) {
                    // Обработка ошибок
                }
            } else if (((!(comboCategory.getValue() == null) ||

```

```

!(comboBoxCategory.getValue() == "Категория"))
    && (!(comboBoxStatus.getValue() == null) || !(comboBoxStatus.getValue() ==
"Статус")))) {
    try {
        if (comboBoxCategory.getValue() != null && comboBoxStatus.getValue() !=
null) {
            listView.getItems().clear();
            ArrayList<Integer> cat =
database.getCategory(String.valueOf(comboBoxCategory.getValue()));
            if (comboBoxStatus.getValue() == "Отсутствует") {
                ArrayList<Integer> stat = database.getProductStatNo();
                for (int i = 0; i < stat.size(); i++) {
                    for (int j = 0; j < cat.size(); j++) {
                        if (stat.get(i) == cat.get(j)) {
                            List<ProductData> l =
database.getProductWithId(stat.get(i));
                            listView.getItems().addAll(l);
                        }
                    }
                }
            } else {
                ArrayList<Integer> stat = database.getProductStat();
                for (int i = 0; i < stat.size(); i++) {
                    for (int j = 0; j < cat.size(); j++) {
                        if (stat.get(i) == cat.get(j)) {
                            List<ProductData> l =
database.getProductWithId(stat.get(i));
                            listView.getItems().addAll(l);
                        }
                    }
                }
            }
        } catch (Exception e) {
            // Обработка ошибок
        }
    }
});
}

```

```

void loadInfo() throws SQLException, ClassNotFoundException {

```

```

    List<String> cat = database.getCategoryMain();
    comboBoxCategory.setItems(FXCollections.observableArrayList(cat));

```

```

comboCategory.getItems().addAll("Категория");

comboStatus.getItems().addAll("Присутствует", "Отсутствует", "Статус");

List<String> man = database.getManufactureMain();
comboManufacture.setItems(FXCollections.observableArrayList(man));
comboManufacture.getItems().addAll("Поставщики");

comboManufacture.setOnAction(event -> {

    if (comboManufacture.getValue().equals("AutoPro")) {
        listView.getItems().clear();

        try {
            List<ProductData> ls = database.getProductSortMan(1);
            listView.getItems().addAll(ls);
            listView.setCellFactory(stringListView -> {
                ListCell<ProductData> cell = new Data();
                cell.setContextMenu(null);
                return cell;
            });
        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
    if (comboManufacture.getValue().equals("AutoTop")) {
        listView.getItems().clear();
        try {
            List<ProductData> ls = database.getProductSortMan(2);
            listView.getItems().addAll(ls);
            listView.setCellFactory(stringListView -> {
                ListCell<ProductData> cell = new Data();
                cell.setContextMenu(null);
                return cell;
            });
        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
    openContextMenu();
});

List<ProductData> ls = database.getProduct();
listView.getItems().addAll(ls);

```



```

listView.setCellFactory(stringListView -> {
    ListCell<ProductData> cell = new Data();
    cell.setContextMenu(null);
    openContextMenu();
    return cell;
});
}

public void openContextMenu() {

    listView.setCellFactory(stringListView -> {
        ListCell<ProductData> cell = new Data();
        ContextMenu contextMenu = new ContextMenu();

        MenuItem editItemPhoto = new MenuItem("Просмотр дополнительных
фото");
        editItemPhoto.setOnAction(event -> {
            ProductData item = cell.getItem();
            try {
                id_productInAccount =
String.valueOf(database.getIdProduct(item.getName()));
                FXMLLoader fxmLoader = new
FXMLLoader(Main.class.getResource("additionalPhotosForAccount.fxml"));
                Scene scene = new Scene(fxmLoader.load(), 518, 303);
                Stage stage = new Stage();
                stage.setScene(scene);
                stage.show();
                stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
            } catch (IOException | SQLException | ClassNotFoundException e) {
                throw new RuntimeException(e);
            }
        });

        contextMenu.getItems().addAll(editItemPhoto);
        cell.emptyProperty().addListener((obs, wasEmpty, isNowEmpty) -> {
            if (isNowEmpty) {
                cell.setContextMenu(null);
            } else {
                cell.setContextMenu(contextMenu);
            }
        });
        return cell;
    });
}

```

```
}  
}
```

Листинг класса MainAccount.

```
package com.example.coursework;  
  
import javafx.collections.FXCollections;  
import javafx.event.EventHandler;  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.control.*;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.scene.input.MouseEvent;  
import javafx.stage.Stage;  
  
import java.io.IOException;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.List;  
  
public class MainAccount {  
  
    @FXML  
    ImageView imageHome, imageAudi, imageKia, imageMitsubishi, imageClear,  
    imageSearch;  
  
    @FXML  
    TextField textSearch;  
  
    @FXML  
    Label labelFio, labelRole;  
  
    @FXML  
    ComboBox<String> comboCategory, comboManufacture, comboStatus;  
  
    @FXML  
    ListView<ProductData> listView;  
  
    @FXML  
    private Button createOrder, buttonAddProduct, buttonUpdate;
```

```
public static int model;
static String id_product;
```

```
Database database = new Database();
```

```
@FXML
```

```
public void close() {
    Stage stage = (Stage) imageHome.getScene().getWindow();
    stage.close();
}
```

```
private void search() {
    textSearch.textProperty().addListener((observable, oldValue, newValue) -> {
        String filter = newValue.toLowerCase();
```

```
        if (newValue.isEmpty()) { // Если строка поиска пустая, то отображаем
все данные
```

```
            try {
                listView.getItems().clear();
                List<ProductData> allData = database.getProduct();
                listView.getItems().addAll(allData);
            } catch (SQLException | ClassNotFoundException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

```
        while (!filter.isEmpty()) {
            try {
                listView.getItems().clear();
                List<ProductData> ls =
database.getProductSearch(textSearch.getText());
                listView.getItems().addAll(ls);
            } catch (SQLException | ClassNotFoundException e) {
                throw new RuntimeException(e);
            }
            break;
        }
    });
}
```

```
@FXML
```

```
void initialize() throws SQLException, ClassNotFoundException {
```

```

loadInfo();
search();

createOrder.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent ->
{
    try {
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("order.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 833, 538);
        Stage stage = new Stage();
        stage.setScene(scene);
        stage.show();
        stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
});
buttonAddProduct.addEventHandler(MouseEvent.MOUSE_CLICKED,
mouseEvent -> {
    try {
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("productAdd.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 524, 597);
        Stage stage = new Stage();
        stage.setScene(scene);
        stage.show();
        stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
});
imageHome.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -
> {
    try {
        FXMLLoader fxmlLoader = new
FXMLLoader(Main.class.getResource("authorization.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 409, 494);
        Stage stage = new Stage();
        stage.setScene(scene);
        stage.show();
        stage.getIcons().add(new

```

```

Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    close();
});
imageAudi.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent ->
{
    model = 1;

    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProductModel(model);
        listView.getItems().addAll(ls);
        openContextMenu();
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
});
imageKia.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -> {
    model = 2;

    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProductModel(model);
        listView.getItems().addAll(ls);
        openContextMenu();
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
});
imageMitsubishi.addEventHandler(MouseEvent.MOUSE_CLICKED,
mouseEvent -> {
    model = 3;

    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProductModel(model);
        listView.getItems().addAll(ls);
        openContextMenu();
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}

```

```

    });

    imageClear.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent ->
{
    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProduct();
        listView.getItems().addAll(ls);
        openContextMenu();
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
    comboCategory.getSelectionModel().select("Категория");
    comboStatus.getSelectionModel().select("Статус");
    comboManufacture.getSelectionModel().select("Поставщики");
});

    buttonUpdate.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent
-> {
    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProduct();
        listView.getItems().addAll(ls);
        openContextMenu();
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
});

    imageSearch.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -
> {
        if (((!comboCategory.getValue() == null) || !(comboCategory.getValue() ==
"Категория"))
            && (comboStatus.getValue() == null || comboStatus.getValue() ==
"Статус")) {
            try {
                //сортируем по категории
                ArrayList<Integer> category =
database.getCategory(String.valueOf(comboCategory.getValue()));
                listView.getItems().clear();
                // Создаем отсортированный список категорий для вывода
                List<ProductData> sortedItems = new ArrayList<>();
                for (int i = 0; i < category.size(); i++) {
                    List<ProductData> ls =

```

```

database.getCategoryForSorting(category.get(i));
        sortedItems.addAll(l);
    }
    Collections.sort(sortedItems,
Comparator.comparing(ProductData::getCategory));
    listView.getItems().addAll(sortedItems);

    } catch (Exception e) {
        // Обработка ошибок
    }
    } else if (((comboCategory.getValue() == null) || (comboCategory.getValue()
== "Категория"))
        && (!(comboStatus.getValue() == null) || !(comboStatus.getValue() ==
"Статус")))) {
        try {
            //сортируем по статусу
            listView.getItems().clear();
            List<ProductData> sortedItemsStat = new ArrayList<>();
            List<ProductData> l =
database.getStatusForSorting(comboStatus.getValue());
            sortedItemsStat.addAll(l);
            Collections.sort(sortedItemsStat,
Comparator.comparing(ProductData::getStatus));
            listView.getItems().addAll(sortedItemsStat);
        } catch (Exception e) {
            // Обработка ошибок
        }
    } else if (((!(comboCategory.getValue() == null) ||
!(comboCategory.getValue() == "Категория"))
        && (!(comboStatus.getValue() == null) || !(comboStatus.getValue() ==
"Статус")))) {
        try {
            if (comboCategory.getValue() != null && comboStatus.getValue() !=
null) {
                listView.getItems().clear();
                ArrayList<Integer> cat =
database.getCategory(String.valueOf(comboCategory.getValue()));
                if (comboStatus.getValue() == "Отсутствует") {
                    ArrayList<Integer> stat = database.getProductStatNo();
                    for (int i = 0; i < stat.size(); i++) {
                        for (int j = 0; j < cat.size(); j++) {
                            if (stat.get(i) == cat.get(j)) {
                                List<ProductData> l = database.getProductWithId(stat.get(i));
                                listView.getItems().addAll(l);

```

```

        }
    }
} else {
    ArrayList<Integer> stat = database.getProductStat();
    for (int i = 0; i < stat.size(); i++) {
        for (int j = 0; j < cat.size(); j++) {
            if (stat.get(i) == cat.get(j)) {
                List<ProductData> l = database.getProductWithId(stat.get(i));
                listView.getItems().addAll(l);
            }
        }
    }
}
} catch (Exception e) {
    // Обработка ошибок
}
});
}
}

```

```

void loadInfo() throws SQLException, ClassNotFoundException {

```

```

    List<String> cat = database.getCategoryMain();
    comboCategory.setItems(FXCollections.observableArrayList(cat));
    comboCategory.getItems().addAll("Категория");

    comboStatus.getItems().addAll("Присутствует", "Отсутствует", "Статус");

    List<String> man = database.getManufactureMain();
    comboManufacture.setItems(FXCollections.observableArrayList(man));
    comboManufacture.getItems().addAll("Поставщики");

    comboManufacture.setOnAction(event -> {

        if (comboManufacture.getValue().equals("AutoPro")) {
            listView.getItems().clear();

            try {
                List<ProductData> ls = database.getProductSortMan(1);
                listView.getItems().addAll(ls);
                listView.setCellFactory(stringListView -> {

```



```

        ListCell<ProductData> cell = new Data();
        cell.setContextMenu(null);
        return cell;
    });
} catch (SQLException | ClassNotFoundException e) {
    throw new RuntimeException(e);
}
}
}
if (comboManufacture.getValue().equals("AutoTop")) {
    listView.getItems().clear();
    try {
        List<ProductData> ls = database.getProductSortMan(2);
        listView.getItems().addAll(ls);
        listView.setCellFactory(stringListView -> {
            ListCell<ProductData> cell = new Data();
            cell.setContextMenu(null);
            return cell;
        });
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}
openContextMenu();
});
}

List<ProductData> ls = database.getProduct();
listView.getItems().addAll(ls);
listView.setCellFactory(stringListView -> {
    ListCell<ProductData> cell = new Data();
    cell.setContextMenu(null);
    openContextMenu();
    return cell;
});
}

```

```

public void openContextMenu() {

    listView.setCellFactory(stringListView -> {
        ListCell<ProductData> cell = new Data();
        ContextMenu contextMenu = new ContextMenu();

        MenuItem editItemEdit = new MenuItem("Редактировать");
        editItemEdit.setOnAction(event -> {
            ProductData item = cell.getItem();

```

```

        try {
            id_product = String.valueOf(database.getIdProduct(item.getName()));
            FXMLLoader fxmLoader = new
FXMLLoader(Main.class.getResource("productEdit.fxml"));
            Stage stage = new Stage();
            Scene scene = new Scene(fxmLoader.load(), 524, 597);
            stage.setScene(scene);
            stage.show();
            stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
        } catch (SQLException | ClassNotFoundException | IOException e) {
            throw new RuntimeException(e);
        }
    });
    MenuItem editItemDelete = new MenuItem("Удалить");
    editItemDelete.setOnAction(event -> {
        ProductData item = cell.getItem();
        try {
            id_product = String.valueOf(database.getIdProduct(item.getName()));
            database.deleteAdditionalPhotot(Integer.valueOf(id_product));
            database.deleteProduct(Integer.valueOf(id_product));
            Authorization.showAlert("", "Данные удалены. Обновите.");
        } catch (SQLException | ClassNotFoundException e) {
            Authorization.showAlertError("Ошибка", "Этот товар нельзя удалить,
пока он есть в заказе.");
        }
    });

    MenuItem editItemPhoto = new MenuItem("Просмотр дополнительных
фото");
    editItemPhoto.setOnAction(event -> {
        ProductData item = cell.getItem();
        try {
            id_product = String.valueOf(database.getIdProduct(item.getName()));
            FXMLLoader fxmLoader = new
FXMLLoader(Main.class.getResource("additionalPhotos.fxml"));
            Scene scene = new Scene(fxmLoader.load(), 518, 303);
            Stage stage = new Stage();
            stage.setScene(scene);
            stage.show();
            stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
        } catch (IOException | SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    });

```

```

    }
    });

    MenuItem editItemPhotoAdd = new MenuItem("Добавить дополнительные
фото");
    editItemPhotoAdd.setOnAction(event -> {
        ProductData item = cell.getItem();
        try {
            id_product = String.valueOf(database.getIdProduct(item.getName()));
            FXMLLoader fxmlloader = new
FXMLLoader(Main.class.getResource("additionalPhotosAdd.fxml"));
            Scene scene = new Scene(fxmlloader.load(), 388, 346);
            Stage stage = new Stage();
            stage.setScene(scene);
            stage.show();
            stage.getIcons().add(new
Image("C:/Users/Anna/IdeaProjects/coursework/logo.png"));
        } catch (IOException | SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    });
    contextMenu.getItems().addAll(editItemEdit, editItemDelete,
editItemPhotoAdd, editItemPhoto);
    cell.emptyProperty().addListener((obs, wasEmpty, isNowEmpty) -> {
        if (isNowEmpty) {
            cell.setContextMenu(null);
        } else {
            cell.setContextMenu(contextMenu);
        }
    });
    return cell;
});
}
}

```

3.8 Разработка функционала к аккаунту сотрудника

Данный модуль предоставляет сотруднику возможность добавления, редактирования, удаления товара, добавление и просмотр дополнительных фото, оформление заказа. (Рисунок 10-14)

Класс ProductAdd отвечает за отображение окна добавления продукта, ввод данных и их отправку в базу данных.

Листинг класса ProductAdd.

```

package com.example.coursework;

import javafx.collections.FXCollections;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

import java.sql.SQLException;
import java.util.List;

public class ProductAdd {
    @FXML
    private Button buttonAdd;

    @FXML
    private ComboBox<String> comboCategory, comboManufacture, comboModel;

    @FXML
    private TextField textPrice, textImage, textName, textStock;

    @FXML
    private TextArea textDescription;

    String getStatusForAdd, getImageForAdd;
    Database database = new Database();
    @FXML
    public void close() {
        Stage stage = (Stage) buttonAdd.getScene().getWindow();
        stage.close();
    }

    @FXML
    void initialize() throws SQLException, ClassNotFoundException {

        List<String> cat = database.getCategoryMain();
        comboCategory.setItems(FXCollections.observableArrayList(cat));

        List<String> man = database.getManufactureMain();
        comboManufacture.setItems(FXCollections.observableArrayList(man));
    }
}

```

```

List<String> mod = database.getModelMain();
comboModel.setItems(FXCollections.observableArrayList(mod));

loadInfo();
}

void loadInfo() {
    try {
        buttonAdd.addEventHandler(MouseEvent.MOUSE_CLICKED, new
EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent mouseEvent) {
                try {
                    if (!textName.getText().isEmpty() &&
!textDescription.getText().isEmpty() && !textPrice.getText().isEmpty()
&& !textStock.getText().isEmpty() &&
comboCategory.getValue() != null
&& comboManufacture.getValue() != null &&
comboModel.getValue() != null
&& (Integer.parseInt(textPrice.getText()) > 0) &&
Integer.parseInt(textStock.getText()) >= 0) {
                        if (textImage.getText().isEmpty()) {
                            getImageForAdd = "no.jpg";
                        } else getImageForAdd = textImage.getText();
                        if (Integer.parseInt(textStock.getText()) == 0) {
                            getStatusForAdd = "Отсутствует";
                        } else getStatusForAdd = "Присутствует";

                        database.insertProduct(textName.getText(),
textDescription.getText(),
Integer.parseInt(textPrice.getText()),
Integer.parseInt(textStock.getText()),
getStatusForAdd, getImageForAdd,
database.getCategoryForInsert(comboCategory.getValue()),
database.getManufactureForInsert(comboManufacture.getValue()),
database.getModelForInsert(comboModel.getValue()));
                        Authorization.showAlert("", "Товар добавлен. Обновите.");
                        close();
                    } else {
                        Authorization.showAlertError("Ошибка", "Заполните все поля
или проверьте корректность вводимых данных.");
                    }
                }
            }
        });
    }
}

```

```

        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
});
} catch (Exception e) {
    throw new RuntimeException(e);
}
}
}

```

Класс ProductEdit отвечает за отображение окна редактирования продукта, ввод данных и обновления данных в базе данных.

Листинг класса ProductEdit.

```

package com.example.coursework;

import javafx.collections.FXCollections;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

import java.io.File;
import java.net.MalformedURLException;
import java.sql.SQLException;
import java.util.List;

public class ProductEdit extends ListCell<ProductData> {

    @FXML
    private Button buttonEdit;

    @FXML
    private ComboBox<String> comboCategory, comboManufacture, comboModel;

    @FXML
    private TextField textPrice, textImage, textName, textStock;

```

```

@FXML
private TextArea textDescription;

String getStatusForEdit, getImageForEdit;
Database database = new Database();
@FXML
public void close() {
    Stage stage = (Stage) buttonEdit.getScene().getWindow();
    stage.close();
}

@FXML
void initialize() throws SQLException, ClassNotFoundException {

    textName.setText(database.getOneProductName(MainAccount.id_product));

    textPrice.setText(String.valueOf(database.getOneProductPrice(MainAccount.id_product)));

    textStock.setText(String.valueOf(database.getOneProductQuantityInStock(MainAccount.id_product)));

    textDescription.setText(database.getOneProductDescription(MainAccount.id_product));

    textImage.setText(database.getOneProductPhoto(MainAccount.id_product));

    List<String> cat = database.getCategoryMain();
    comboCategory.setItems(FXCollections.observableArrayList(cat));

    List<String> man = database.getManufactureMain();
    comboManufacture.setItems(FXCollections.observableArrayList(man));

    List<String> mod = database.getModelMain();
    comboModel.setItems(FXCollections.observableArrayList(mod));

    loadInfo();
}

void loadInfo() {
    try {
        buttonEdit.addEventHandler(MouseEvent.MOUSE_CLICKED, new
EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent mouseEvent) {

```

```

        try {
            if (!textName.getText().isEmpty() &&
!textDescription.getText().isEmpty() && !textPrice.getText().isEmpty()
                && !textStock.getText().isEmpty() &&
comboCategory.getValue() != null
                && comboManufacture.getValue() != null &&
comboModel.getValue() != null
                && (Integer.parseInt(textPrice.getText()) > 0) &&
Integer.parseInt(textStock.getText()) >= 0) {
                if (textImage.getText().isEmpty()) {
                    getImageForEdit = "no.jpg";
                } else getImageForEdit = textImage.getText();
                if (Integer.parseInt(textStock.getText()) == 0) {
                    getStatusForEdit = "Отсутствует";
                } else getStatusForEdit = "Присутствует";

                database.updateProduct(textName.getText(),
textDescription.getText(),Integer.parseInt(textPrice.getText()),
                    Integer.parseInt(textStock.getText()), getStatusForEdit,
getImageForEdit, database.getCategoryForInsert(comboCategory.getValue()),

database.getManufactureForInsert(comboManufacture.getValue()),
database.getModelForInsert(comboModel.getValue()),
                    Integer.parseInt(MainAccount.id_product));

                Authorization.showAlert("", "Данные отредактированы.
Обновите.");
                close();
            } else {
                Authorization.showAlertError("Ошибка", "Заполните все поля
или проверьте корректность данных.");
            }
        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
});
} catch (Exception e) {
    throw new RuntimeException(e);
}
}
}

```


Класс Order отвечает за отображение окна формирования заказа, и добавления данных в базу данных.

Листинг класса Order.

```
package com.example.coursework;

import javafx.collections.FXCollections;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.input.MouseEvent;

import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Order {
    @FXML
    private Button buttonCreateOrder, buttonAddOrder, buttonClear;

    @FXML
    private ComboBox<String> comboClient, comboPoint, comboProduct;

    @FXML
    private TextField countProduct;

    @FXML
    private Label labelPrice;

    @FXML
    private ListView<String> listCount, listPrice, listProduct;

    double finalPrice, newPrice, price = 0;

    Database database = new Database();

    @FXML
    void initialize() throws SQLException, ClassNotFoundException {
        loadInfo();
    }

    void loadInfo() throws SQLException, ClassNotFoundException {
```

```

labelPrice.setText(String.valueOf(price));

List<String> point = database.getPoint();
comboPoint.setItems(FXCollections.observableArrayList(point));

List<String> client = database.getClient();
comboClient.setItems(FXCollections.observableArrayList(client));

List<String> product = database.getProductMain();
comboProduct.setItems(FXCollections.observableArrayList(product));

buttonCreateOrder.setDisable(true);

buttonAddOrder.addEventHandler(MouseEvent.MOUSE_CLICKED,
mouseEvent -> {
    if (comboClient.getValue() == null || comboPoint.getValue() == null ||
comboProduct.getValue() == null
        || countProduct.getText() == null ||
Integer.parseInt(countProduct.getText()) <= 0) {
        Authorization.showAlertError("Ошибка", "Заполните данные
корректно.");
    } else {

        buttonCreateOrder.setDisable(false);

        try {
            price = database.getPriceInt(comboProduct.getValue());
            labelPrice.setText(String.valueOf(price));
        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
        for (int i = 0; i < listProduct.getItems().size(); i++) {
            if (listProduct.getItems().get(i).equals(comboProduct.getValue())) {
                int newCount = Integer.parseInt(listCount.getItems().get(i)) +
Integer.parseInt(countProduct.getText());
                listCount.getItems().set(i, String.valueOf(newCount));
                newPrice = Double.parseDouble(String.valueOf(price * newCount));
                listPrice.getItems().set(i, String.valueOf(newPrice));
                labelPrice.setText(String.valueOf(newPrice));

                finalPrice = 0;
                for (int j = 0; j < listPrice.getItems().size(); j++) {
                    finalPrice += Double.parseDouble(listPrice.getItems().get(j));
                    labelPrice.setText(String.valueOf(finalPrice));
                }
            }
        }
    }
});

```

```

        }
        return;
        // ВЫХОДИМ ИЗ ЦИКЛА, ТАК КАК ТОВАР НАЙДЕН И ОБНОВЛЕН
    }
}
// Если товар не был найден в списке, добавляем новый элемент
listProduct.getItems().add(comboProduct.getValue());
listCount.getItems().add(countProduct.getText());
try {
    List<String> ls = database.getPrice(comboProduct.getValue(),
countProduct.getText());
    listPrice.getItems().addAll(ls);
} catch (SQLException | ClassNotFoundException e) {
    throw new RuntimeException(e);
}
finalPrice = 0;
for (int i = 0; i < listPrice.getItems().size(); i++) {
    finalPrice += Double.parseDouble(listPrice.getItems().get(i));
    labelPrice.setText(String.valueOf(finalPrice));
}
}
});

buttonCreateOrder.addEventHandler(MouseEvent.MOUSE_CLICKED,
mouseEvent -> {
    try {
        for (int i = 0; i < listCount.getItems().size(); i++) {
            int count = Integer.parseInt(listCount.getItems().get(i));
            int finalPrice = (int) Double.parseDouble(listPrice.getItems().get(i));
            int getPoint = database.getPointForInsert(comboPoint.getValue());
            int getProduct =
database.getProductForInsert(String.valueOf(listProduct.getItems().get(i)));
            int getClient =
database.getClientForInsert(String.valueOf(comboClient.getValue()));
            database.insertOrder(count, finalPrice, getPoint, getProduct, getClient);
        }
        Authorization.showAlert("", "Заказ оформлен.");
    } catch (SQLException | ClassNotFoundException e) {
        // Обрабатываем исключение
        Authorization.showError("Ошибка", "Недостаточно количества на
складе.");
    }
});
buttonClear.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent ->

```

```

{
    listProduct.getItems().clear();
    listPrice.getItems().clear();
    listCount.getItems().clear();
    labelPrice.setText("0.0");
});
}
}

```

Класс AdditionalPhotos отвечает за отображение окна, где можно посмотреть дополнительные фотографии к тому или иному товару.

Листинг класса AdditionalPhotos.

```
package com.example.coursework;
```

```

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;

```

```

import java.io.File;
import java.net.MalformedURLException;
import java.sql.SQLException;
import java.util.ArrayList;

```

```
public class AdditionalPhotos {
```

```
    @FXML
```

```
    Button button1, button2;
```

```
    @FXML
```

```
    private ImageView image;
```

```
    @FXML
```

```
    Label labelName;
```

```
    Database database = new Database();
```

```
    private int currentPhotoIndex = 0;
```

```
    ArrayList<String> listPhoto; // Создаем список для хранения имен файлов или
URL-адресов
```

```
    @FXML
```

```
    void initialize() {
```

```
        try {
```

```

        labelName.setText(database.getProductName(MainAccount.id_product));
        listPhoto = database.getPhoto(Integer.valueOf(MainAccount.id_product)); //
Заполняем список фотографиями из базы данных
        displayImage(0); // Отображаем первую фотографию при инициализации
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }

    button1.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -> {
        showPreviousImage(); // Показываем предыдущую фотографию при
нажатии на кнопку
    });

    button2.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -> {

        showNextImage(); // Показываем следующую фотографию при нажатии
на кнопку
    });
}
private void displayImage(int index) {
    try {
        String urlImage = new File(listPhoto.get(index)).toURI().toURL().toString();
// Преобразуем путь к файлу в URL
        Image image1 = new Image(urlImage);
        image.setImage(image1); // Отображаем изображение в ImageView
    } catch (Exception e) {
        Authorization.showAlertError("Ошибка", "Нет дополнительных фото к
выбранному товару.");
        // Выводим информацию об исключении
    }
}

private void showNextImage() {
    if (currentPhotoIndex < listPhoto.size() - 1) {
        currentPhotoIndex++; // Увеличиваем индекс для отображения
следующей фотографии
        displayImage(currentPhotoIndex); // Отображаем следующую
фотографию
    }
}

// Метод для отображения предыдущей фотографии
private void showPreviousImage() {
    if (currentPhotoIndex > 0) {

```

```

        currentPhotoIndex--; // Уменьшаем индекс для отображения
предыдущей фотографии
        displayImage(currentPhotoIndex); // Отображаем предыдущую
фотографию
    }
}
}

```

Класс AdditionalPhotosAdd отвечает за отображение окна, где можно добавить дополнительные фотографии к тому или иному товару.

Листинг класса AdditionalPhotosAdd.

```

package com.example.coursework;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;

import java.sql.SQLException;

public class AdditionalPhotosAdd {

    @FXML
    private Button buttonAdd;

    @FXML
    private TextField nameImage1, nameImage2;

    Database database = new Database();

    @FXML
    void initialize() {
        buttonAdd.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent ->
        {

            if (nameImage1.getText().trim().isEmpty() &&
nameImage2.getText().trim().isEmpty()) {
                Authorization.showError("Ошибка", "Заполните хотя бы одно
поле.");
            }
            if (!nameImage1.getText().trim().isEmpty() &&

```

```

nameImage2.getText().trim().isEmpty()) {
    try {
        database.insertAdditionalPhotos(nameImage1.getText(),
Integer.valueOf(MainAccount.id_product));
        Authorization.showAlert("", "Фото добавлено.");
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}
if (!nameImage2.getText().trim().isEmpty() &&
nameImage1.getText().trim().isEmpty()) {
    try {
        database.insertAdditionalPhotos(nameImage2.getText(),
Integer.valueOf(MainAccount.id_product));
        Authorization.showAlert("", "Фото добавлено.");
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}
if (!nameImage2.getText().trim().isEmpty() &&
!nameImage1.getText().trim().isEmpty()) {
    try {
        database.insertAdditionalPhotos(nameImage1.getText(),
Integer.valueOf(MainAccount.id_product));
        database.insertAdditionalPhotos(nameImage2.getText(),
Integer.valueOf(MainAccount.id_product));
        Authorization.showAlert("", "Фото добавлены.");
    } catch (SQLException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}
});
}
}

```

3.9 Разработка функционала к аккаунту клиента

Данный модуль предоставляет клиенту доступ к дополнительному функционалу: просмотр дополнительных фотографий продукта и оценку выбранного продукта. (Рисунок 8-9)

Класс AdditionalPhotosForAccount отвечает за отображение дополнительных фотографий для учетной записи пользователя.

Листинг класса AdditionalPhotosForAccount.

```
package com.example.coursework;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import java.io.File;
import java.sql.SQLException;
import java.util.ArrayList;

public class AdditionalPhotosForAccount {
    @FXML
    Button button1, button2;
    @FXML
    private ImageView image;
    @FXML
    Label labelName;
    Database database = new Database();
    private int currentPhotoIndex = 0;

    ArrayList<String> listPhoto; // Создаем список для хранения имен файлов или
    URL-адресов

    @FXML
    void initialize() {
        try {

labelName.setText(database.getProductName(Account.id_productInAccount));
            listPhoto =
database.getPhoto(Integer.valueOf(Account.id_productInAccount)); // Заполняем
список фотографиями из базы данных
            displayImage(0); // Отображаем первую фотографию при инициализации
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        }

        button1.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -> {
            showPreviousImage(); // Показываем предыдущую фотографию при
нажатии на кнопку
        });
    }
}
```



```

        button2.addEventHandler(MouseEvent.MOUSE_CLICKED, mouseEvent -> {

            showNextImage(); // Показываем следующую фотографию при нажатии
на кнопку
        });
    }
    private void displayImage(int index) {
        try {
            String urlImage = new File(listPhoto.get(index)).toURI().toURL().toString();
// Преобразуем путь к файлу в URL
            Image image1 = new Image(urlImage);
            image.setImage(image1); // Отображаем изображение в ImageView
        } catch (Exception e) {
            Authorization.showAlertError("Ошибка", "Нет дополнительных фото к
выбранному товару.");
            // Выводим информацию об исключении
        }
    }

    private void showNextImage() {
        if (currentPhotoIndex < listPhoto.size() - 1) {
            currentPhotoIndex++; // Увеличиваем индекс для отображения
следующей фотографии
            displayImage(currentPhotoIndex); // Отображаем следующую фотографию
        }
    }

    // Метод для отображения предыдущей фотографии
    private void showPreviousImage() {
        if (currentPhotoIndex > 0) {
            currentPhotoIndex--; // Уменьшаем индекс для отображения предыдущей
фотографии
            displayImage(currentPhotoIndex); // Отображаем предыдущую
фотографию
        }
    }
}

```

Класс Review отвечает за отображение окна отзыва, где пользователь может выбрать продукт из выпадающего списка, оставить комментарий и оценку.

Листинг класса Review.

```

package com.example.coursework;

import javafx.collections.FXCollections;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.stage.Stage;

import java.io.IOException;
import java.sql.SQLException;
import java.util.List;

public class Review {
    @FXML
    private Button buttonSend;

    @FXML
    private ComboBox<String> comboProduct;

    @FXML
    private TextArea textComment;

    @FXML
    private TextField textRating;
    Database database = new Database();

    @FXML
    void initialize() throws SQLException, ClassNotFoundException {

        List<String> prod = database.getProductMain();
        comboProduct.setItems(FXCollections.observableArrayList(prod));

        buttonSend.addEventHandler(MouseEvent.MOUSE_CLICKED, new
        EventHandler<MouseEvent>() {
            @Override
            public void handle(MouseEvent mouseEvent) {
                try {
                    if (comboProduct.getValue() == null || textRating.getText().isEmpty()) {

```

```

        Authorization.showAlertError("Ошибка", "Заполните все данные.");
    } else {
        if(Integer.parseInt(textRating.getText()) > 0 &&
Integer.parseInt(textRating.getText()) <= 5) {
            database.insertReviews(textRating.getText(),
textComment.getText(), Authorization.id_client,
            database.getProductForInsert(comboProduct.getValue()));
            Authorization.showAlert("", "Спасибо за Ваш отзыв!");
        }else Authorization.showAlertError("Ошибка", "Оцените нас от 1
до 5.");
    }
} catch (SQLException | ClassNotFoundException e) {
    throw new RuntimeException(e);
}
}
});
}
}
}

```

4. Руководство пользователя

4.1 Страница главного окна

Для того, чтобы попасть на окно авторизации, необходимо нажать на соответствующую кнопку в верхнем правом углу. (Рисунок 2)

4.2 Окна авторизации, регистрации, капчи

Для того, чтобы попасть в аккаунт и получить доступ ко всем его функциям, необходимо корректно выполнить вход в систему, пройдя процедуру авторизации. Форма авторизации предложит ввести соответствующие данные.

На форме будет предложено ввести логин и пароль в соответствующие одноименные поля.

Поле «Логин» должно быть заполнено соответствующим логином пользователя в системе. Поле «Пароль» требует ввода личного пароля пользователя для его дальнейшей авторизации в системе. При необходимости пароль можно посмотреть, нажав на соответствующую кнопку.

После ввода персональных данных необходимо нажать левой кнопкой мыши на кнопку «Авторизоваться» и, в случае ввода корректных данных, пользователь получает доступ к главной странице приложения, в противном случае пользователю будет отображено сообщение с ошибкой о некорректных данных для входа.

Если аккаунта ещё нет, то есть возможность регистрации пользователя, необходимо нажать на соответствующую кнопку внизу формы авторизации. Когда откроется окно регистрации необходимо заполнить данные и зарегистрироваться. После регистрации следует авторизоваться.

Если данные введены некорректно, а именно: вводимые данные не находятся в базе данных, то с целями безопасности на экран выводится капча и блокируется остальная часть программы, пока пользователь не введёт правильно символы с капчи. (Рисунок 7-6, 15-19)

4.3 Главная страница сотрудника

На главной странице отображается информация о существующих для продажи продуктов с возможностью поиска по названию и сортировки по выбранному фильтру.

Также на странице расположены иконки, позволяющие отсортировать продукты по марке автомобиля или же вернуть список к начальному состоянию.

При вводе поискового запроса в соответствующее поле и выбора условий сортировки и фильтрации список продуктов обновляется в реальном времени.

Также на странице есть кнопки, которые открывают дополнительный функционал: добавление товара, обновление списка товаров, оформление заказа.

Помимо кнопок, сотруднику предоставляет функционал контекстное меню, которое содержит такой функционал, как: редактирование, удаление, добавление или просмотр дополнительных фото выбранного товара. (Рисунок 13, 20-25)

4.4 Главная страница клиента

На главной странице отображается информация о существующих для продажи продуктов с возможностью поиска по названию и сортировки по выбранному фильтру.

Также на странице расположены иконки, позволяющие отсортировать продукты по марке автомобиля или же вернуть список к начальному состоянию.

При вводе поискового запроса в соответствующее поле и выбора условий сортировки и фильтрации список продуктов обновляется в реальном времени.

Клиенту предоставляет функционал контекстное меню: просмотр дополнительных фото выбранного товара.

Также клиент может оставить отзыв к конкретному товару, поставив оценку от 1 до 5 и написав комментарий. (Рисунок 13, 26-28)

Заключение

В современном мире автозапчасти стали неотъемлемой частью автомобильной индустрии, поскольку они играют важную роль как для автовладельцев, так и для автомобильных сервисов. С постоянно растущим спросом на автозапчасти возникают новые вызовы, представленные в области эффективного управления складскими операциями, точного учета товаров и обслуживания клиентов. В контексте этой динамичной среды для успешной деятельности магазинов автозапчастей ключевым фактором является оптимальное функционирование базы данных, специально адаптированной под нужды данной отрасли.

В ходе работы выполнялась разработка и создание базы данных в сочетании с информационной системой, предназначенной для удовлетворения потребностей магазина автозапчастей. Это позволит не только оптимизировать управление складскими операциями, но и автоматизировать процессы учета товаров и обслуживания клиентов, в конечном итоге повысив эффективность работы и улучшив удовлетворенность клиентов.

Основные задачи, которые были поставлены перед данной работой, охватывают анализ требований и потребностей магазина автозапчастей, проектирование структуры базы данных, разработку функциональной части информационной системы, ее реализацию и интеграцию, а также анализ эффективности использования базы данных и информационной системы в повседневной работе магазина.

Результатом работы является разработанное и функционирующее настольное приложение для магазина автозапчастей, которое в полном объеме соответствует с описанным заданием на курсовое проектирование. Так же в процессе выполнения проекта была разработана документация, для подробного ознакомления пользователей с работой в системе.

Список литературы

1. Документация sql

URL: <https://dev.mysql.com/doc/>

2. Документация Java

URL: <https://metanit.com/java/tutorial/?ysclid=lp6x9ct2z8558025549>

3. Как открыть магазин автозапчастей

URL: <https://kdelu.vtb.ru/articles/kak-otkryt-magazin-avtozapchastej/>

4. Секреты автобизнеса

URL: <https://www.drom.ru/info/misc/33301.html>

5. Как работают магазины автозапчастей

URL: <https://nirax.ru/kak-rabotayut-magaziny-avtozapchastej/>

Приложения

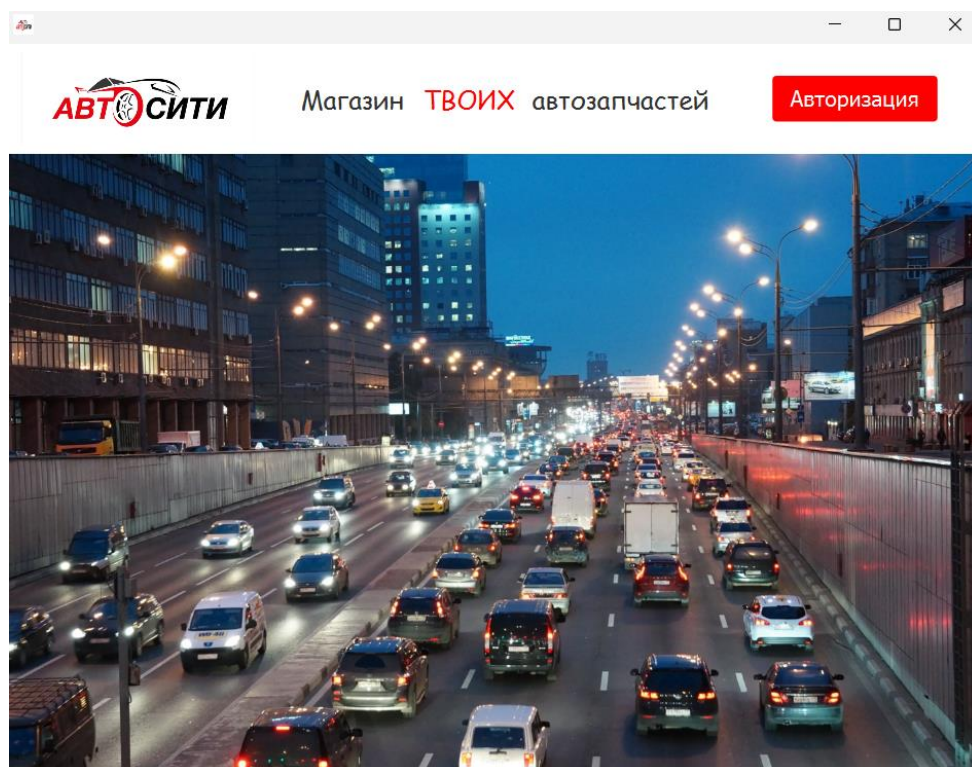


Рисунок 2. Главное окно приложения

АВТОРИЗАЦИЯ

Логин

Пароль

Войти

Нет аккаунта?

Зарегистрироваться

Рисунок 3. Форма авторизации

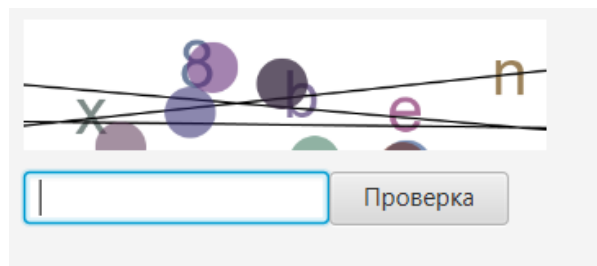
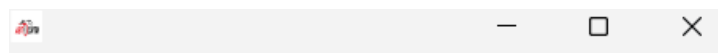


Рисунок 4. Форма капчи



РЕГИСТРАЦИЯ

Зарегистрироваться

Рисунок 5. Форма регистрации

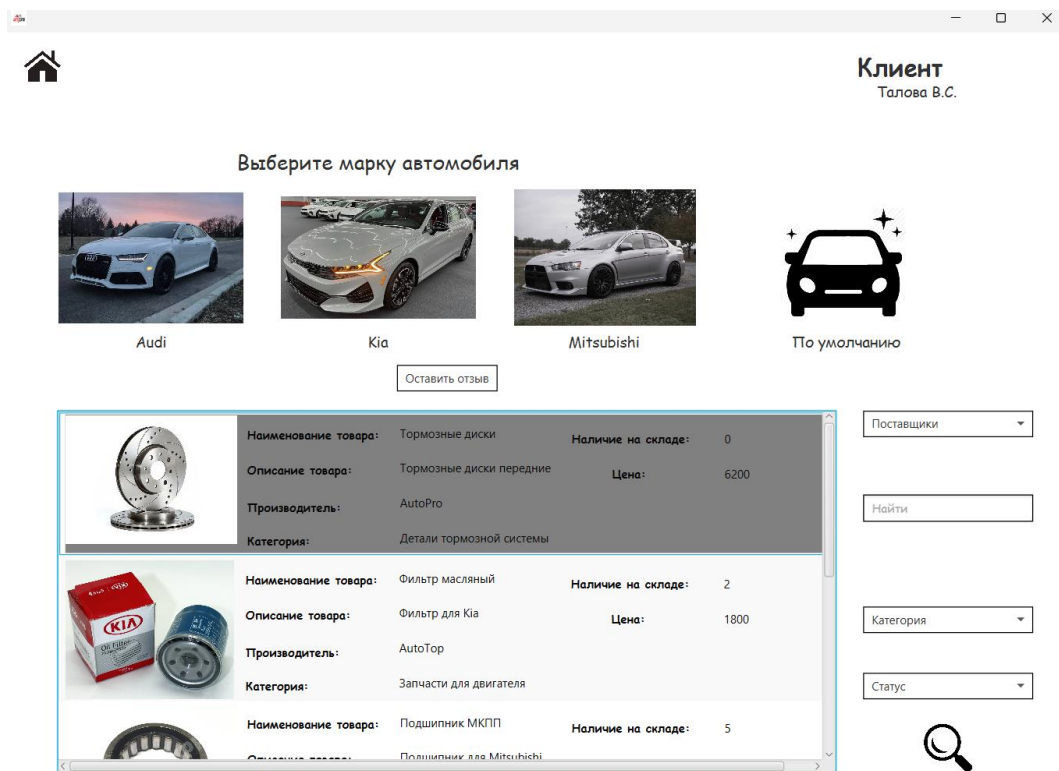


Рисунок 6. Окно личного кабинета клиента после успешной авторизации

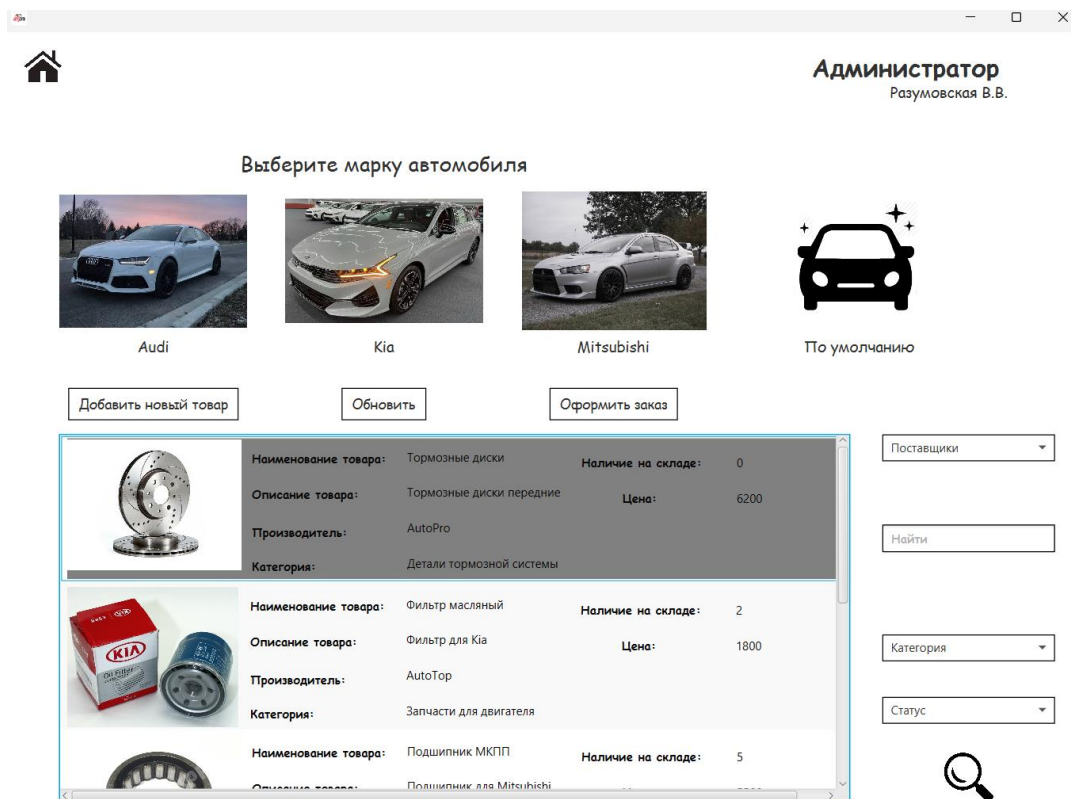


Рисунок 7. Окно личного кабинета сотрудника после успешной авторизации

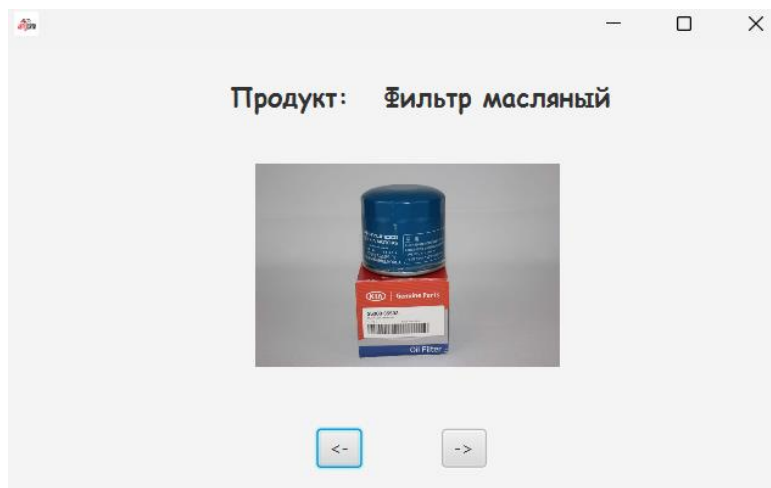
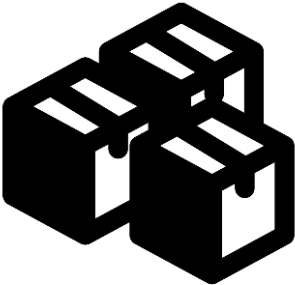


Рисунок 8. Окно дополнительных фото

A screenshot of a web application window titled "Ваши отзывы помогают нам становиться лучше!". The form contains a dropdown menu labeled "Продукт", a text input field labeled "Оценка (от 0 до 5)", a larger text area labeled "Комментарий", and a button labeled "Отправить". The window has standard OS controls in the top right corner.


Рисунок 9. Окно для создания отзыва



Добавить товара

Добавить

Рисунок 10. Окно добавления товара



Редактирование товара

Тормозные диски передни

<

>

Редактировать

Рисунок 11. Окно редактирования товара

Оформление заказа

Выберите клиента ▾

Выберите продукт ▾

Кол-во

Выберите пункт выдачи ▾

Добавить в заказ

Продукт	Кол-во	Цена
---------	--------	------

Оформить

Цена заказа: 0.0

Очистить

Рисунок 12. Окно оформления заказа

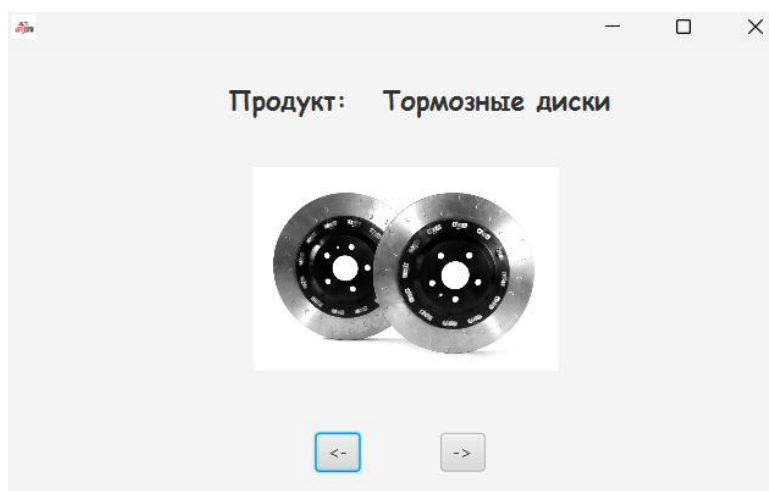


Рисунок 13. Окно просмотра дополнительных фото

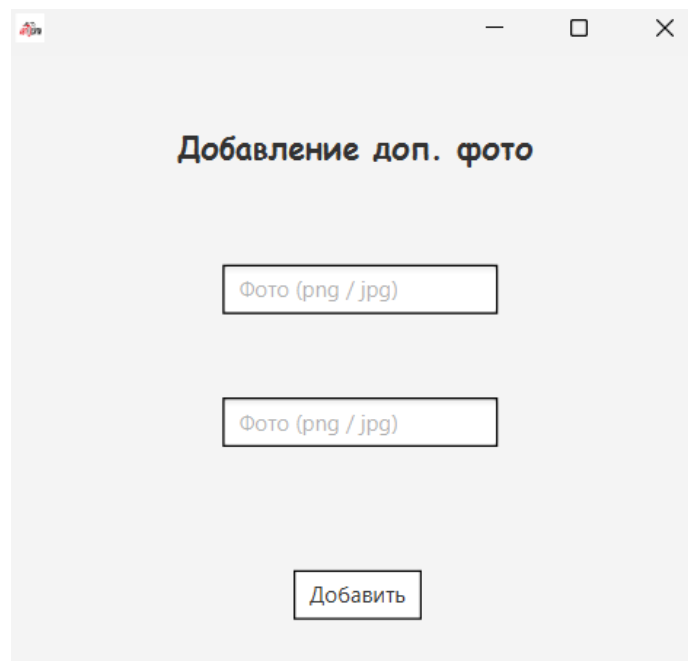
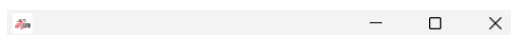


Рисунок 14. Окно добавления дополнительных фото



АВТОРИЗАЦИЯ

Войти

Нет аккаунта?

Зарегистрироваться

Рисунок 15. Ввод данных для авторизации

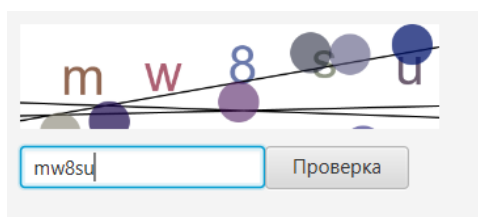
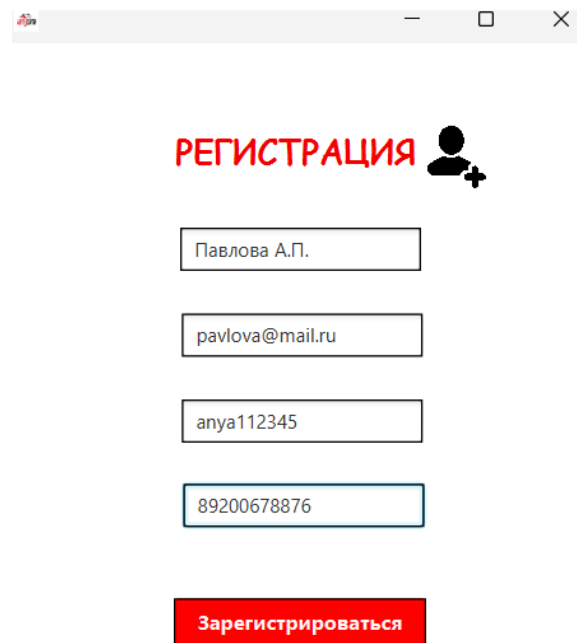



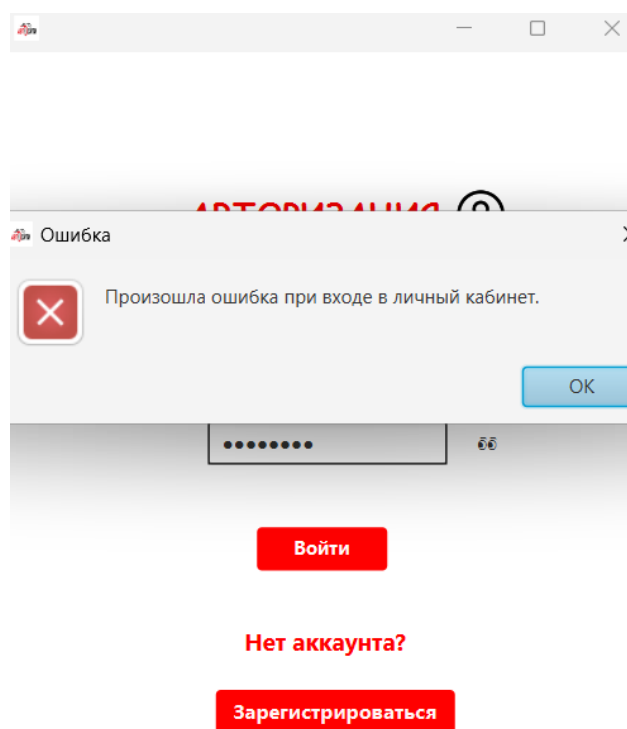
Рисунок 16. Ввод капчи




РЕГИСТРАЦИЯ 

Зарегистрироваться

Рисунок 17. Ввод данных для регистрации



Ошибка

 Произошла ошибка при входе в личный кабинет.

OK

Войти

Нет аккаунта?

Зарегистрироваться

Рисунок 18. Ошибка в случае некорректных данных

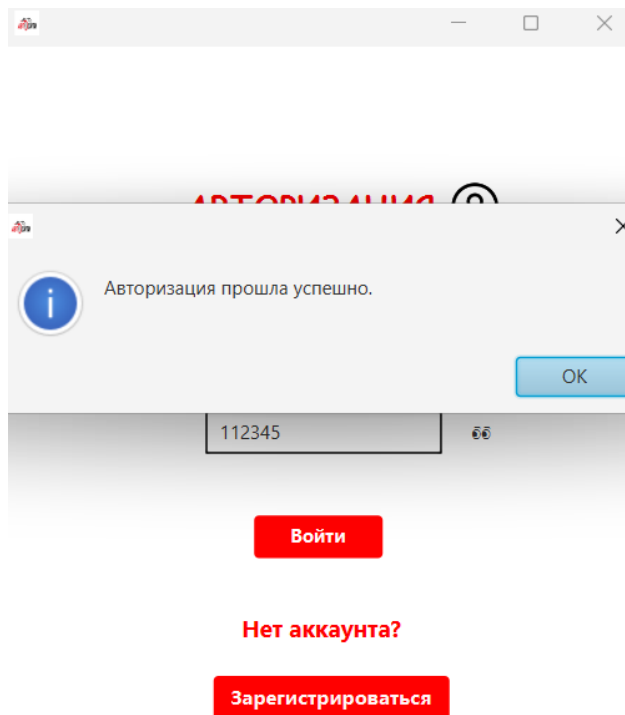


Рисунок 19. Успешная авторизация

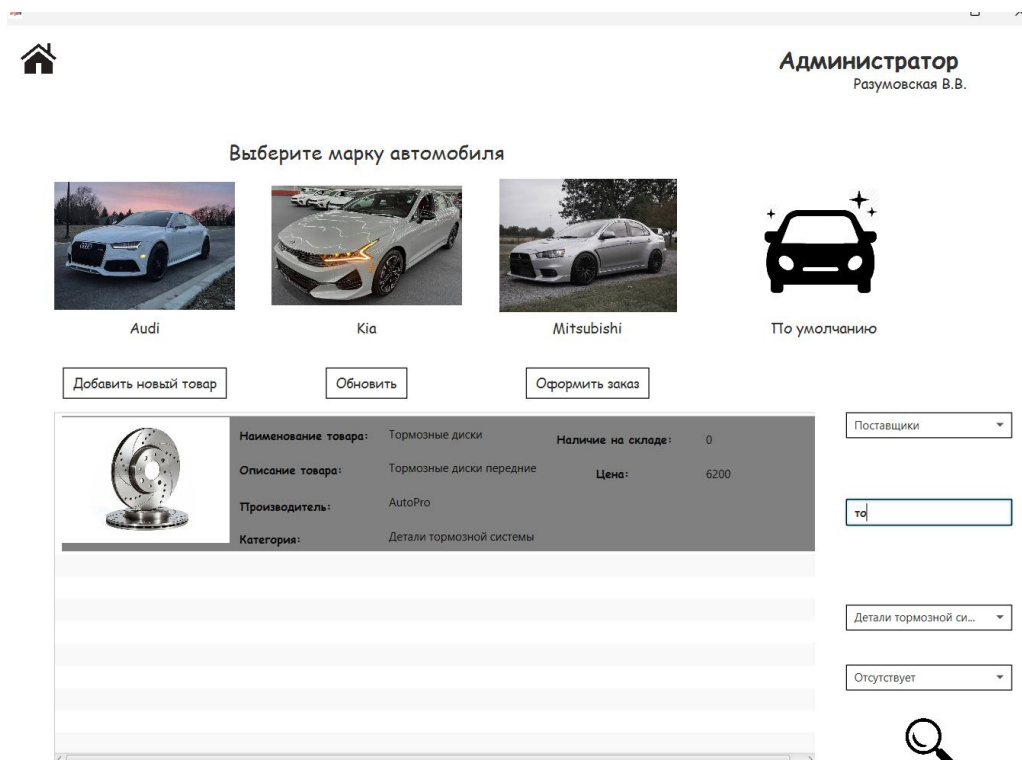


Рисунок 20. Сортировка товаров в аккаунте сотрудника

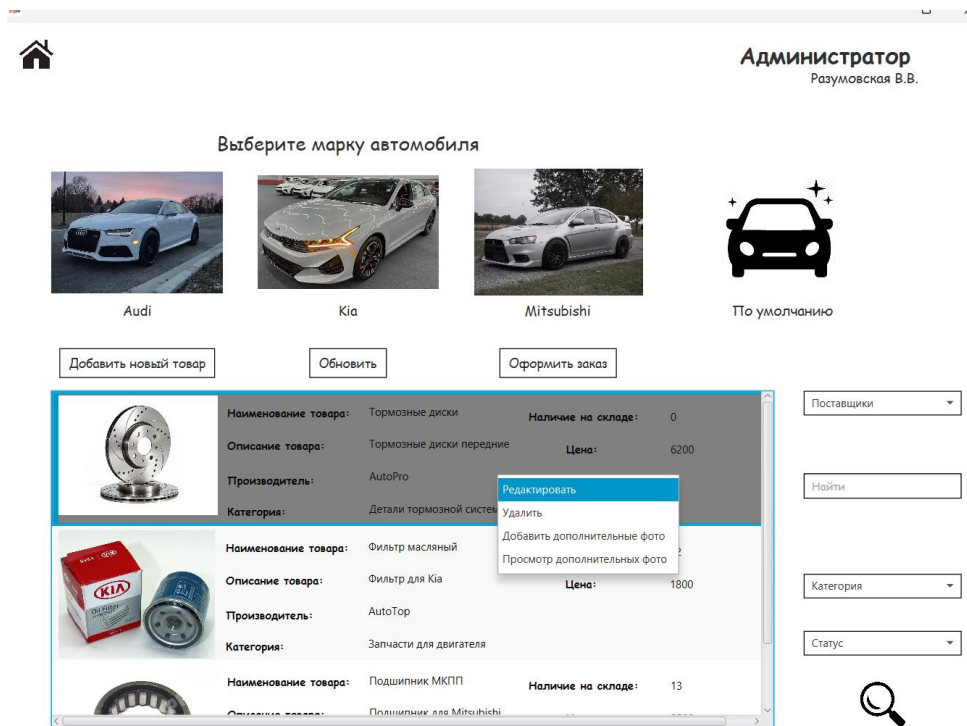


Рисунок 21. Контекстное меню в аккаунте сотрудника



Рисунок 22. Добавление товара в аккаунте сотрудника

Редактирование товара

Сальники

Сальники ВАЗ полуоси

270

70

sal.jpg

Детали трансмисс...

AutoTop

Mitsubishi

Редактировать

Рисунок 23. Редактирование товара в аккаунте сотрудника

Добавление доп. фото

name1.png

name2.jpg

Добавить

Рисунок 24. Добавление дополнительных фото в аккаунте сотрудника

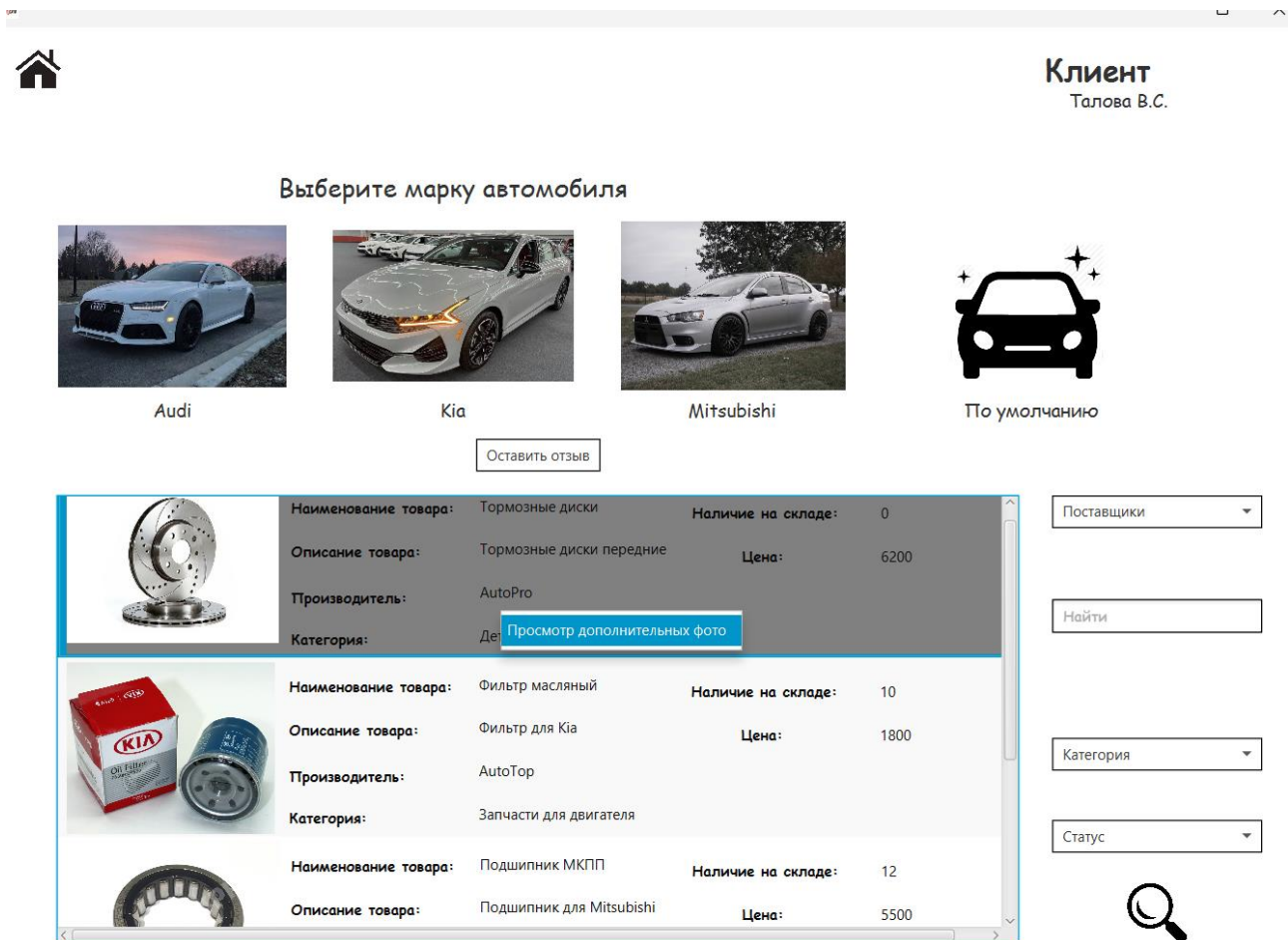


Рисунок 27. Контекстное меню в аккаунте клиента

The screenshot shows a review form titled "Ваши отзывы помогают нам становиться лучше!" (Your reviews help us become better!). The form includes a dropdown menu for product selection, currently showing "Подшипник МКПП" (Clutch bearing). Below this is a rating input field with the value "5". A text area for the review contains the text "Отличный продукт!" (Excellent product!). At the bottom of the form is an "Отправить" (Send) button.

Рисунок 28. Создание отзыва в аккаунте клиента