

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 компьютерной техники»

Выполнил студент группы

ЗИСиП-22-2

Пронин Дмитрий Алексеевич

Проверил преподаватель
Гутянская Е.М.

Проект защищен с оценкой

Дата защиты _____

Подпись

Нижний Новгород, 2024

Оглавление

КУРСОВАЯ РАБОТА.....	1
Оглавление.....	2
Введение.....	3
1. Анализ предметной области.....	4
2. Проектирование базы данных информационной системы.....	6
2.1 Информационно-логическая модель базы данных.....	6
2.2 Словарь данных.....	6
2.3 Ограничения ссылочной целостности.....	10
2.4 Обоснование выбора СУБД.....	12
2.5 Триггеры и хранимые процедуры.....	13
3. Разработка информационной системы.....	15
3.1 Описание структуры системы.....	15
3.2 Разработка класса – точки входа.....	17
3.3 Хранение общей информации.....	18
3.4 Модели.....	22
3.5 Разработка класса подключения к базе данных.....	42
3.6 Разработка формы авторизации.....	80
3.7 Разработка формы регистрации.....	82
4. Пользовательское руководство.....	85
4.1 Страница авторизации.....	85
4.2 Страница регистрации.....	85
4.3 Главная страница.....	86
4.4 Страница каталога товаров.....	86
4.5 Страница информации о товаре.....	87
4.6 Страница фильтров.....	87
4.7 Страница корзины товаров.....	88
4.8 Страница заказов.....	88
4.9 Админ панель.....	89
5. Заключение.....	90
6. Список литературы.....	91

Введение

В современном мире компьютерная техника занимает важное место в жизни пользователей и организаций. С каждым годом увеличивается спрос на компьютеры, что ставит перед магазинами задачу эффективного управления товарными запасами, учётом продуктов и обслуживанием клиентов.

Ключевым элементом для достижения этих целей является создание и функционирование базы данных, специально разработанной для магазина компьютерной техники.

Целью данной курсовой работы является проектирование и разработка базы данных и информационной системы для магазина компьютерной техники. Это позволит оптимизировать управление складскими операциями, автоматизировать процессы учёта товаров и обслуживания клиентов, а также повысить общую эффективность работы магазина.

Задачи работы:

- Анализ требований и потребностей магазина компьютерной техники.
- Проектирование структуры базы данных с учётом специфики магазина.
- Разработка функциональной части информационной системы для автоматизации бизнес-процессов.
- Реализация и интеграция базы данных и информационной системы.
- Оценка эффективности использования базы данных и информационной системы в работе магазина.

Ожидаемый результат работы:

Готовая функционирующая база данных и информационная система для магазина компьютерной техники. В процессе выполнения данной курсовой работы будут применены современные методы и инструменты проектирования баз данных и информационных систем, что позволит создать удобную, надёжную и эффективную систему управления информацией для магазина компьютерной техники. Исследование лучших практик в области проектирования баз данных поможет учесть все необходимые функции и требования, обеспечивая оптимальное использование ресурсов и повышая конкурентоспособность магазина на рынке.

1. Анализ предметной области

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

- Многопользовательский доступ
- Разграничение функционала по ролям
- Клиент и сотрудник магазина могут осуществлять следующие действия:
 - получать оперативную информацию о наличии, описании, фото и стоимости техники;
 - подбирать технику по заявленным критериям;
 - осуществлять отбор техники по категориям, производителям;
 - находить необходимую информацию, содержащую описание техники и комплектующих;
- Сотрудник магазина может осуществлять следующие действия:
 - вести справочники (добавление, удаление, редактирование);
 - оформлять заказ на товар (в одном заказе может быть несколько различных товаров в разном количестве);
 - стоимость заказа рассчитывается динамически.

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

- Пользователи. В этой таблице хранится информация о пользователях системы, включая их имя пользователя, адрес электронной почты, имя, фамилию, пароль и роль.
- Процессоры. В этой таблице хранится информация о процессорах, включая их название, бренд, количество ядер и потоков, базовую и турбочастоты.
- Видеокарты. В этой таблице хранится информация о графических картах, включая название, бренд, объем памяти и тип памяти.
- Блоки питания. В этой таблице хранится информация о блоках питания, включая название, бренд, мощность и рейтинг эффективности.
- Материнские платы. В этой таблице хранится информация о материнских платах, включая название, бренд, тип сокета и форм-фактор.
- Охладители. В этой таблице хранится информация об охладителях, включая название, бренд, тип и мощность охлаждения.
- Корпуса. В этой таблице хранится информация о корпусах компьютеров, включая название, бренд, форм-фактор и цвет.
- Оперативная память (RAM). В этой таблице хранится информация об оперативной памяти, включая название, бренд, объем и скорость.
- Компьютеры. В этой таблице хранится информация обо всех компьютерах, доступных для продажи, включая описание, цену и ссылки на компоненты (процессор, видеокарта и т.д.).

- Корзина покупок. В этой таблице хранится информация о товарах в корзине пользователей, включая владельца корзины и количество каждого компьютера.
- Заказы. В этой таблице хранится информация о заказах клиентов, включая дату заказа, общую сумму и статус заказа.
- Элементы заказа. В этой таблице хранится информация о конкретных товарах в каждом заказе, включая количество и цену каждого товара.

Рассмотрим функции, которые будут реализованы в нашем приложении:

- Многопользовательский доступ:
Это возможность одновременного доступа нескольких пользователей к сервису.
- 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 только для сотрудника магазина:
 - Получение любой информации о товаре;
 - Ведение справочников (добавление, удаление, редактирование);
 - Оформление заказа за пользователя (доступно поле комментария при оформлении заказа, в которое можно указать контактную информацию о заказчике);
- Функции, доступные для всех:
 - Получение информации о названии, описании, стоимости компьютера;
 - Получение информации о комплектующих, установленных в ПК;
 - Осуществлять фильтрацию продуктов в каталоге;
 - Составление своей корзины покупок;
 - Просмотр информации о всех своих заказах

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.

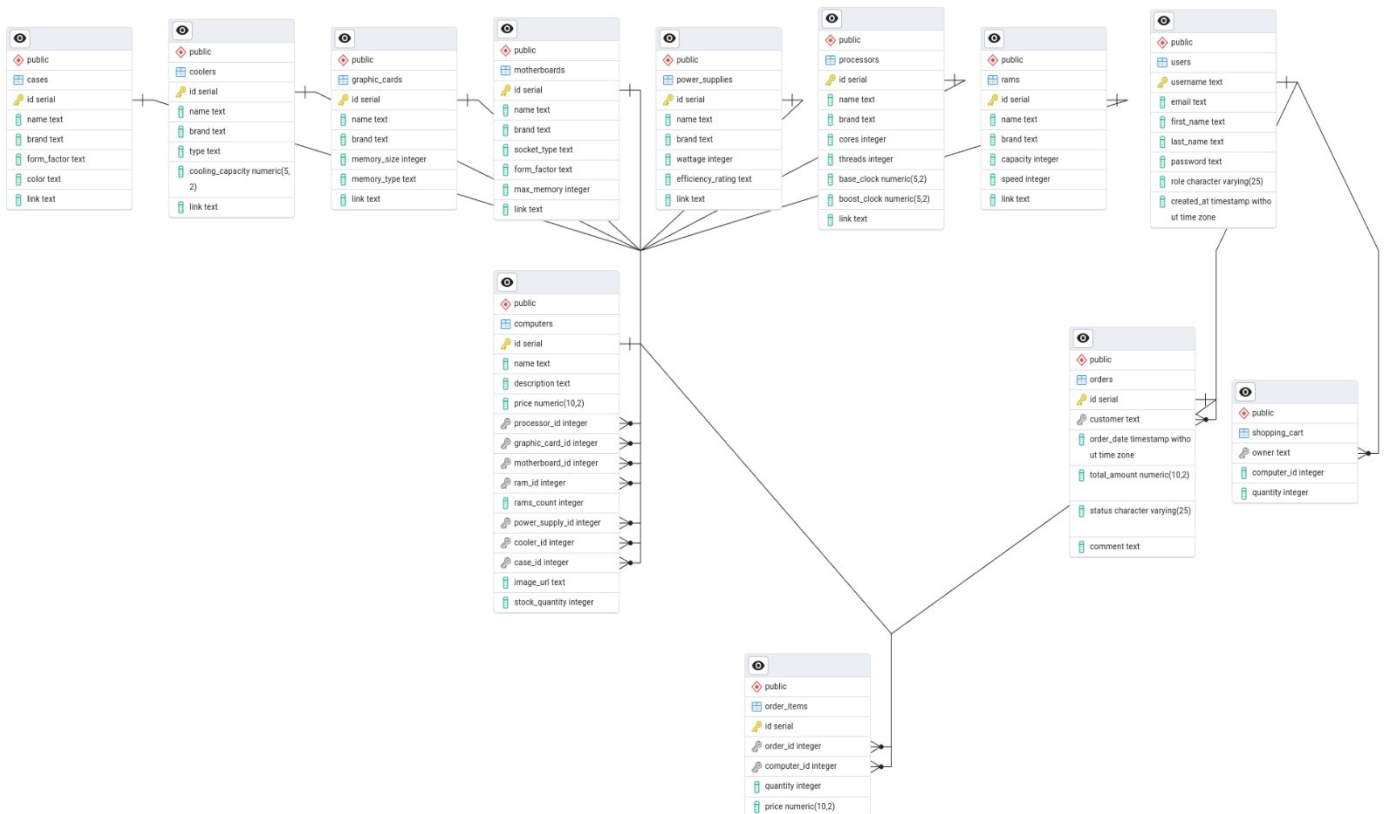


Рисунок 1. Информационно-логическая модель базы данных

2.2 Словарь данных

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

users			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
username	Имя пользователя	TEXT	PK
email	Электронная почта	TEXT	
first_name	Имя	TEXT	
last_name	Фамилия	TEXT	

users			
password	Пароль	TEXT	
role	Роль	VARCHAR(25)	
created_at	Дата создания	TIMESTAMP	

processors			
Имя атрибута	Значение атрибута	Тип данных	Ключ
id	Идентификатор	SERIAL	PK
name	Название	TEXT	
brand	Бренд	TEXT	
cores	Количество ядер	INT	
threads	Количество потоков	INT	
base_clock	Базовая частота	DECIMAL(5,2)	
boost_clock	Турбочастота	DECIMAL(5,2)	
link	Ссылка	TEXT	

graphic_cards			
Имя атрибута	Значение атрибута	Тип данных	Ключ
id	Идентификатор	SERIAL	PK
name	Название	TEXT	
brand	Бренд	TEXT	
memory_size	Объем памяти	INT	
memory_type	Тип памяти	TEXT	
link	Ссылка	TEXT	

power_supplies			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id	Идентификатор	SERIAL	PK
name	Название	TEXT	
brand	Бренд	TEXT	
wattage	Мощность	INT	
efficiency_rating	Рейтинг эффективности	TEXT	
link	Ссылка	TEXT	

Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id	Идентификатор	SERIAL	PK
name	Название	TEXT	

brand	Бренд	TEXT	
socket_type	Тип сокета	TEXT	
form_factor	Форм-фактор	TEXT	
max_memory	Максимальный объем памяти	INT	
link	Ссылка	TEXT	

coolers			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id	Идентификатор	SERIAL	PK
name	Название	TEXT	
brand	Бренд	TEXT	
type	Тип охладителя	TEXT	
cooling_capacity	Мощность охлаждения	DECIMAL(5,2)	
link	Ссылка	TEXT	

cases			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id	Идентификатор	SERIAL	PK
name	Название	TEXT	
brand	Бренд	TEXT	
form_factor	Форм-фактор	TEXT	
color	Цвет	TEXT	
link	Ссылка	TEXT	

rams			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id	Идентификатор	SERIAL	PK
name	Название	TEXT	
brand	Бренд	TEXT	
capacity	Объем памяти в ГБ	INT	
speed	Скорость в МГц	INT	
link	Ссылка	TEXT	

computers			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id	Идентификатор	SERIAL	PK
name	Название	TEXT	
description	Описание	TEXT	
price	Цена	DECIMAL(10,2)	
processor_id	Идентификатор процессора	INT NOT NULL REFERENCES processors(id)	FK
graphic_card_id	Идентификатор видеокарты	INT NOT NULL REFERENCES graphic_cards(id)	FK
motherboard_id	Идентификатор материнской платы	INT NOT NULL REFERENCES motherboards(id)	FK
ram_id	Идентификатор оперативной памяти	INT NOT NULL REFERENCES rams(id)	FK
rams_count	Количество модулей оперативной памяти	INT NOT NULL	
power_supply_id	Идентификатор блока питания	INT NOT NULL REFERENCES power_supplies(id)	FK
cooler_id	Идентификатор охладителя	INT NOT NULL REFERENCES coolers(id)	FK
case_id	Идентификатор корпуса	INT NOT NULL REFERENCES cases(id)	FK
image_url	URL изображения	TEXT	
stock_quantity	Количество на складе	INT NOT NULL DEFAULT 0	

shopping_cart			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
owner	Владелец корзины (username)	TEXT NOT NULL REFERENCES users(username)	FK
computer_id	Идентификатор компьютера	INT NOT NULL REFERENCES computers(id)	FK
quantity	Количество товара в корзине	INT NOT NULL DEFAULT 1	

orders			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id	Идентификатор заказа	SERIAL	PK
customer	Клиент (username)	TEXT NOT NULL REFERENCES users(username)	FK
order_date	Дата заказа	TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP	
total_amount	Общая сумма заказа	DECIMAL(10,2) NOT NULL	
status	Статус заказа	VARCHAR(25) NOT NULL DEFAULT 'pending'	
comment	Комментарий к заказу	TEXT	

order_items			
Имя атрибута	Значение атрибута	Тип данных	Ключ (ссылка)
id	Идентификатор элемента заказа	SERIAL	PK
order_id	Идентификатор заказа	INT NOT NULL REFERENCES orders(id)	FK
computer_id	Идентификатор компьютера	INT NOT NULL REFERENCES computers(id)	FK
quantity	Количество товара в заказе	INT NOT NULL	
price	Цена товара на момент заказа	DECIMAL(10,2) NOT NULL	

2.3 Ограничения ссылочной целостности

- Ограничение на таблице «users»:** Идентификатор пользователя (username) является первичным ключом. Это ограничение гарантирует, что каждый пользователь будет иметь уникальное имя пользователя, что позволяет избежать дублирования записей и обеспечивает целостность данных в системе.
- Ограничение на таблице «processors»:** Идентификатор процессора (id) является первичным ключом. Это ограничение обеспечивает уникальность каждой записи о процессоре и позволяет точно идентифицировать каждый процессор в базе данных.
- Ограничение на таблице «graphic_cards»:** Идентификатор видеокарты (id) является первичным ключом. Это ограничение гарантирует, что каждая видеокарта будет иметь уникальный идентификатор, что позволяет точно отслеживать информацию о каждой видеокарте.

4. **Ограничение на таблице «power_supplies»:** Идентификатор блока питания (id) является первичным ключом. Это ограничение обеспечивает уникальность каждой записи о блоке питания и позволяет избежать путаницы между различными моделями блоков питания.
5. **Ограничение на таблице «motherboards»:** Идентификатор материнской платы (id) является первичным ключом. Это ограничение гарантирует, что каждая материнская плата будет иметь уникальный идентификатор, что позволяет точно отслеживать информацию о каждой материнской плате.
6. **Ограничение на таблице «coolers»:** Идентификатор охладителя (id) является первичным ключом. Это ограничение обеспечивает уникальность каждой записи об охладителе и позволяет точно идентифицировать каждый охладитель в базе данных.
7. **Ограничение на таблице «cases»:** Идентификатор корпуса (id) является первичным ключом. Это ограничение гарантирует, что каждый корпус будет иметь уникальный идентификатор, что позволяет избежать дублирования записей и обеспечивает целостность данных.
8. **Ограничение на таблице «rams»:** Идентификатор оперативной памяти (id) является первичным ключом. Это ограничение обеспечивает уникальность каждой записи об оперативной памяти и позволяет точно отслеживать информацию о каждом модуле RAM.

9. Ограничение на таблице «computers»:

- Идентификатор процессора (processor_id) является внешним ключом, связанным с таблицей «processors» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительный процессор, представленный в таблице «processors». Такая связь помогает контролировать совместимость компонентов.
- Идентификатор видеокарты (graphic_card_id) является внешним ключом, связанным с таблицей «graphic_cards» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительную видеокарту, представленную в таблице «graphic_cards».
- Идентификатор материнской платы (motherboard_id) является внешним ключом, связанным с таблицей «motherboards» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительную материнскую плату.
- Идентификатор оперативной памяти (ram_id) является внешним ключом, связанным с таблицей «rams» по полю идентификатора (id). Это ограничение гарантирует, что каждый компьютер будет иметь действительную оперативную память.
- Идентификатор блока питания (power_supply_id) является внешним ключом, связанным с таблицей «power_supplies» по полю идентификатора (id). Это

ограничение гарантирует, что каждый компьютер будет иметь действительный блок питания.

- Идентификатор охладителя (`cooler_id`) является внешним ключом, связанным с таблицей «coolers» по полю идентификатора (`id`). Это ограничение гарантирует, что каждый компьютер будет иметь действительное охлаждение.
- Идентификатор корпуса (`case_id`) является внешним ключом, связанным с таблицей «cases» по полю идентификатора (`id`). Это ограничение гарантирует, что каждый компьютер будет иметь действительный корпус.

10. Ограничение на таблице «shopping_cart»: Идентификатор владельца корзины (`owner`) является внешним ключом, связанным с таблицей «users» по полю идентификатора пользователя (`username`). Это ограничение гарантирует, что каждая корзина принадлежит действительному пользователю. Идентификатор компьютера (`computer_id`) является внешним ключом, связанным с таблицей «computers» по полю идентификатора (`id`). Это ограничение гарантирует, что каждая запись в корзине содержит действительный компьютер.

11. Ограничение на таблице «orders»: Идентификатор клиента (`customer`) является внешним ключом, связанным с таблицей «users» по полю идентификатора пользователя (`username`). Это ограничение гарантирует, что каждый заказ принадлежит действительному клиенту.

12. Ограничение на таблице «order_items»: Идентификатор заказа (`order_id`) является внешним ключом, связанным с таблицей «orders» по полю идентификатора заказа (`id`). Это ограничение гарантирует, что каждый элемент заказа относится к действительному заказу. Идентификатор компьютера (`computer_id`) является внешним ключом, связанным с таблицей «computers» по полю идентификатора компьютера (`id`). Это ограничение гарантирует, что каждый элемент заказа содержит действительный компьютер.

2.4 Обоснование выбора СУБД

Система управления базами данных (СУБД) – это набор программ, которые управляют структурой БД и контролируют доступ к данным, хранящимся в ней. СУБД служит посредником между пользователем и БД. В качестве СУБД для моей информационной системы я выбрал PostgreSQL. Это решение обусловлено рядом преимуществ, которые PostgreSQL предлагает по сравнению с другими СУБД:

- 1. Расширяемость:** PostgreSQL поддерживает пользовательские типы данных, функции и операторы, что позволяет адаптировать систему под специфические требования приложения. Это делает PostgreSQL особенно подходящим для сложных и нестандартных проектов.

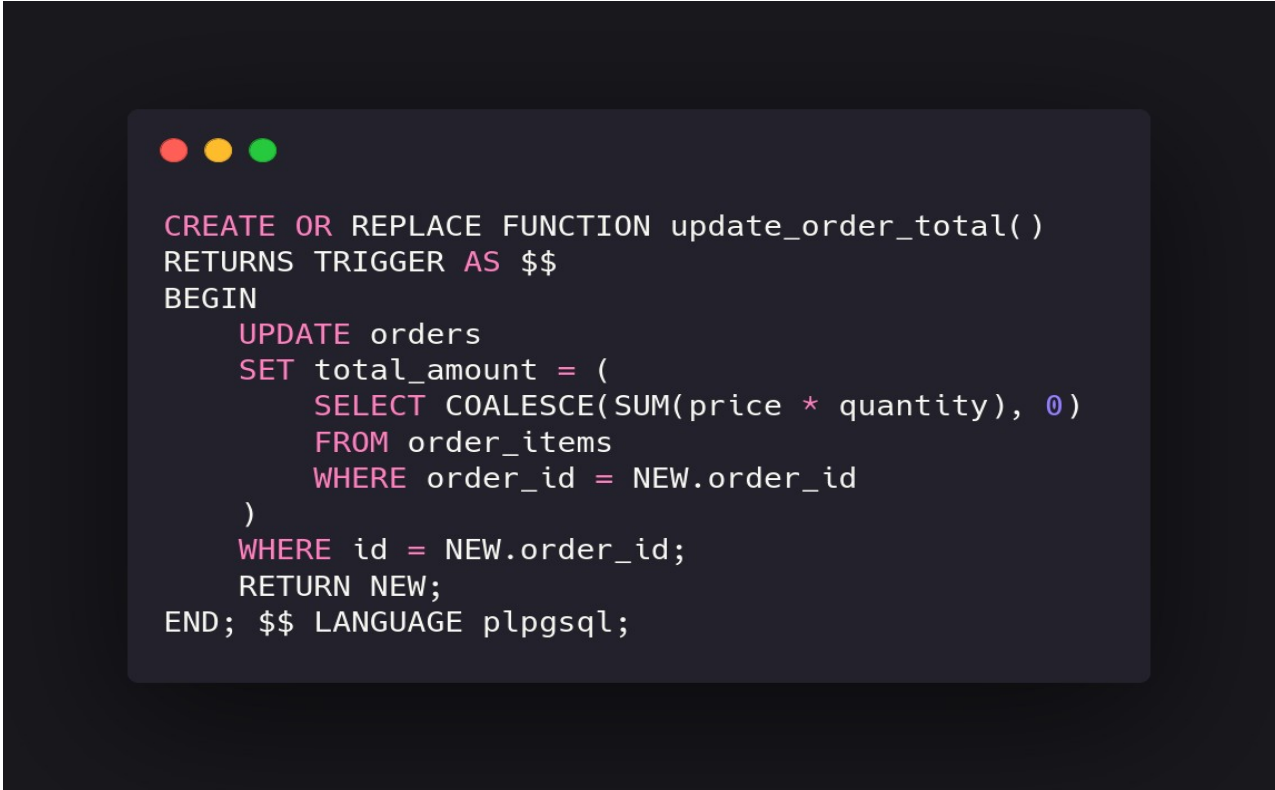
2. **Поддержка стандартов SQL:** PostgreSQL строго следует стандартам SQL, что обеспечивает совместимость с другими системами и упрощает миграцию данных. Это также облегчает обучение пользователей, знакомых с SQL.
3. **Высокая производительность:** PostgreSQL использует эффективные механизмы индексации и оптимизации запросов, что обеспечивает высокую производительность при работе с большими объемами данных и сложными запросами.
4. **Надежность и целостность данных:** PostgreSQL поддерживает ACID-транзакции (атомарность, согласованность, изолированность, долговечность), что гарантирует надежность и целостность данных. Это особенно важно для критически важных приложений.
5. **Многоуровневая безопасность:** PostgreSQL предлагает расширенные функции безопасности, включая аутентификацию на уровне базы данных, шифрование данных и управление правами доступа на уровне строк и столбцов. Это обеспечивает защиту данных от несанкционированного доступа.
6. **Активное сообщество и поддержка:** PostgreSQL имеет большое и активное сообщество разработчиков и пользователей, что обеспечивает доступ к обширной документации, руководствам и онлайн-ресурсам. Пользователи могут легко получать поддержку от опытных специалистов.
7. **Частые обновления и новые функции:** PostgreSQL активно разрабатывается и регулярно получает обновления, которые включают новые функции, исправления ошибок и улучшения производительности. Это гарантирует, что система будет оставаться актуальной и безопасной.
8. **Личный опыт использования:** Я постоянно использую PostgreSQL в своих проектах, так как он мне хорошо знаком. Его удобный интерфейс и многофункциональность позволяют эффективно решать задачи различной сложности. Например, поддержка массивов и JSON-данных предоставляет возможности для работы с более сложными структурами данных, которые отсутствуют в других СУБД. Это делает его идеальным выбором для разработки современных приложений.

2.5 Триггеры и хранимые процедуры

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

- Хранимая функция для обновления общей суммы заказа
Хранимая функция **update_order_total** предназначена для автоматического обновления общей суммы заказа в таблице «**orders**». Она вычисляет новую общую сумму заказа, суммируя стоимость всех элементов заказа из таблицы

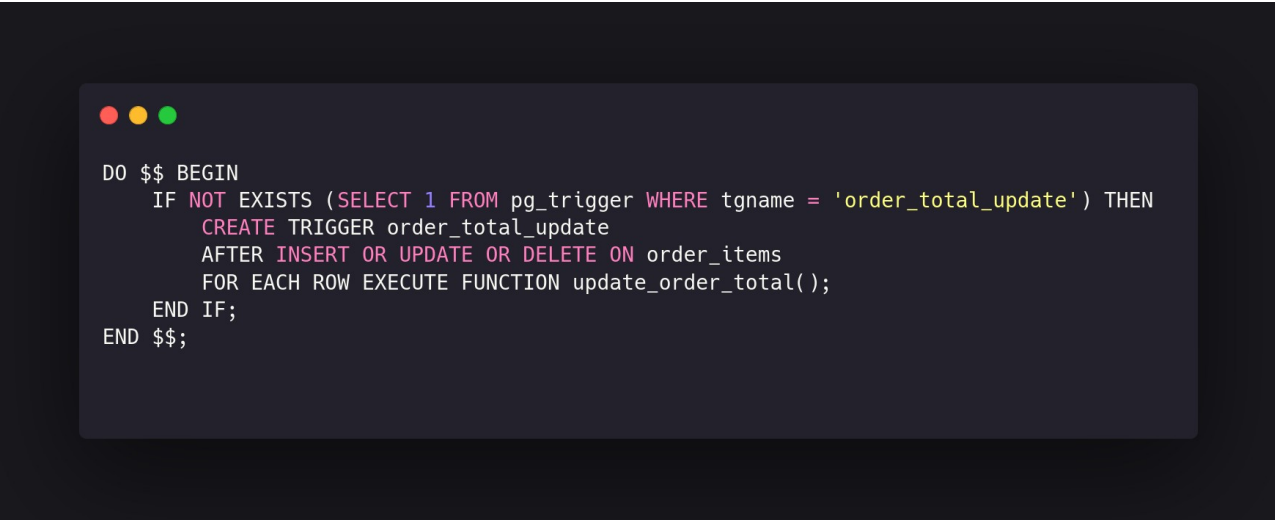
«**order_items**» и обновляет соответствующую запись в таблице «**orders**». Функция будет вызываться триггером после вставки, обновления или удаления элементов заказа.



```
CREATE OR REPLACE FUNCTION update_order_total()  
RETURNS TRIGGER AS $$  
BEGIN  
    UPDATE orders  
    SET total_amount = (  
        SELECT COALESCE(SUM(price * quantity), 0)  
        FROM order_items  
        WHERE order_id = NEW.order_id  
    )  
    WHERE id = NEW.order_id;  
    RETURN NEW;  
END; $$ LANGUAGE plpgsql;
```

Рисунок 2. Функция **update_order_total**

- Триггер для обновления общей суммы заказа
Триггер **order_total_update** будет активироваться после каждой вставки, обновления или удаления записи в таблице «**order_items**». Он будет вызывать функцию **update_order_total**, чтобы обеспечить автоматическое обновление общей суммы заказа при любых изменениях в связанных элементах заказа, что позволяет поддерживать целостность данных и актуальность информации о заказах.



```
DO $$ BEGIN  
    IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname = 'order_total_update') THEN  
        CREATE TRIGGER order_total_update  
        AFTER INSERT OR UPDATE OR DELETE ON order_items  
        FOR EACH ROW EXECUTE FUNCTION update_order_total();  
    END IF;  
END $$;
```

Рисунок 3. Триггер **order_total_update**

3. Разработка информационной системы

3.1 Описание структуры системы

В рамках данного проекта была разработана информационная система, представляющая собой приложение, предназначенное для функционирования в качестве магазина компьютерной техники. Данная система состоит из восьми взаимосвязанных страниц, каждая из которых выполняет свою уникальную функцию и обеспечивает пользователю доступ к различным аспектам работы с приложением. Основные страницы приложения включают:

- Регистрация – страница, на которой пользователи могут создать новый аккаунт, заполнив необходимые поля с личной информацией.
- Авторизация – интерфейс для входа в систему, где пользователи вводят свои учетные данные для доступа к функционалу приложения.
- Каталог товаров – основная страница, на которой представлены все доступные товары, сгруппированные по категориям.
- Информация о товаре – страница, предоставляющая детальную информацию о выбранном товаре, включая его характеристики и цену.
- Фильтры – инструмент для сортировки и фильтрации товаров по различным критериям, таким как цена, производитель и технические характеристики.
- Корзина товаров – страница, на которой пользователи могут просмотреть добавленные в корзину товары перед оформлением заказа.
- Заказы – интерфейс для просмотра истории заказов пользователя и их статусов.
- Админ панель – специализированный интерфейс для администраторов, позволяющий управлять данными в системе.

Административная панель включает в себя дополнительные одиннадцать страниц, каждая из которых соответствует отдельной таблице базы данных. Эти таблицы содержат информацию о различных аспектах работы магазина и включают:

- **users** – таблица с данными пользователей приложения.
- **processors** – информация о процессорах, доступных в магазине.
- **graphic_cards** – данные о видеокартах.
- **motherboards** – сведения о материнских платах.
- **power_supplies** – информация о блоках питания.
- **rams** – данные об оперативной памяти.

- **coolers** – информация о системах охлаждения.
- **cases** – сведения о корпусах компьютеров.
- **computers** – таблица с информацией о готовых сборках компьютеров.
- **orders** – данные о заказах пользователей.
- **order_items** – информация о товарах в каждом заказе.

Переходы между страницами приложения реализованы с помощью кнопок и контекстного меню, что обеспечивает удобство навигации для пользователей.

При запуске приложения пользователь сразу попадает на страницу авторизации. Здесь он может пройти процедуру идентификации, введя свои учетные данные. В случае отсутствия аккаунта имеется возможность перейти к регистрации нового пользователя.

После успешного ввода данных и нажатия на кнопку **"Login"**, пользователь автоматически перенаправляется на основной интерфейс приложения, где ему становится доступен полный функционал системы.

На главной странице пользователя встречается меню каталога товаров. В этом меню представлено обычное перечисление компьютеров с карточками товаров, включающими фотографии, названия и описания.

Каждая карточка содержит информацию о цене и кнопку **"Add to cart"**, нажатие на которую позволяет добавить выбранный компьютер в корзину. Если пользователь нажимает на область с названием или описанием товара, он попадает на страницу просмотра информации о продукте. На данной странице отображается вся информация из карточки товара с добавлением детальных характеристик компьютера, таких как процессор, видеокарта, материнская плата, блок питания, оперативная память, кулер и корпус.

Страница фильтров предоставляет пользователю возможность отфильтровать каталог товаров по необходимым комплектующим или интересующим ценовым диапазонам.

Страница **"Cart" (корзина товаров)** аналогична каталогу товаров, но с дополнительными функциями. Здесь пользователь видит количество товара на складе и виджет для изменения количества выбранного компьютера в заказе.

Также присутствует кнопка для удаления товара из корзины. В верхней части списка отображается итоговая стоимость заказа и кнопка **"Make an order" (сделать заказ)**.

Если пользователь имеет роль администратора, то данная страница дополнительно включает текстовое поле для указания контактов пользователя, который сделал данный заказ. Это может быть полезно в случаях, когда клиент связывается с магазином по телефону и предоставляет информацию для оформления заказа.

После завершения оформления заказа пользователь перенаправляется на страницу **"Orders"(заказы)**, где он может ознакомиться с подробной информацией обо всех совершенных заказах. Доступная информация включает:

- дату оформления заказа
- итоговую цену заказа
- статус выполнения заказа

- краткие сведения о компьютерах, приобретенных в конкретном заказе (цены указаны на момент покупки).

Если у пользователя приложения установлен статус администратора, он получает доступ к кнопке "admin panel", которая открывает контекстное меню с выбором таблицы для редактирования. После выбора соответствующей таблицы открывается страница, на которой отображается таблица с данными во весь экран. В верхней части страницы расположены несколько кнопок управления таблицей:

- **Add row** – добавление новой строки в таблицу,
- **Delete row** – удаление выделенной строки,
- **Save changes** – сохранение всех внесенных изменений в таблице,
- **Refresh** – перезагрузка страницы с обновлением актуальных данных из базы данных.

3.2 Разработка класса – точки входа

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

```
...
package com.shop;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

import java.io.IOException;

import com.shop.database.DbConnection;

/**
 * JavaFX App
 */
public class App extends Application {

    private static Scene scene;

    @Override
    public void start(@SuppressWarnings("exports") Stage stage) throws IOException {
        DbConnection dbConnection = DbConnection.getDatabaseConnection();
        dbConnection.createTablesIfNotExists();

        scene = new Scene(loadFXML("auth/LoginView"), 640, 480);
```

```

        stage.setScene(scene);
        stage.initStyle(StageStyle.UNDECORATED);
        stage.show();
        stage.setAlwaysOnTop(true);
    }

    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
        return fxmlLoader.load();
    }

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

```

3.3 Хранение общей информации

Листинг класса SharedData (класс для хранения промежуточной информации, использующейся разными сценами приложения)

```

package com.shop.controllers;

import com.shop.database.models.Case;
import com.shop.database.models.Computer;
import com.shop.database.models.Cooler;
import com.shop.database.models.GraphicCard;
import com.shop.database.models.Motherboard;
import com.shop.database.models.PowerSupply;
import com.shop.database.models.Processor;
import com.shop.database.models.RAM;
import com.shop.database.models.User;

public class SharedData {
    public static User authenticatedUser;
    public static MainPanelController controller;
    public static Computer selectedComputer;
}

```

```

public static Integer minCostFilter = 1;
public static Integer maxCostFilter = 1000000000;
public static Processor processorFilter;
public static GraphicCard graphicCardFilter;
public static Motherboard motherboardFilter;
public static PowerSupply powerSupplyFilter;
public static RAM ramFilter;
public static Integer countRAMFilter = 0;
public static Cooler coolerFilter;
public static Case caseFilter;

public static void setAuthenticatedUser(User value) {
    authenticatedUser = value;
}

public static User getAuthenticatedUser() {
    return authenticatedUser;
}

public static void setMainController(MainPanelController value) {
    controller = value;
}

public static MainPanelController getMainController() {
    return controller;
}

public static void setSelectedComputer(Computer value) {
    selectedComputer = value;
}

public static Computer getSelectedComputer() {
    return selectedComputer;
}

public static Integer getMinCostFilter() {
    return minCostFilter;
}

public static void setMinCostFilter(Integer minCost) {
    minCostFilter = minCost;
}

```

```

public static Integer getMaxCostFilter() {
    return maxCostFilter;
}

public static void setMaxCostFilter(Integer maxCost) {
    maxCostFilter = maxCost;
}

public static Processor getProcessorFilter() {
    return processorFilter;
}

public static void setProcessorFilter(Processor processor) {
    processorFilter = processor;
}

public static GraphicCard getGraphicCardFilter() {
    return graphicCardFilter;
}

public static void setGraphicCardFilter(GraphicCard gpu) {
    graphicCardFilter = gpu;
}

public static Motherboard getMotherboardFilter() {
    return motherboardFilter;
}

public static void setMotherboardFilter(Motherboard motherboard) {
    motherboardFilter = motherboard;
}

public static PowerSupply getPowerSupplyFilter() {
    return powerSupplyFilter;
}

public static void setPowerSupplyFilter(PowerSupply ps) {
    powerSupplyFilter = ps;
}

public static RAM getRamFilter() {
    return ramFilter;
}

```

```

public static void setRamFilter(RAM ram) {
    ramFilter = ram;
}

public static Integer getCountRAMFilter() {
    return countRAMFilter;
}

public static void setCountRAMFilter(Integer countRAM) {
    countRAMFilter = countRAM;
}

public static Cooler getCoolerFilter() {
    return coolerFilter;
}

public static void setCoolerFilter(Cooler cooler) {
    coolerFilter = cooler;
}

public static Case getCaseFilter() {
    return caseFilter;
}

public static void setCaseFilter(Case value) {
    caseFilter = value;
}

public static void resetFilters() {
    processorFilter = null;
    graphicCardFilter = null;
    motherboardFilter = null;
    powerSupplyFilter = null;
    ramFilter = null;
    countRAMFilter = 0;
    coolerFilter = null;
    caseFilter = null;
    minCostFilter = 1;
    maxCostFilter = 1000000000;
}
}

```

3.4 Модели

Листинги классов, олицетворяющих колонки определенных таблиц. (нужны для представления полученных данных в виде объектов, для удобного взаимодействия приложения с данными).

```
...
```

```
package com.shop.database.models;
```

```
public interface Identifiable {
```

```
    Integer getId();
```

```
    String toString();
```

```
}
```

```
...
```

```
...
```

```
package com.shop.database.models;
```

```
import java.sql.Timestamp;
```

```
public class User {
```

```
    private String username;
```

```
    private String new_username;
```

```
    private String email;
```

```
    private String firstName;
```

```
    private String lastName;
```

```
    private String password;
```

```
    private String role;
```

```
    private Timestamp createdAt;
```

```
    public User(String username, String email, String firstName, String lastName, String password, String role) {
```

```
        // Конструктор без createdAt
```

```
        this.username = username;
```

```
        this.new_username = username;
```

```
        this.email = email;
```

```
        this.firstName = firstName;
```

```
        this.lastName = lastName;
```

```
        this.password = password;
```

```
        this.role = role;
```

```
}
```

```

    public User(String username, String email, String firstName, String lastName, String
password, String role, Timestamp createdAt) {
        this.username = username;
        this.new_username = username;
        this.email = email;
        this.firstName = firstName;
        this.lastName = lastName;
        this.password = password;
        this.role = role;
        this.createdAt = createdAt;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.new_username = username;
    }

    public String getNewUsername() {
        return new_username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

```

```

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getRole() {
    return role;
}

public void setRole(String role) {
    this.role = role;
}

public Timestamp getCreatedAt() {
    return createdAt;
}

public void setCreatedAt(Timestamp createdAt) {
    this.createdAt = createdAt;
}
}

...

...

package com.shop.database.models;

import java.math.BigDecimal;

public class Processor implements Identifiable {
    private Integer id;
    private String name;
    private String brand;
    private Integer cores;
    private Integer threads;
    private BigDecimal baseClock; // Using BigDecimal for decimal values
    private BigDecimal boostClock; // Using BigDecimal for decimal values

```



```

private String link;

public Processor(Integer id, String name, String brand, Integer cores, Integer threads,
BigDecimal baseClock, BigDecimal boostClock, String link) {
    this.id = id;
    this.name = name;
    this.brand = brand;
    this.cores = cores;
    this.threads = threads;
    this.baseClock = baseClock;
    this.boostClock = boostClock;
    this.link = link;
}

// Getters and Setters
public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getBrand() {
    return brand;
}

public void setBrand(String brand) {
    this.brand = brand;
}

public Integer getCores() {
    return cores;
}

public void setCores(Integer cores) {

```

```

        this.cores = cores;
    }

    public Integer getThreads() {
        return threads;
    }

    public void setThreads(Integer threads) {
        this.threads = threads;
    }

    public BigDecimal getBaseClock() {
        return baseClock;
    }

    public void setBaseClock(BigDecimal baseClock) {
        this.baseClock = baseClock;
    }

    public BigDecimal getBoostClock() {
        return boostClock;
    }

    public void setBoostClock(BigDecimal boostClock) {
        this.boostClock = boostClock;
    }

    public String getLink() {
        return link;
    }

    public void setLink(String link) {
        this.link = link;
    }

    @Override
    public String toString() {
        return name;
    }
}
...

...

package com.shop.database.models;

```

```

public class GraphicCard implements Identifiable {
    private Integer id;
    private String name;
    private String brand;
    private Integer memorySize;
    private String memoryType;
    private String link;

    public GraphicCard(Integer id, String name, String brand, Integer memorySize, String
memoryType, String link) {
        this.id = id;
        this.name = name;
        this.brand = brand;
        this.memorySize = memorySize;
        this.memoryType = memoryType;
        this.link = link;
    }

    // Getters and Setters
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }
}

```

```

public Integer getMemorySize() {
    return memorySize;
}

public void setMemorySize(Integer memorySize) {
    this.memorySize = memorySize;
}

public String getMemoryType() {
    return memoryType;
}

public void setMemoryType(String memoryType) {
    this.memoryType = memoryType;
}

public String getLink() {
    return link;
}

public void setLink(String link) {
    this.link = link;
}

@Override
public String toString() {
    return name;
}
}
...

...

package com.shop.database.models;

public class Motherboard implements Identifiable {
    private Integer id;
    private String name;
    private String brand;
    private String socketType;
    private String formFactor;
    private Integer maxMemory;
    private String link;

```

```

    public Motherboard(Integer id, String name, String brand, String socketType, String
formFactor, Integer maxMemory, String link) {
        this.id = id;
        this.name = name;
        this.brand = brand;
        this.socketType = socketType;
        this.formFactor = formFactor;
        this.maxMemory = maxMemory;
        this.link = link;
    }

    // Getters and Setters
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public String getSocketType() {
        return socketType;
    }

    public void setSocketType(String socketType) {
        this.socketType = socketType;
    }

```

```

public String getFormFactor() {
    return formFactor;
}

public void setFormFactor(String formFactor) {
    this.formFactor = formFactor;
}

public Integer getMaxMemory() {
    return maxMemory;
}

public void setMaxMemory(Integer maxMemory) {
    this.maxMemory = maxMemory;
}

public String getLink() {
    return link;
}

public void setLink(String link) {
    this.link = link;
}

@Override
public String toString() {
    return name;
}
}
...

...

package com.shop.database.models;

```

```

public class PowerSupply implements Identifiable {
    private Integer id;
    private String name;
    private String brand;
    private Integer wattage;
    private String efficiencyRating;
    private String link;
}

```

```
public PowerSupply(Integer id, String name, String brand, Integer wattage, String
efficiencyRating, String link) {
    this.id = id;
    this.name = name;
    this.brand = brand;
    this.wattage = wattage;
    this.efficiencyRating = efficiencyRating;
    this.link = link;
}
```

```
public Integer getId() {
    return id;
}
```

```
public void setId(Integer id) {
    this.id = id;
}
```

```
public String getName() {
    return name;
}
```

```
public void setName(String name) {
    this.name = name;
}
```

```
public String getBrand() {
    return brand;
}
```

```
public void setBrand(String brand) {
    this.brand = brand;
}
```

```
public Integer getWattage() {
    return wattage;
}
```

```
public void setWattage(Integer wattage) {
    this.wattage = wattage;
}
```

```
public String getEfficiencyRating() {
```

```

        return efficiencyRating;
    }

    public void setEfficiencyRating(String efficiencyRating) {
        this.efficiencyRating = efficiencyRating;
    }

    public String getLink() {
        return link;
    }

    public void setLink(String link) {
        this.link = link;
    }

    @Override
    public String toString() {
        return name;
    }
}
...

...

package com.shop.database.models;

public class RAM implements Identifiable {
    private Integer id;
    private String name;
    private String brand;
    private Integer capacity; // in GB
    private Integer speed; // in MHz
    private String link;

    public RAM(Integer id, String name, String brand, Integer capacity, Integer speed, String
link) {
        this.id = id;
        this.name = name;
        this.brand = brand;
        this.capacity = capacity;
        this.speed = speed;
        this.link = link;
    }

    // Getters and Setters

```



```
public Integer getId() {  
    return id;  
}  
  
public void setId(Integer id) {  
    this.id = id;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
  
public String getBrand() {  
    return brand;  
}  
  
public void setBrand(String brand) {  
    this.brand = brand;  
}  
  
public Integer getCapacity() {  
    return capacity;  
}  
  
public void setCapacity(Integer capacity) {  
    this.capacity = capacity;  
}  
  
public Integer getSpeed() {  
    return speed;  
}  
  
public void setSpeed(Integer speed) {  
    this.speed = speed;  
}  
  
public String getLink() {  
    return link;  
}
```

```

    public void setLink(String link) {
        this.link = link;
    }

    @Override
    public String toString() {
        return name;
    }
}
...

...

package com.shop.database.models;

import java.math.BigDecimal;

public class Cooler implements Identifiable {
    private Integer id;
    private String name;
    private String brand;
    private String type;
    private BigDecimal coolingCapacity; // Using BigDecimal for precision
    private String link;

    public Cooler(Integer id, String name, String brand, String type, BigDecimal
coolingCapacity, String link) {
        this.id = id;
        this.name = name;
        this.brand = brand;
        this.type = type;
        this.coolingCapacity = coolingCapacity;
        this.link = link;
    }

    // Getters and Setters
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {

```

```

    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getBrand() {
    return brand;
}

public void setBrand(String brand) {
    this.brand = brand;
}

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = type;
}

public BigDecimal getCoolingCapacity() {
    return coolingCapacity;
}

public void setCoolingCapacity(BigDecimal coolingCapacity) {
    this.coolingCapacity = coolingCapacity;
}

public String getLink() {
    return link;
}

public void setLink(String link) {
    this.link = link;
}

@Override
public String toString() {
    return name;
}
}

```

...

...

```
package com.shop.database.models;
```

```
public class Case implements Identifiable {
```

```
    private Integer id;
```

```
    private String name;
```

```
    private String brand;
```

```
    private String formFactor;
```

```
    private String color;
```

```
    private String link;
```

```
    public Case(Integer id, String name, String brand, String formFactor, String color, String link) {
```

```
        this.id = id;
```

```
        this.name = name;
```

```
        this.brand = brand;
```

```
        this.formFactor = formFactor;
```

```
        this.color = color;
```

```
        this.link = link;
```

```
    }
```

```
// Getters and Setters
```

```
public Integer getId() {
```

```
    return id;
```

```
}
```

```
public void setId(Integer id) {
```

```
    this.id = id;
```

```
}
```

```
public String getName() {
```

```
    return name;
```

```
}
```

```
public void setName(String name) {
```

```
    this.name = name;
```

```
}
```

```
public String getBrand() {
```

```
    return brand;
```

```
}
```

```

public void setBrand(String brand) {
    this.brand = brand;
}

public String getFormFactor() {
    return formFactor;
}

public void setFormFactor(String formFactor) {
    this.formFactor = formFactor;
}

public String getColor() {
    return color;
}

public void setColor(String color) {
    this.color = color;
}

public String getLink() {
    return link;
}

public void setLink(String link) {
    this.link = link;
}

@Override
public String toString() {
    return name;
}
}
...

...

package com.shop.database.models;

public class ShoppingCartItem {
    private String owner;
    private Integer computerId;
    private Integer quantity;

    public ShoppingCartItem(String owner, Integer computerId, Integer quantity) {

```

```

        this.owner = owner;
        this.computerId = computerId;
        this.quantity = quantity;
    }

    public String getOwner() {
        return owner;
    }

    public void setOwner(String owner) {
        this.owner = owner;
    }

    public Integer getComputerId() {
        return computerId;
    }

    public void setComputerId(Integer computerId) {
        this.computerId = computerId;
    }

    public Integer getQuantity() {
        return quantity;
    }

    public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }
}
...

...

package com.shop.database.models;

import java.math.BigDecimal;
import java.sql.Timestamp;
import java.util.List;

public class Order {
    private Integer id;
    private String customer;
    private Timestamp orderDate;
    private BigDecimal totalAmount;
    private String comment;

```

```

private String status;
private List<OrderItem> items;

public Order(Integer id, String customer, Timestamp orderDate, BigDecimal totalAmount,
String status, String comment, List<OrderItem> items) {
    this.id = id;
    this.customer = customer;
    this.orderDate = orderDate;
    this.totalAmount = totalAmount;
    this.status = status;
    this.items = items;
    this.comment = comment;
}

public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}

public String getCustomer() {
    return customer;
}

public void setCustomer(String customer) {
    this.customer = customer;
}

public Timestamp getOrderDate() {
    return orderDate;
}

public void setOrderDate(Timestamp orderDate) {
    this.orderDate = orderDate;
}

public BigDecimal getTotalAmount() {
    return totalAmount;
}

public void setTotalAmount(BigDecimal totalAmount) {
    this.totalAmount = totalAmount;
}

```

```

    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public void setItems(List<OrderItem> itms) {
        this.items = itms;
    }

    public List<OrderItem> getItems() {
        return items;
    }

    public void setComment(String value) {
        this.comment = value;
    }

    public String getComment() {
        return comment;
    }
}
...

...

package com.shop.database.models;

public class OrderItem {
    private Integer id;
    private Integer orderId;
    private Computer computer;
    private Integer quantity;

    public OrderItem(Integer id, Integer orderId, Computer pc, Integer quantity) {
        this.id = id;
        this.orderId = orderId;
        this.computer = pc;
        this.quantity = quantity;
    }
}

```



```

public Integer getId() {
    return id;
}

public void setId(Integer id) {
    this.id = id;
}

public Integer getOrderId() {
    return orderId;
}

public void setOrderId(Integer orderId) {
    this.orderId = orderId;
}

public Computer getComputer() {
    return computer;
}

public void setComputer(Computer pc) {
    this.computer = pc;
}

public Integer getQuantity() {
    return quantity;
}

public void setQuantity(Integer quantity) {
    this.quantity = quantity;
}
}
...

...

package com.shop.database.models;

public class OrderResult {
    private boolean success;
    private String message;

    public OrderResult(boolean success, String message) {
        this.success = success;
        this.message = message;
    }
}

```

```

    }

    public boolean isSuccess() {
        return success;
    }

    public String getMessage() {
        return message;
    }
}
...

```

3.5 Разработка класса подключения к базе данных

Листинг класса **DbConnection** (класс для взаимодействия с базой данных)

```

...

package com.shop.database;

import java.io.FileInputStream;
import java.math.BigDecimal;
import java.security.Timestamp;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;

import com.shop.database.models.User;
import com.shop.helper.AlertHelper;
import com.shop.database.models.Processor;
import com.shop.database.models.GraphicCard;
import com.shop.database.models.Motherboard;
import com.shop.database.models.Order;
import com.shop.database.models.OrderItem;
import com.shop.database.models.OrderResult;
import com.shop.database.models.PowerSupply;

```

```

import com.shop.database.models.RAM;
import com.shop.database.models.ShoppingCartItem;
import com.shop.database.models.Cooler;
import com.shop.database.models.Case;
import com.shop.database.models.Computer;

public class DbConnection {
    private Connection con;
    private static DbConnection dbc;

    public void createTablesIfNotExists() {
        String createUserTable = "CREATE TABLE IF NOT EXISTS users (" +
            "username TEXT PRIMARY KEY NOT NULL, " +
            "email TEXT NOT NULL, " +
            "first_name TEXT NOT NULL, " +
            "last_name TEXT NOT NULL, " +
            "password TEXT NOT NULL, " +
            "role VARCHAR(25) NOT NULL DEFAULT 'user', " +
            "created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP" +
            ");";

        String createProcessorsTable = "CREATE TABLE IF NOT EXISTS processors (" +
            "id SERIAL PRIMARY KEY, " +
            "name TEXT NOT NULL, " +
            "brand TEXT NOT NULL, " +
            "cores INT NOT NULL, " +
            "threads INT NOT NULL, " +
            "base_clock DECIMAL(5,2) NOT NULL, " +
            "boost_clock DECIMAL(5,2) NOT NULL, " +
            "link TEXT NOT NULL" +
            ");";

        String createGraphicsCardsTable = "CREATE TABLE IF NOT EXISTS graphic_cards ("
+
            "id SERIAL PRIMARY KEY, " +
            "name TEXT NOT NULL, " +
            "brand TEXT NOT NULL, " +
            "memory_size INT NOT NULL, " +
            "memory_type TEXT NOT NULL, " +
            "link TEXT NOT NULL" +
            ");";
    }
}

```

```
String createPowerSuppliesTable = "CREATE TABLE IF NOT EXISTS power_supplies
(" +
    "id SERIAL PRIMARY KEY, " +
    "name TEXT NOT NULL, " +
    "brand TEXT NOT NULL, " +
    "wattage INT NOT NULL, " +
    "efficiency_rating TEXT NOT NULL, " +
    "link TEXT NOT NULL" +
    ");";
```

```
+ String createMotherboardsTable = "CREATE TABLE IF NOT EXISTS motherboards ("
    "id SERIAL PRIMARY KEY, " +
    "name TEXT NOT NULL, " +
    "brand TEXT NOT NULL, " +
    "socket_type TEXT NOT NULL, " +
    "form_factor TEXT NOT NULL, " +
    "max_memory INT NOT NULL, " +
    "link TEXT NOT NULL" +
    ");";
```

```
String createCoolersTable = "CREATE TABLE IF NOT EXISTS coolers (" +
    "id SERIAL PRIMARY KEY, " +
    "name TEXT NOT NULL, " +
    "brand TEXT NOT NULL, " +
    "type TEXT NOT NULL, " +
    "cooling_capacity DECIMAL(5,2) NOT NULL, " +
    "link TEXT NOT NULL" +
    ");";
```

```
String createCasesTable = "CREATE TABLE IF NOT EXISTS cases (" +
    "id SERIAL PRIMARY KEY, " +
    "name TEXT NOT NULL, " +
    "brand TEXT NOT NULL, " +
    "form_factor TEXT NOT NULL, " +
    "color TEXT, " +
    "link TEXT NOT NULL" +
    ");";
```

```
String createRAMTable = "CREATE TABLE IF NOT EXISTS rams (" +
    "id SERIAL PRIMARY KEY, " +
    "name TEXT NOT NULL, " +
    "brand TEXT NOT NULL, " +
    "capacity INT NOT NULL, " + // Объем памяти в ГБ
```

```
"speed INT NOT NULL," + // Скорость в МГц
"link TEXT NOT NULL" +
");";
```

```
String createComputersTable = "CREATE TABLE IF NOT EXISTS computers (" +
    "id SERIAL PRIMARY KEY, " +
    "name TEXT NOT NULL, " +
    "description TEXT NOT NULL, " +
    "price DECIMAL(10,2) NOT NULL, " +
    "processor_id INT NOT NULL REFERENCES processors(id), " +
    "graphic_card_id INT NOT NULL REFERENCES graphic_cards(id), " +
    "motherboard_id INT NOT NULL REFERENCES motherboards(id), " +
    "ram_id INT NOT NULL REFERENCES rams(id), " +
    "rams_count INT NOT NULL, " +
    "power_supply_id INT NOT NULL REFERENCES power_supplies(id), " +
    "cooler_id INT NOT NULL REFERENCES coolers(id), " +
    "case_id INT NOT NULL REFERENCES cases(id), " +
    "image_url TEXT NOT NULL, " +
    "stock_quantity INT NOT NULL DEFAULT 0" +
    ");";
```

```
+
String createShoppingCartTable = "CREATE TABLE IF NOT EXISTS shopping_cart (" +
    "owner TEXT NOT NULL REFERENCES users(username), " +
    "computer_id INT NOT NULL REFERENCES computers(id), " +
    "quantity INT NOT NULL DEFAULT 1" +
    ");";
```

```
String createOrdersTable = "CREATE TABLE IF NOT EXISTS orders (" +
    "id SERIAL PRIMARY KEY, " +
    "customer TEXT NOT NULL REFERENCES users(username), " +
    "order_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, " +
    "total_amount DECIMAL(10,2) NOT NULL, " +
    "status VARCHAR(25) NOT NULL DEFAULT 'pending', " +
    "comment TEXT" +
    ");";
```

```
String createOrderItemsTable = "CREATE TABLE IF NOT EXISTS order_items (" +
    "id SERIAL PRIMARY KEY, " +
    "order_id INT NOT NULL REFERENCES orders(id), " +
    "computer_id INT NOT NULL REFERENCES computers(id), " +
    "quantity INT NOT NULL, " +
    "price DECIMAL(10,2) NOT NULL" +
    ");";
```

```

String createOrderFunction = "CREATE OR REPLACE FUNCTION
update_order_total() " +
    "RETURNS TRIGGER AS $$ " +
    "BEGIN " +
    "    UPDATE orders " +
    "    SET total_amount = ( " +
    "        SELECT COALESCE(SUM(price * quantity), 0) " +
    "        FROM order_items " +
    "        WHERE order_id = NEW.order_id " +
    "    ) " +
    "    WHERE id = NEW.order_id; " +
    "    RETURN NEW; " +
    "END; $$ LANGUAGE plpgsql;";

```

```

String createOrderTrigger = "DO $$ BEGIN " +
    "IF NOT EXISTS (SELECT 1 FROM pg_trigger WHERE tgname =
'order_total_update') THEN " +
    "CREATE TRIGGER order_total_update " +
    "AFTER INSERT OR UPDATE OR DELETE ON order_items " +
    "FOR EACH ROW EXECUTE FUNCTION update_order_total(); " +
    "END IF; END $$;";

```

```

try (Statement statement = con.createStatement()) {
    statement.executeUpdate(createUsersTable);
    statement.executeUpdate(createProcessorsTable);
    statement.executeUpdate(createGraphicsCardsTable);
    statement.executeUpdate(createPowerSuppliesTable);
    statement.executeUpdate(createMotherboardsTable);
    statement.executeUpdate(createCoolersTable);
    statement.executeUpdate(createCasesTable);
    statement.executeUpdate(createRAMTable);
    statement.executeUpdate(createComputersTable);
    statement.executeUpdate(createShoppingCartTable);
    statement.executeUpdate(createOrdersTable);
    statement.executeUpdate(createOrderItemsTable);
    statement.executeUpdate(createOrderFunction);
    statement.executeUpdate(createOrderTrigger);
} catch (Exception ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
}
}

```

```

private DbConnection() {
    try {
        DriverManager.registerDriver(new org.postgresql.Driver());
        FileInputStream fis = new FileInputStream("connection.prop");
        Properties p = new Properties();
        p.load(fis);
        con = DriverManager.getConnection((String) p.get("url"), (String) p.get("username"),
(String) p.get("password"));
    } catch (Exception ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
    }
}

public static DbConnection getDatabaseConnection() {
    if (dbc == null) {
        dbc = new DbConnection();
    }
    return dbc;
}

public Connection getConnection() {
    return con;
}

```

```

//==> USER CRUD
//

```

```

=====
=====

```

```

public List<User> getAllUsers() {
    List<User> users = new ArrayList<>();
    String query = "SELECT * FROM users";
    try (PreparedStatement pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            User user = new User(
                rs.getString("username"),
                rs.getString("email"),
                rs.getString("first_name"),
                rs.getString("last_name"),
                rs.getString("password"),
                rs.getString("role"),
            );
            users.add(user);
        }
    }
    return users;
}

```

```

        rs.getTimestamp("created_at")
    );

    users.add(user);
}
} catch (SQLException ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
}
return users;
}

```

```

public User getUserByUsername(String username) {
    String query = "SELECT * FROM users WHERE username = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setString(1, username);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            User user = new User(
                rs.getString("username"),
                rs.getString("email"),
                rs.getString("first_name"),
                rs.getString("last_name"),
                rs.getString("password"),
                rs.getString("role"),
                rs.getTimestamp("created_at")
            );
            return user;
        } else {
            return null;
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return null;
    }
}

```

```

public boolean deleteUser(String username) {
    String deleteUser = "DELETE FROM users WHERE username = ?";
    try (PreparedStatement pstmt = con.prepareStatement(deleteUser)) {
        pstmt.setString(1, username);
        return pstmt.executeUpdate() > 0;
    }
}

```



```

    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean updateUser(User user) {
    String updateUser = "UPDATE users SET email = ?, first_name = ?, last_name = ?,
password = ?, role = ? WHERE username = ?";
    try (PreparedStatement pstmt = con.prepareStatement(updateUser)) {
        pstmt.setString(1, user.getEmail());
        pstmt.setString(2, user.getFirstName());
        pstmt.setString(3, user.getLastName());
        pstmt.setString(4, user.getPassword());
        pstmt.setString(5, user.getRole());
        pstmt.setString(6, user.getUsername());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean addUser(User user) {
    String insertUser = "INSERT INTO users (first_name, last_name, email, username,
password, role) VALUES (?, ?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt = con.prepareStatement(insertUser)) {
        pstmt.setString(1, user.getFirstName());
        pstmt.setString(2, user.getLastName());
        pstmt.setString(3, user.getEmail());
        pstmt.setString(4, user.getUsername());
        pstmt.setString(5, user.getPassword());
        pstmt.setString(6, user.getRole());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean isUsernameExists(String username) {

```

```

String query = "SELECT * FROM users WHERE username = ?";
try (PreparedStatement ps = con.prepareStatement(query)) {
    ps.setString(1, username);
    try (ResultSet rs = ps.executeQuery()) {
        return rs.next();
    }
} catch (SQLException ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
    return false;
}
}

public boolean authenticateUser(String username, String password) {
    String query = "SELECT * FROM users WHERE username = ? AND password = ?";
    try (PreparedStatement ps = con.prepareStatement(query)) {
        ps.setString(1, username);
        ps.setString(2, password);
        try (ResultSet rs = ps.executeQuery()) {
            return rs.next();
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}
}

```

//==> PROCESSOR CRUD

//

```

=====
=====

```

```

public List<Processor> getAllProcessors() {
    List<Processor> processors = new ArrayList<>();
    String query = "SELECT * FROM processors";
    try (PreparedStatement pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            Processor processor = new Processor(
                rs.getInt("id"),

```

```

        rs.getString("name"),
        rs.getString("brand"),
        rs.getInt("cores"),
        rs.getInt("threads"),
        rs.getBigDecimal("base_clock"),
        rs.getBigDecimal("boost_clock"),
        rs.getString("link")
    );
    processors.add(processor);
}
} catch (SQLException ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
}
return processors;
}

public Processor getProcessorById(int id) {
    String query = "SELECT * FROM processors WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return new Processor(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getInt("cores"),
                rs.getInt("threads"),
                rs.getBigDecimal("base_clock"),
                rs.getBigDecimal("boost_clock"),
                rs.getString("link")
            );
        } else {
            return null;
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return null;
    }
}
}

```

```

public boolean deleteProcessor(Integer id) {
    String deleteProcessor = "DELETE FROM processors WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(deleteProcessor)) {
        pstmt.setInt(1, id);
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

```

public boolean addProcessor(Processor processor) {
    String insertProcessor = "INSERT INTO processors (name, brand, cores, threads,
base_clock, boost_clock, link) VALUES (?, ?, ?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt = con.prepareStatement(insertProcessor)) {
        pstmt.setString(1, processor.getName());
        pstmt.setString(2, processor.getBrand());
        pstmt.setInt(3, processor.getCores());
        pstmt.setInt(4, processor.getThreads());
        pstmt.setBigDecimal(5, processor.getBaseClock());
        pstmt.setBigDecimal(6, processor.getBoostClock());
        pstmt.setString(7, processor.getLink());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

```

public boolean updateProcessor(Processor processor) {
    String updateProcessor = "UPDATE processors SET name = ?, brand = ?, cores = ?,
threads = ?, base_clock = ?, boost_clock = ?, link = ? WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(updateProcessor)) {
        pstmt.setString(1, processor.getName());
        pstmt.setString(2, processor.getBrand());
        pstmt.setInt(3, processor.getCores());
        pstmt.setInt(4, processor.getThreads());
        pstmt.setBigDecimal(5, processor.getBaseClock());
        pstmt.setBigDecimal(6, processor.getBoostClock());
        pstmt.setString(7, processor.getLink());
        pstmt.setInt(8, processor.getId());
        return pstmt.executeUpdate() > 0;
    }
}

```

```

    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

//==> GRAPHICSCARD CRUD

//

```

=====
=====

```

```

public List<GraphicCard> getAllGraphicCards() {
    List<GraphicCard> graphicsCards = new ArrayList<>();
    String query = "SELECT * FROM graphic_cards";
    try (PreparedStatement pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            GraphicCard graphicsCard = new GraphicCard(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getInt("memory_size"),
                rs.getString("memory_type"),
                rs.getString("link")
            );
            graphicsCards.add(graphicsCard);
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
    }
    return graphicsCards;
}

```

```

public GraphicCard getGraphicCardById(int id) {
    String query = "SELECT * FROM graphic_cards WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return new GraphicCard(

```

```

        rs.getInt("id"),
        rs.getString("name"),
        rs.getString("brand"),
        rs.getInt("memory_size"),
        rs.getString("memory_type"),
        rs.getString("link")
    );
} else {
    return null;
}
} catch (SQLException ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
    return null;
}
}

public boolean deleteGraphicCard(Integer id) {
    String deleteGraphicsCard = "DELETE FROM graphic_cards WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(deleteGraphicsCard)) {
        pstmt.setInt(1, id);
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean addGraphicCard(GraphicCard graphicsCard) {
    String insertGraphicsCard = "INSERT INTO graphic_cards (name, brand, memory_size, memory_type, link) VALUES (?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt = con.prepareStatement(insertGraphicsCard)) {
        pstmt.setString(1, graphicsCard.getName());
        pstmt.setString(2, graphicsCard.getBrand());
        pstmt.setInt(3, graphicsCard.getMemorySize());
        pstmt.setString(4, graphicsCard.getMemoryType());
        pstmt.setString(5, graphicsCard.getLink());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

```

}

public boolean updateGraphicCard(GraphicCard graphicsCard) {
    String updateGraphicsCard = "UPDATE graphic_cards SET name = ?, brand = ?,
memory_size = ?, memory_type = ?, link = ? WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(updateGraphicsCard)) {
        pstmt.setString(1, graphicsCard.getName());
        pstmt.setString(2, graphicsCard.getBrand());
        pstmt.setInt(3, graphicsCard.getMemorySize());
        pstmt.setString(4, graphicsCard.getMemoryType());
        pstmt.setString(5, graphicsCard.getLink());
        pstmt.setInt(6, graphicsCard.getId());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null ,ex);
        return false;
    }
}

```

//==> POWERSUPPLY CRUD

//

```

=====
=====

```

```

public List<PowerSupply> getAllPowerSupplies() {
    List<PowerSupply> powerSupplies = new ArrayList<>();
    String query = "SELECT * FROM power_supplies";

    try (PreparedStatement pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            PowerSupply powerSupply = new PowerSupply(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getInt("wattage"),
                rs.getString("efficiency_rating"),
                rs.getString("link")
            );
            powerSupplies.add(powerSupply);
        }
    }
}

```

```

    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
    }
    return powerSupplies;
}

public PowerSupply getPowerSupplyById(int id) {
    String query = "SELECT * FROM power_supplies WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return new PowerSupply(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getInt("wattage"),
                rs.getString("efficiency_rating"),
                rs.getString("link")
            );
        } else {
            return null;
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return null;
    }
}

public boolean deletePowerSupply(Integer id) {
    String deletePowerSupply = "DELETE FROM power_supplies WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(deletePowerSupply)) {
        pstmt.setInt(1, id);
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```



```

public boolean addPowerSupply(PowerSupply powerSupply) {
    String insertPowerSupply = "INSERT INTO power_supplies (name, brand, wattage,
efficiency_rating, link) VALUES (?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt = con.prepareStatement(insertPowerSupply)) {
        pstmt.setString(1, powerSupply.getName());
        pstmt.setString(2, powerSupply.getBrand());
        pstmt.setInt(3, powerSupply.getWattage());
        pstmt.setString(4, powerSupply.getEfficiencyRating());
        pstmt.setString(5, powerSupply.getLink());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

```

public boolean updatePowerSupply(PowerSupply powerSupply) {
    String updatePowerSupply = "UPDATE power_supplies SET name = ?, brand = ?,
wattage = ?, efficiency_rating = ?, link = ? WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(updatePowerSupply)) {
        pstmt.setString(1, powerSupply.getName());
        pstmt.setString(2, powerSupply.getBrand());
        pstmt.setInt(3, powerSupply.getWattage());
        pstmt.setString(4, powerSupply.getEfficiencyRating());
        pstmt.setString(5, powerSupply.getLink());
        pstmt.setInt(6, powerSupply.getId());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

//==> RAM CRUD

//

=====

=====

```

public List<RAM> getAllRAMs() {
    List<RAM> rams = new ArrayList<>();
    String query = "SELECT * FROM rams";

```

```

try (PreparedStatement pstmt = con.prepareStatement(query);
    ResultSet rs = pstmt.executeQuery()) {
    while (rs.next()) {
        RAM ram = new RAM(
            rs.getInt("id"),
            rs.getString("name"),
            rs.getString("brand"),
            rs.getInt("capacity"), // in GB
            rs.getInt("speed"), // in MHz
            rs.getString("link")
        );
        rams.add(ram);
    }
} catch (SQLException ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
}
return rams;
}

```

```

public RAM getRAMById(int id) {
    String query = "SELECT * FROM rams WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return new RAM(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getInt("capacity"), // in GB
                rs.getInt("speed"), // in MHz
                rs.getString("link")
            );
        } else {
            return null;
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return null;
    }
}

```

```

}

public boolean deleteRAM(Integer id) {
    String deleteRAM = "DELETE FROM rams WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(deleteRAM)) {
        pstmt.setInt(1, id);
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean addRAM(RAM ram) {
    String insertRAM = "INSERT INTO rams (name, brand, capacity, speed, link) VALUES
(?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt = con.prepareStatement(insertRAM)) {
        pstmt.setString(1, ram.getName());
        pstmt.setString(2, ram.getBrand());
        pstmt.setInt(3, ram.getCapacity());
        pstmt.setInt(4, ram.getSpeed());
        pstmt.setString(5, ram.getLink());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean updateRAM(RAM ram) {
    String updateRAM = "UPDATE rams SET name = ?, brand = ?, capacity = ?, speed = ?,
link = ? WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(updateRAM)) {
        pstmt.setString(1, ram.getName());
        pstmt.setString(2, ram.getBrand());
        pstmt.setInt(3, ram.getCapacity());
        pstmt.setInt(4, ram.getSpeed());
        pstmt.setString(5, ram.getLink());
        pstmt.setInt(6, ram.getId());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);

```

```

        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

```

//==> MOTHERBOARD CRUD
//

```

```

=====
=====

```

```

public List<Motherboard> getAllMotherboards() {
    List<Motherboard> motherboards = new ArrayList<>();
    String query = "SELECT * FROM motherboards";

    try (PreparedStatement pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            Motherboard motherboard = new Motherboard(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getString("socket_type"),
                rs.getString("form_factor"),
                rs.getInt("max_memory"),
                rs.getString("link")
            );
            motherboards.add(motherboard);
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
    }
    return motherboards;
}

```

```

public Motherboard getMotherboardById(int id) {
    String query = "SELECT * FROM motherboards WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
    }
}

```

```

        if (rs.next()) {
            return new Motherboard(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getString("socket_type"),
                rs.getString("form_factor"),
                rs.getInt("max_memory"),
                rs.getString("link")
            );
        } else {
            return null;
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return null;
    }
}

```

```

public boolean deleteMotherboard(Integer id) {
    String deleteMotherboard = "DELETE FROM motherboards WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(deleteMotherboard)) {
        pstmt.setInt(1, id);
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

```

public boolean addMotherboard(Motherboard motherboard) {
    String insertMotherboard = "INSERT INTO motherboards (name, brand, socket_type, form_factor, max_memory, link) VALUES (?, ?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt = con.prepareStatement(insertMotherboard)) {
        pstmt.setString(1, motherboard.getName());
        pstmt.setString(2, motherboard.getBrand());
        pstmt.setString(3, motherboard.getSocketType());
        pstmt.setString(4, motherboard.getFormFactor());
        pstmt.setInt(5, motherboard.getMaxMemory());
        pstmt.setString(6, motherboard.getLink());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {

```

```

        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean updateMotherboard(Motherboard motherboard) {
    String updateMotherboard = "UPDATE motherboards SET name = ?, brand = ?,
socket_type = ?, form_factor = ?, max_memory = ?, link = ? WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(updateMotherboard)) {
        pstmt.setString(1, motherboard.getName());
        pstmt.setString(2, motherboard.getBrand());
        pstmt.setString(3, motherboard.getSocketType());
        pstmt.setString(4, motherboard.getFormFactor());
        pstmt.setInt(5, motherboard.getMaxMemory());
        pstmt.setString(6, motherboard.getLink());
        pstmt.setInt(7, motherboard.getId());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

//==> COOLER CRUD

//

=====

=====

```

public List<Cooler> getAllCoolers() {
    List<Cooler> coolers = new ArrayList<>();
    String query = "SELECT * FROM coolers";

    try (PreparedStatement pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            Cooler cooler = new Cooler(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getString("type"),
            );
            coolers.add(cooler);
        }
    }
}

```

```

        rs.getBigDecimal("cooling_capacity"),
        rs.getString("link")
    );
    coolers.add(cooler);
}
} catch (SQLException ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
}
return coolers;
}

```

```

public Cooler getCoolerById(int id) {
    String query = "SELECT * FROM coolers WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return new Cooler(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getString("type"),
                rs.getBigDecimal("cooling_capacity"),
                rs.getString("link")
            );
        } else {
            return null;
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return null;
    }
}

```

```

public boolean deleteCooler(Integer id) {
    String deleteCooler = "DELETE FROM coolers WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(deleteCooler)) {
        pstmt.setInt(1, id);
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

```

        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean addCooler(Cooler cooler) {
    String insertCooler = "INSERT INTO coolers (name, brand, type, cooling_capacity, link)
VALUES (?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt = con.prepareStatement(insertCooler)) {
        pstmt.setString(1, cooler.getName());
        pstmt.setString(2, cooler.getBrand());
        pstmt.setString(3, cooler.getType());
        pstmt.setBigDecimal(4, cooler.getCoolingCapacity());
        pstmt.setString(5, cooler.getLink());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean updateCooler(Cooler cooler) {
    String updateCooler = "UPDATE coolers SET name = ?, brand = ?, type = ?,
cooling_capacity = ?, link = ? WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(updateCooler)) {
        pstmt.setString(1, cooler.getName());
        pstmt.setString(2, cooler.getBrand());
        pstmt.setString(3, cooler.getType());
        pstmt.setBigDecimal(4, cooler.getCoolingCapacity());
        pstmt.setString(5, cooler.getLink());
        pstmt.setInt(6, cooler.getId());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

//==> CASE CRUD


```
//
```

```
=====
```

```
public List<Case> getAllCases() {
    List<Case> cases = new ArrayList<>();
    String query = "SELECT * FROM cases";

    try (PreparedStatement pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            Case computerCase = new Case(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getString("form_factor"),
                rs.getString("color"),
                rs.getString("link")
            );
            cases.add(computerCase);
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
    }
    return cases;
}
```

```
public Case getCaseById(int id) {
    String query = "SELECT * FROM cases WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return new Case(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("brand"),
                rs.getString("form_factor"),
                rs.getString("color"),
                rs.getString("link")
            );
        } else {
            return null;
        }
    }
}
```

```

    }
} catch (SQLException ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
    return null;
}
}

public boolean deleteCase(Integer id) {
    String deleteCase = "DELETE FROM cases WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(deleteCase)) {
        pstmt.setInt(1, id);
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean addCase(Case computerCase) {
    String insertCase = "INSERT INTO cases (name, brand, form_factor, color, link)
VALUES (?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt = con.prepareStatement(insertCase)) {
        pstmt.setString(1, computerCase.getName());
        pstmt.setString(2, computerCase.getBrand());
        pstmt.setString(3, computerCase.getFormFactor());
        pstmt.setString(4, computerCase.getColor());
        pstmt.setString(5, computerCase.getLink());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean updateCase(Case computerCase) {
    String updateCase = "UPDATE cases SET name = ?, brand = ?, form_factor = ?, color
= ?, link = ? WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(updateCase)) {
        pstmt.setString(1, computerCase.getName());
        pstmt.setString(2, computerCase.getBrand());
        pstmt.setString(3, computerCase.getFormFactor());

```

```

        pstmt.setString(4, computerCase.getColor());
        pstmt.setString(5, computerCase .getLink());
        pstmt.setInt (6 ,computerCase .getId());
        return  pstmt.executeUpdate()>0;
    } catch(SQLException ex){
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

//==> COMPUTER CRUD

//

```

=====
=====

```

```

public List<Computer> getAllComputers() {
    List<Computer> computers = new ArrayList<>();
    String query = "SELECT * FROM computers";

    try (PreparedStatement pstmt = con.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery()) {
        while (rs.next()) {
            Computer computer = new Computer(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("description"),
                rs.getBigDecimal("price"),
                rs.getInt("processor_id"),
                rs.getInt("graphic_card_id"),
                rs.getInt("motherboard_id"),
                rs.getInt("ram_id"),
                rs.getInt("rams_count"),
                rs.getInt("power_supply_id"),
                rs.getInt("cooler_id"),
                rs.getInt("case_id"),
                rs.getString("image_url"),
                rs.getInt("stock_quantity")
            );
            computers.add(computer);
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

```

        AlertHelper.showErrorAlert("Unknown Error. Try again");
    }
    return computers;
}

public Computer getComputerById(Integer id) {
    String query = "SELECT * FROM computers WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return new Computer(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("description"),
                rs.getBigDecimal("price"),
                rs.getInt("processor_id"),
                rs.getInt("graphic_card_id"),
                rs.getInt("motherboard_id"),
                rs.getInt("ram_id"),
                rs.getInt("rams_count"),
                rs.getInt("power_supply_id"),
                rs.getInt("cooler_id"),
                rs.getInt("case_id"),
                rs.getString("image_url"),
                rs.getInt("stock_quantity")
            );
        } else {
            return null;
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return null;
    }
}

```

```

public boolean addComputer(Computer computer) {
    String insertComputerSQL = "INSERT INTO computers (name, description, price,
processor_id, graphic_card_id, " +
        "motherboard_id, ram_id, rams_count, power_supply_id, cooler_id, case_id,
image_url, stock_quantity) " +
        "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
}

```

```

try (PreparedStatement pstmt = con.prepareStatement(insertComputerSQL)) {
    pstmt.setString(1, computer.getName());
    pstmt.setString(2, computer.getDescription());
    pstmt.setBigDecimal(3, computer.getPrice());
    pstmt.setInt(4, computer.getProcessorId());
    pstmt.setInt(5, computer.getGraphicCardId());
    pstmt.setInt(6, computer.getMotherboardId());
    pstmt.setInt(7, computer.getRamId());
    pstmt.setInt(8, computer.getRamsCount());
    pstmt.setInt(9, computer.getPowerSupplyId());
    pstmt.setInt(10, computer.getCoolerId());
    pstmt.setInt(11, computer.getCaseId());
    pstmt.setString(12, computer.getImageUrl());
    pstmt.setInt(13, computer.getStockQuantity());

    return pstmt.executeUpdate() > 0;

} catch (SQLException ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
    return false;
}

}

public boolean updateComputer(Computer computer) {
    String updateComputer = "UPDATE computers SET name = ?, description = ?, price = ?,
processor_id = ?, graphic_card_id = ?, motherboard_id = ?, ram_id = ?, rams_count = ?,
power_supply_id = ?, cooler_id = ?, case_id = ?, image_url = ?, stock_quantity = ? WHERE id
= ?";
    try (PreparedStatement pstmt = con.prepareStatement(updateComputer)) {
        pstmt.setString(1, computer.getName());
        pstmt.setString(2, computer.getDescription());
        pstmt.setBigDecimal(3, computer.getPrice());
        pstmt.setInt(4, computer.getProcessorId());
        pstmt.setInt(5, computer.getGraphicCardId());
        pstmt.setInt(6, computer.getMotherboardId());
        pstmt.setInt(7, computer.getRamId());
        pstmt.setInt(8, computer.getRamsCount());
        pstmt.setInt(9, computer.getPowerSupplyId());
        pstmt.setInt(10, computer.getCoolerId());
        pstmt.setInt(11, computer.getCaseId());
        pstmt.setString(12, computer.getImageUrl());
        pstmt.setInt(13, computer.getStockQuantity());
    }
}

```

```

        pstmt.setInt(14, computer.getId());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

private void updateStockQuantities(Map<Integer, Integer> computerIdsAndQuantities)
throws SQLException {
    String updateQuery = "UPDATE computers SET stock_quantity = stock_quantity - ?
WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(updateQuery)) {
        for (Map.Entry<Integer, Integer> entry : computerIdsAndQuantities.entrySet()) {
            int computerId = entry.getKey();
            int quantityToReduce = entry.getValue();

            pstmt.setInt(1, quantityToReduce);
            pstmt.setInt(2, computerId);
            pstmt.addBatch();
        }
        pstmt.executeBatch();
    }
}

private String checkStockAvailability(Map<Integer, Integer> computerIdsAndQuantities) {
    for (Map.Entry<Integer, Integer> entry : computerIdsAndQuantities.entrySet()) {
        int computerId = entry.getKey();
        int quantityRequested = entry.getValue();

        String stockCheckQuery = "SELECT stock_quantity, name FROM computers WHERE
id = ?";
        try (PreparedStatement pstmt = con.prepareStatement(stockCheckQuery)) {
            pstmt.setInt(1, computerId);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                int stockQuantity = rs.getInt("stock_quantity");
                String computerName = rs.getString("name");
                if (stockQuantity < quantityRequested) {
                    return "Out of stock: " + computerName; // недостаточно на складе
                }
            }
        } else {

```

```

        return "Computer with ID " + computerId + " not found."; // Компьютер не
найден
    }
    } catch (SQLException e) {
        e.printStackTrace();
        return "Error during warehouse verification: " + e.getMessage();
    }
}
return null; // Все товары доступны
}

```

```

public boolean deleteComputer(Integer id) {
    String deleteComputer = "DELETE FROM computers WHERE id = ?";
    try (PreparedStatement pstmt = con.prepareStatement(deleteComputer)) {
        pstmt.setInt(1, id);
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

```

//==> SHOPPING CART CRUD
//

```

```

=====
=====

```

```

public List<ShoppingCartItem> getShoppingCartItemsByOwner(String owner) {
    String query = "SELECT * FROM shopping_cart WHERE owner = ?";
    List<ShoppingCartItem> items = new ArrayList<>();

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setString(1, owner);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            ShoppingCartItem item = new ShoppingCartItem(
                rs.getString("owner"),
                rs.getInt("computer_id"),
                rs.getInt("quantity")
            );
            items.add(item);
        }
    }
}

```

```

        );
        items.add(item);
    }
} catch (SQLException ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
}
return items;
}

public ShoppingCartItem getShoppingCartItemById(String owner, Integer computer_id) {
    String query = "SELECT * FROM shopping_cart WHERE computer_id = ? AND owner = ?";

    try (PreparedStatement pstmt = con.prepareStatement(query)) {
        pstmt.setInt(1, computer_id);
        pstmt.setString(2, owner);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            return new ShoppingCartItem(
                rs.getString("owner"),
                rs.getInt("computer_id"),
                rs.getInt("quantity")
            );
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
    }
    return null;
}

public boolean addShoppingCartItem(ShoppingCartItem item) {
    String insertComputerSQL = "INSERT INTO shopping_cart (owner, computer_id, quantity) VALUES (?, ?, ?)";

    try (PreparedStatement pstmt = con.prepareStatement(insertComputerSQL)) {
        pstmt.setString(1, item.getOwner());
        pstmt.setInt(2, item.getComputerId());
        pstmt.setInt(3, item.getQuantity());

        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```



```

        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean updateShoppingCartItem(ShoppingCartItem item) {
    String updateQuery = "UPDATE shopping_cart SET quantity = ? WHERE computer_id =
? AND owner = ?";

    try (PreparedStatement pstmt = con.prepareStatement(updateQuery)) {
        pstmt.setInt(1, item.getQuantity());
        pstmt.setInt(2, item.getComputerId());
        pstmt.setString(3, item.getOwner());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean deleteShoppingCartItem(ShoppingCartItem item) {
    String deleteQuery = "DELETE FROM shopping_cart WHERE computer_id = ? AND
owner = ?";

    try (PreparedStatement pstmt = con.prepareStatement(deleteQuery)) {
        pstmt.setInt(1, item.getComputerId());
        pstmt.setString(2, item.getOwner());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

public boolean clearShoppingCart(String username) {
    String deleteQuery = "DELETE FROM shopping_cart WHERE owner = ?";

    try (PreparedStatement pstmt = con.prepareStatement(deleteQuery)) {
        pstmt.setString(1, username);
        int rowsAffected = pstmt.executeUpdate();
        return rowsAffected > 0; // Возвращает true, если были удалены записи
    } catch (SQLException e) {

```

```

        e.printStackTrace();
        AlertHelper.showErrorAlert("Ошибка при очистке корзины: " + e.getMessage());
        return false;
    }
}

```

```

//==> ORDERS CRUD
//

```

```

=====
=====

public List<Order> getUserOrders(String username) {
    List<Order> orders = new ArrayList<>();

    String orderQuery = "SELECT * FROM orders WHERE customer = ?";

    try (PreparedStatement orderStmt = con.prepareStatement(orderQuery)) {
        orderStmt.setString(1, username);
        ResultSet orderRs = orderStmt.executeQuery();

        while (orderRs.next()) {
            Integer orderId = orderRs.getInt("id");
            List<OrderItem> orderItems = getOrderItems(orderId);

            orders.add(new Order(
                orderId,
                orderRs.getString("customer"),
                orderRs.getTimestamp("order_date"),
                new BigDecimal(orderRs.getDouble("total_amount")),
                orderRs.getString("status"),
                orderRs.getString("comment"),
                orderItems
            ));
        }
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
    }

    return orders;
}

```

```

public OrderResult createOrder(String customerUsername, String comment, Map<Integer,
Integer> computerIdsAndQuantities) {
    String checkStatus = checkStockAvailability(computerIdsAndQuantities);
    if (checkStatus != null) {
        return new OrderResult(false, checkStatus);
    }

    try {
        con.setAutoCommit(false); // Отключаем автоматическое подтверждение
        String orderInsertQuery = "INSERT INTO orders (customer, comment, total_amount)
VALUES (?, ?, ?) RETURNING id";
        PreparedStatement orderStmt = con.prepareStatement(orderInsertQuery);
        orderStmt.setString(1, customerUsername);
        orderStmt.setString(2, comment);
        orderStmt.setDouble(3, calculateTotalAmount(computerIdsAndQuantities));

        ResultSet generatedKeys = orderStmt.executeQuery();
        if (generatedKeys.next()) {
            int orderId = generatedKeys.getInt(1);
            createOrderItems(orderId, computerIdsAndQuantities);
            updateStockQuantities(computerIdsAndQuantities);
            con.commit(); // Подтверждаем транзакцию
            return new OrderResult(true, "The order has been successfully created!");
        }
    } catch (SQLException e) {
        try {
            con.rollback(); // Откат транзакции в случае ошибки
        } catch (SQLException rollbackEx) {
            rollbackEx.printStackTrace();
            return new OrderResult(false, "Error on transaction rollback: " +
rollbackEx.getMessage());
        }
        e.printStackTrace();
        return new OrderResult(false, "Error when creating an order");
    } finally {
        try {
            con.setAutoCommit(true); // Включаем автоматическое подтверждение обратно
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    return new OrderResult(false, "Failed to create an order. Try again later");
}

```

```

private double calculateTotalAmount(Map<Integer, Integer> computerIdsAndQuantities)
throws SQLException {
    double totalAmount = 0.0;
    for (Map.Entry<Integer, Integer> entry : computerIdsAndQuantities.entrySet()) {
        int computerId = entry.getKey();
        int quantityRequested = entry.getValue();

        String priceQuery = "SELECT price FROM computers WHERE id = ?";
        try (PreparedStatement pstmt = con.prepareStatement(priceQuery)) {
            pstmt.setInt(1, computerId);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                double price = rs.getDouble("price");
                totalAmount += price * quantityRequested;
            }
        }
    }
    return totalAmount;
}

```

```

public boolean updateOrder(Order order) {
    String updateQuery = "UPDATE orders SET status = ?, comment = ? WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(updateQuery)) {
        pstmt.setString(1, order.getStatus());
        pstmt.setString(2, order.getComment());
        pstmt.setInt(3, order.getId());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
        return false;
    }
}

```

```

public boolean deleteOrder(Integer order_id) {
    String deleteQuery = "DELETE FROM orders WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(deleteQuery)) {
        pstmt.setInt(1, order_id);
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
        AlertHelper.showErrorAlert("Unknown Error. Try again");
    }
}

```

```

        return false;
    }
}

```

```

//==> ORDER ITEMS CRUD
//

```

```

=====
=====

```

```

private List<OrderItem> getOrderItems(Integer orderId) throws SQLException {
    List<OrderItem> orderItems = new ArrayList<>();

```

```

    String itemQuery = "SELECT * FROM order_items WHERE order_id = ?";

```

```

    try (PreparedStatement itemStmt = con.prepareStatement(itemQuery)) {
        itemStmt.setInt(1, orderId);
        ResultSet itemRs = itemStmt.executeQuery();

```

```

        while (itemRs.next()) {
            Integer itemId = itemRs.getInt("id");
            Integer computerId = itemRs.getInt("computer_id");
            Integer quantity = itemRs.getInt("quantity");
            BigDecimal price = new BigDecimal(itemRs.getDouble("price"));

```

```

            Computer pc = getComputerById(computerId);

```

```

            if (pc != null) {
                pc.setPrice(price);
                orderItems.add(new OrderItem(
                    itemId,
                    orderId,
                    pc,
                    quantity
                ));
            }
        }
    }
    return orderItems;
}

```

```

public List<OrderItem> getAllOrderItems() {
    List<OrderItem> orderItems = new ArrayList<>();

```

```

String itemQuery = "SELECT * FROM order_items";

try (PreparedStatement itemStmt = con.prepareStatement(itemQuery)) {
    ResultSet itemRs = itemStmt.executeQuery();

    while (itemRs.next()) {
        Integer orderId = itemRs.getInt("order_id");
        Integer itemId = itemRs.getInt("id");
        Integer computerId = itemRs.getInt("computer_id");
        Integer quantity = itemRs.getInt("quantity");
        BigDecimal price = new BigDecimal(itemRs.getDouble("price"));

        Computer pc = getComputerById(computerId);

        if (pc != null) {
            pc.setPrice(price.multiply(new BigDecimal(quantity)));
            orderItems.add(new OrderItem(
                itemId,
                orderId,
                pc,
                quantity
            ));
        }
    }
} catch (Exception ex) {
    Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    AlertHelper.showErrorAlert("Unknown Error. Try again");
}

return orderItems;
}

public boolean updateOrderItem(OrderItem item) {
    String updateQuery = "UPDATE order_items SET order_id = ?, computer_id = ?, quantity
= ? WHERE id = ?";

    try (PreparedStatement pstmt = con.prepareStatement(updateQuery)) {
        pstmt.setInt(1, item.getOrderId());
        pstmt.setInt(2, item.getComputer().getId());
        pstmt.setInt(3, item.getQuantity());
        pstmt.setInt(4, item.getId());
        return pstmt.executeUpdate() > 0;
    } catch (SQLException ex) {
        Logger.getLogger(DbConnection.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```


3.6 Разработка формы авторизации

...

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.Cursor?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<BorderPane fx:id="rootPane" maxHeight="400.0" maxWidth="600.0" minHeight="400.0"
minWidth="600.0" prefHeight="400.0" prefWidth="600.0" style="-fx-background-color:
#1e1e2e;" xmlns="http://javafx.com/javafx/23.0.1" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.shop.controllers.auth.LoginController">
    <left>
        <AnchorPane prefHeight="400.0" prefWidth="300.0" style="-fx-background-color:
#181825;" BorderPane.alignment="CENTER">
            <children>
                <Text fill="WHITE" layoutY="195.0" strokeType="OUTSIDE" strokeWidth="0.0"
text="NexusPC" textAlignment="CENTER" wrappingWidth="302.0"
AnchorPane.bottomAnchor="185.45199584960938" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="141.22">
                    <font>
                        <Font name="System Bold" size="54.0" />
                    </font>
                </Text>
                <Text fill="WHITE" layoutY="231.0" strokeType="OUTSIDE" strokeWidth="0.0"
text="Cheap and cheerful" textAlignment="CENTER" wrappingWidth="302.0"
AnchorPane.bottomAnchor="157.74" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="217.0">
                    <font>
                        <Font size="18.0" />
                    </font>
                </Text>
                <Button ellipsisString="" layoutX="14.0" layoutY="14.0" maxHeight="20.0"
maxWidth="20.0" minHeight="20.0" minWidth="20.0" mnemonicParsing="false"
```



```

onAction="#handleCloseButton" prefHeight="20.0" prefWidth="20.0" style="-fx-background-
color: #eba0ac; -fx-background-radius: 10; -fx-border-radius: 10;"
AnchorPane.leftAnchor="15.0" AnchorPane.topAnchor="15.0" />
    </children>
</AnchorPane>
</left>
<right>
    <AnchorPane prefHeight="400.0" prefWidth="300.0" style="-fx-background-color:
#1e1e2e;" BorderPane.alignment="CENTER">
        <children>
            <TextField fx:id="username" layoutX="49.0" layoutY="163.0" prefHeight="25.0"
prefWidth="202.0" promptText="Username" style="-fx-background-color: transparent; -fx-
border-color: #ffffff; -fx-border-width: 0px 0px 2px 0px; -fx-text-fill: #ffffff;" />
            <PasswordField fx:id="password" layoutX="49.0" layoutY="202.0" prefHeight="25.0"
prefWidth="202.0" promptText="Password" style="-fx-background-color: transparent; -fx-
border-color: #ffffff; -fx-border-width: 0px 0px 2px 0px; -fx-text-fill: #ffffff;" />
            <Button fx:id="loginButton" layoutX="49.0" layoutY="254.0"
mnemonicParsing="false" onAction="#login" prefHeight="40.0" prefWidth="203.0" style="-
fx-background-color: #b4befe; -fx-background-radius: 0px; -fx-cursor: hand; -fx-background-
radius: 10;" text="Login" textFill="WHITE">
                <cursor>
                    <Cursor fx:constant="OPEN_HAND" />
                </cursor>
                <font>
                    <Font name="System Bold" size="13.0" />
                </font></Button>
            <Text fill="WHITE" layoutX="75.0" layoutY="131.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="User Login" wrappingWidth="150.24869537353516">
                <font>
                    <Font name="System Bold" size="27.0" />
                </font>
            </Text>
            <Text fill="WHITE" layoutX="60.0" layoutY="312.0"
onMouseClicked="#showRegisterStage" strokeType="OUTSIDE" strokeWidth="0.0" style="-
fx-cursor: hand;" text="Don't have an account? Register" textOrigin="CENTER"
wrappingWidth="180.5459747314453">
                <cursor>
                    <Cursor fx:constant="HAND" />
                </cursor>
                <font>
                    <Font name="System Bold" size="11.0" />
                </font>
            </Text>
        </children>
    </AnchorPane>
</right>
</GridPane>
</VBox>
</Scene>
</Window>
</FXML>

```

```

    </AnchorPane>
</right>
</BorderPane>
...

```

3.7 Разработка формы регистрации

```

...
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.Cursor?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<BorderPane fx:id="rootPane" maxHeight="400.0" maxWidth="600.0" minHeight="400.0"
minWidth="600.0" prefHeight="400.0" prefWidth="600.0" style="-fx-background-color:
#1e1e2e;" xmlns="http://javafx.com/javafx/23.0.1" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.shop.controllers.auth.RegisterController">
    <left>
        <AnchorPane prefHeight="400.0" prefWidth="300.0" style="-fx-background-color:
#181825;" BorderPane.alignment="CENTER">
            <children>
                <Text fill="WHITE" layoutY="178.0" strokeType="OUTSIDE" strokeWidth="0.0"
text="NexusPC" textAlignment="CENTER" textOrigin="CENTER" wrappingWidth="302.0"
AnchorPane.bottomAnchor="185.45199584960938" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="141.2259979248047">
                    <font>
                        <Font name="System Bold" size="54.0" />
                    </font>
                </Text>
                <Text fill="WHITE" layoutY="230.0" strokeType="OUTSIDE" strokeWidth="0.0"
text="Cheap and cheerful" textAlignment="CENTER" textOrigin="CENTER"
wrappingWidth="302.0" AnchorPane.bottomAnchor="157.74199962615967"
AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
AnchorPane.topAnchor="217.0">
                    <font>
                        <Font size="18.0" />
                    </font>
                </Text>
            </children>
        </AnchorPane>
    </left>

```

```

    <Button ellipsisString="" layoutX="14.0" layoutY="14.0" maxHeight="20.0"
maxWidth="20.0" minHeight="20.0" minWidth="20.0" mnemonicParsing="false"
onAction="#handleCloseButton" prefHeight="20.0" prefWidth="20.0" style="-fx-background-
color: #eba0ac; -fx-background-radius: 10; -fx-border-radius: 10;"
AnchorPane.leftAnchor="15.0" AnchorPane.topAnchor="15.0" />
    </children>
</AnchorPane>
</left>
<right>
    <AnchorPane prefHeight="400.0" prefWidth="310.0" style="-fx-background-color:
#1e1e2e;" BorderPane.alignment="CENTER">
        <children>
            <Text fill="WHITE" layoutX="23.0" layoutY="54.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="User Registration" wrappingWidth="251.24869537353516">
                <font>
                    <Font name="System Bold" size="27.0" />
                </font>
            </Text>
            <TextField fx:id="firstName" layoutX="49.0" layoutY="76.0" prefHeight="25.0"
prefWidth="202.0" promptText="First Name" style="-fx-background-color: transparent; -fx-
border-color: #ffffff; -fx-border-width: 0px 0px 2px 0px; -fx-text-fill: #ffffff;" />
            <TextField fx:id="lastName" layoutX="49.0" layoutY="113.0" prefHeight="25.0"
prefWidth="202.0" promptText="Last Name" style="-fx-background-color: transparent; -fx-
border-color: #ffffff; -fx-border-width: 0px 0px 2px 0px; -fx-text-fill: #ffffff;" />
            <TextField fx:id="email" layoutX="49.0" layoutY="147.0" prefHeight="25.0"
prefWidth="202.0" promptText="Email" style="-fx-background-color: transparent; -fx-border-
color: #ffffff; -fx-border-width: 0px 0px 2px 0px; -fx-text-fill: #ffffff;" />
            <TextField fx:id="username" layoutX="49.0" layoutY="183.0" prefHeight="25.0"
prefWidth="202.0" promptText="Username" style="-fx-background-color: transparent; -fx-
border-color: #ffffff; -fx-border-width: 0px 0px 2px 0px; -fx-text-fill: #ffffff;" />
            <PasswordField fx:id="password" layoutX="49.0" layoutY="219.0" prefHeight="25.0"
prefWidth="202.0" promptText="Password" style="-fx-background-color: transparent; -fx-
border-color: #ffffff; -fx-border-width: 0px 0px 2px 0px; -fx-text-fill: #ffffff;" />
            <PasswordField fx:id="confirmPassword" layoutX="49.0" layoutY="254.0"
prefHeight="25.0" prefWidth="202.0" promptText="Confirm Password" style="-fx-
background-color: transparent; -fx-border-color: #ffffff; -fx-border-width: 0px 0px 2px 0px;
-fx-text-fill: #ffffff;" />
            <Button fx:id="registerButton" layoutX="51.0" layoutY="302.0"
mnemonicParsing="false" onAction="#register" prefHeight="40.0" prefWidth="201.0"
style="-fx-background-color: #b4befe; -fx-background-radius: 10px; -fx-cursor: hand;"
text="Submit" textFill="WHITE">
                <cursor>
                    <Cursor fx:constant="HAND" />
                </cursor>
            </Button>
        </children>
    </AnchorPane>
</right>
</BorderPane>
</HBox>
</VBox>
</Scene>
</Stage>
</FXML>

```

```

    <font>
      <Font name="System Bold" size="13.0" />
    </font></Button>
    <Text fill="WHITE" layoutX="65.0" layoutY="365.0"
onMouseClicked="#showLoginStage" strokeType="OUTSIDE" strokeWidth="0.0" style="-fx-
cursor: hand;" text="Already have an account? Login">
      <cursor>
        <Cursor fx:constant="HAND" />
      </cursor>
      <font>
        <Font name="System Bold" size="11.0" />
      </font>
    </Text>
  </children>
</AnchorPane>
</right>
</BorderPane>
'''

```

4. Пользовательское руководство

4.1 Страница авторизации

Для того чтобы попасть на страницу авторизации, необходимо запустить приложение. При запуске вы сразу увидите окно авторизации (Рисунок 4).

На этой странице вам будет предложено ввести логин и пароль в соответствующие поля. Поле «Логин» должно содержать ваш уникальный идентификатор, а поле «Пароль» — ваш личный пароль.

После ввода данных нажмите кнопку «Login». Если введенные данные корректны, вы будете перенаправлены на главную страницу приложения. В противном случае отобразится сообщение об ошибке.

Если у вас еще нет аккаунта, нажмите на кнопку «Регистрация» внизу формы авторизации, чтобы перейти к процессу создания нового аккаунта.

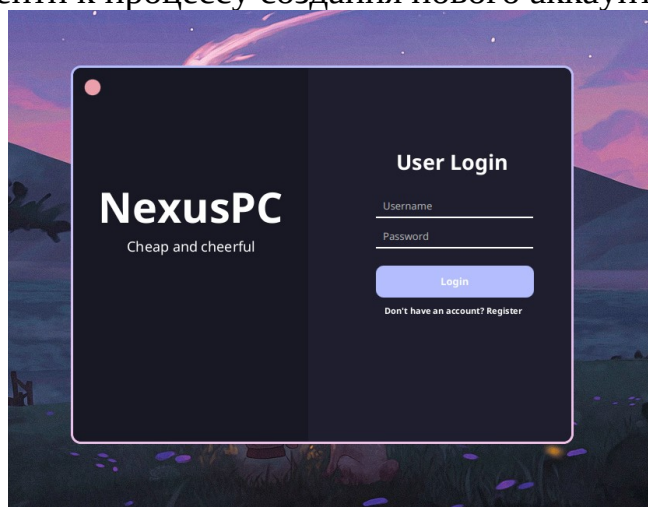


Рисунок 4. Окно авторизации

4.2 Страница регистрации

На странице регистрации (Рисунок 5) вам необходимо заполнить все обязательные поля с личной информацией, такие как имя, адрес электронной почты, логин и пароль. После заполнения всех полей нажмите кнопку «Зарегистрироваться». При успешной регистрации вы получите уведомление и сможете вернуться на страницу авторизации для входа в систему.

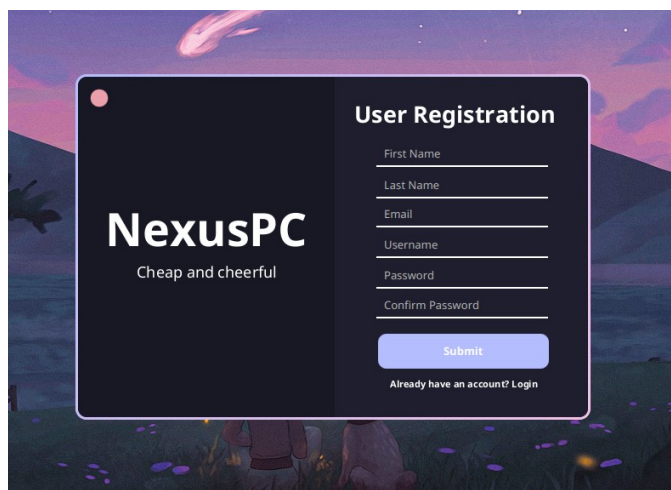


Рисунок 5. Окно регистрации

4.3 Главная страница

После успешной авторизации вы попадаете на главную страницу приложения (Рисунок 6). Слева вы можете наблюдать основное меню с кнопками перехода на различные страницы приложения.

Справа, в качестве стартовой страницы, представлено меню каталога товаров, где отображаются карточки товаров с их фотографиями, названиями и описаниями. Вы можете просмотреть доступные товары и выбрать интересующий вас компьютер, кликнув на его карточку.

Сверху по центру вы можете наблюдать название страницы. Слева – кнопки для управления самим окном, а справа – кнопку «logout», нажав на которую Вы выйдете из аккаунта.

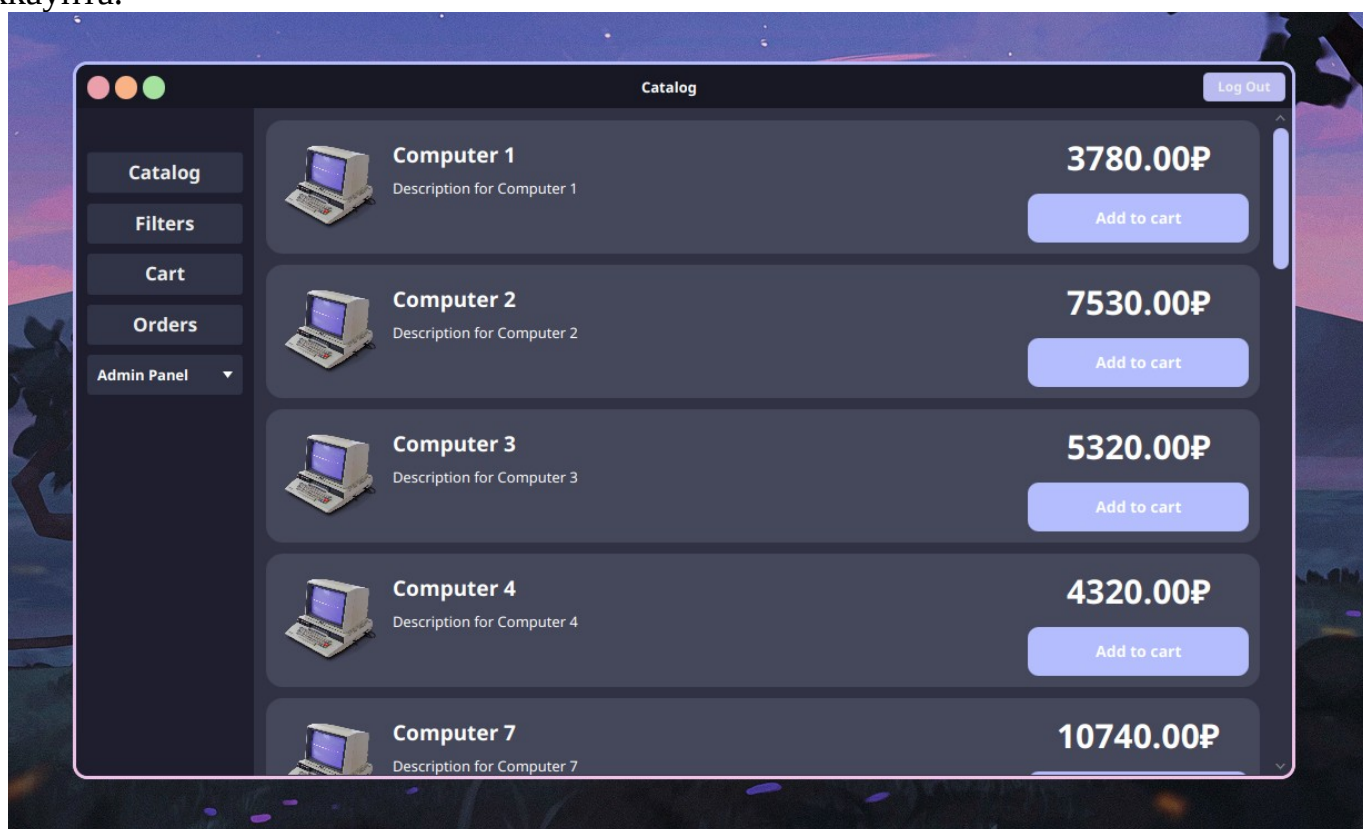


Рисунок 6. Главная страница (Каталог товаров)

4.4 Страница каталога товаров

На странице каталога товаров (Рисунок 6) представлены все доступные товары в виде карточек с их фотографиями, названиями, описаниями, ценой и кнопкой «Add to cart», что в переводе означает «Добавить в корзину».

Вы можете нажать на область названия и описания товара, чтобы посмотреть его подробное описание.

Так-же вы можете использовать фильтры для сортировки товаров по различным критериям, таким как цена или производитель. Для этого перейдите на страницу фильтров, нажав соответствующую кнопку в меню слева.

4.5 Страница информации о товаре

Когда вы нажмёте на область названия и описания товара в каталоге, вы будете перенаправлены на страницу информации о товаре (Рисунок 7). Здесь вы найдёте детальную информацию о выбранном компьютере, включая характеристики всех комплектующих, ссылки на официальные сайты производителя, и цену. Чтобы добавить товар в корзину, нажмите кнопку «Add to cart».

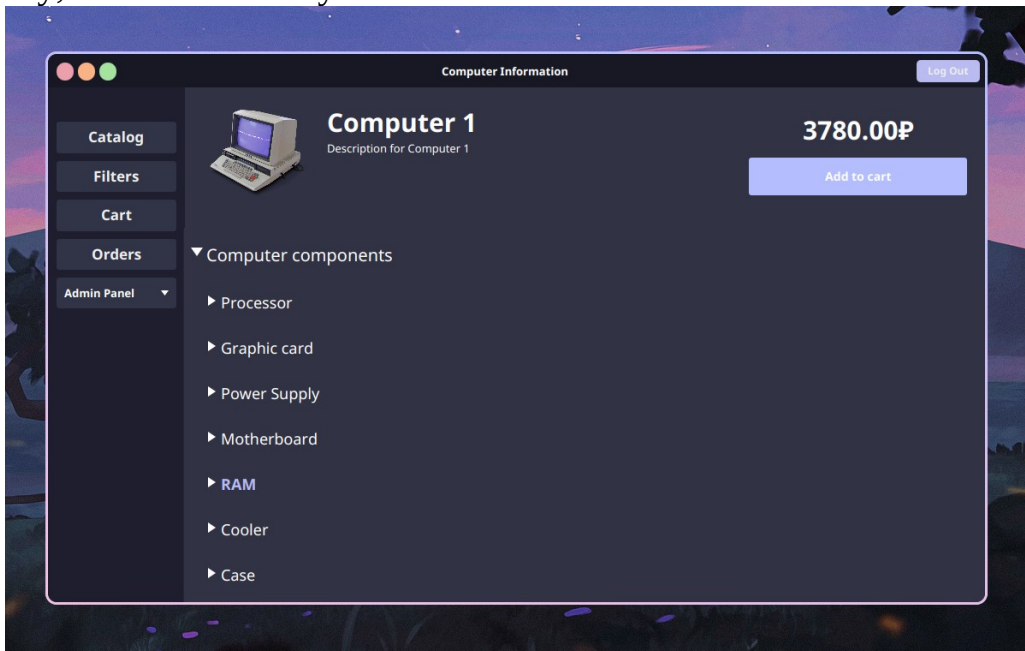


Рисунок 7. Подробная информация о компьютере

4.6 Страница фильтров

На странице фильтров (Рисунок 8) вы можете задать параметры для сортировки товаров. Просто выберите нужные характеристики, и они применятся автоматически. Для сброса фильтров используйте кнопку «Reset filters»

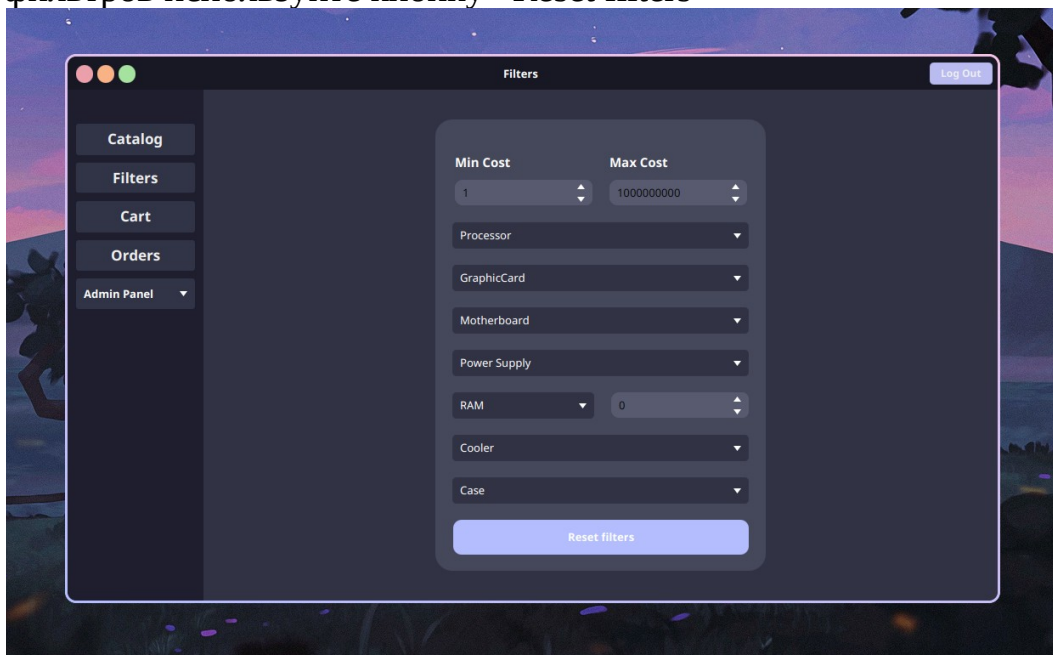


Рисунок 8. Фильтры

4.7 Страница корзины товаров

На странице корзины товаров (Рисунок 9) отображаются все добавленные вами товары. Здесь вы можете изменить количество каждого товара или удалить его из корзины. В верхней части страницы будет показана итоговая стоимость заказа. Чтобы оформить заказ, нажмите кнопку «Make an order». Если у вас есть роль администратора, на этой странице также будет доступно текстовое поле для указания контактов пользователя, которому принадлежит оформляемый заказ.

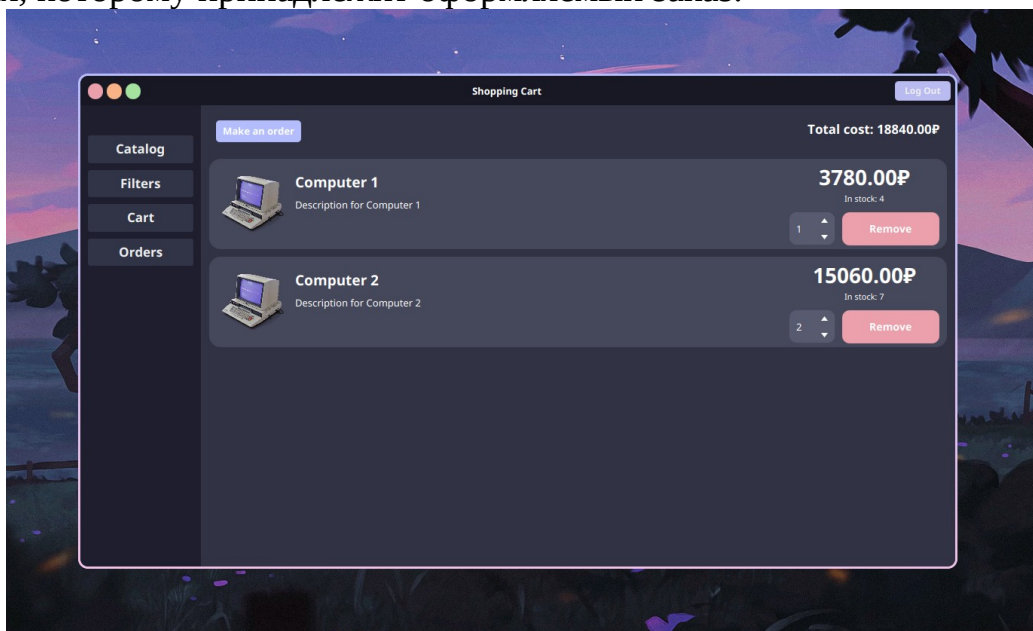


Рисунок 9. Корзина товаров

4.8 Страница заказов

После оформления заказа вы будете перенаправлены на страницу заказов (Рисунок 10). Здесь вы можете просмотреть историю своих заказов, включая дату оформления, итоговую цену и статус выполнения заказа. Каждый заказ будет содержать краткие сведения о приобретенных компьютерах.

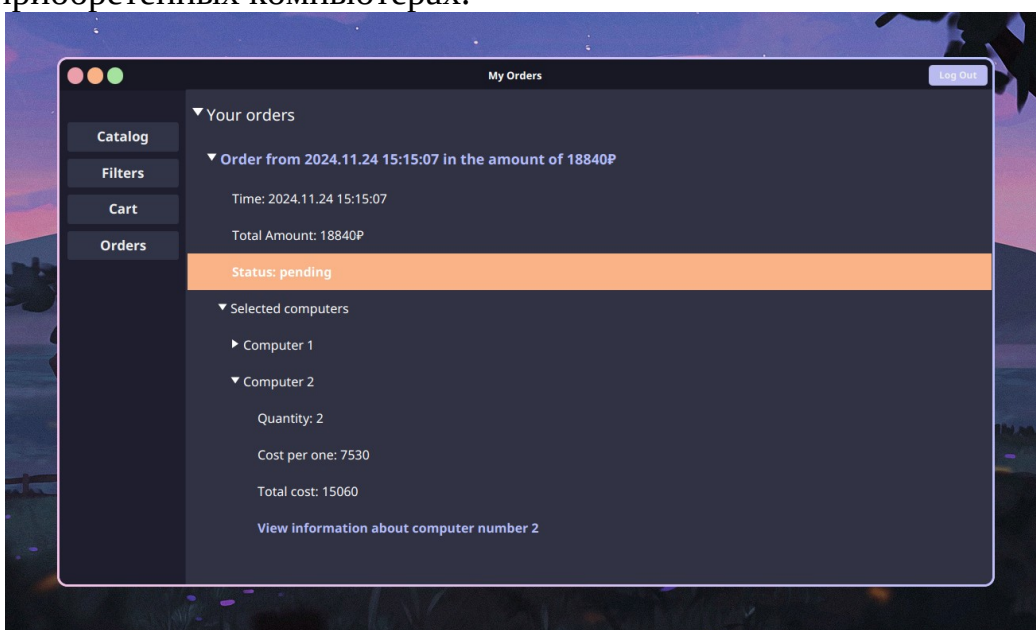


Рисунок 10. Страница заказов

4.9 Админ панель

Если у вас есть права администратора, вы сможете получить доступ к админ панели (Рисунок 11), нажав соответствующую кнопку в меню, и выбрав нужную таблицу.

В админ панели доступны таблицы с данными о пользователях, заказах, комплектующих и компьютерах.

Вы можете добавлять новые записи или редактировать существующие с помощью кнопок управления, расположенных в верхней части страницы:

- **Add row** – добавление новой строки в таблицу.
- **Delete row** – удаление выделенной строки.
- **Save changes** – сохранение всех внесенных изменений.
- **Refresh** – перезагрузка страницы с актуальными данными из базы данных.

Name	Description	Price	Processor ID	Graphic Card ID	Motherboard ID	RAM ID	RAMs Count	Power Supply ID	Cooler ID	Case ID	Image Url	Stock Quantity
Computer 3	Description for Computer 3	5320.00	2	1	5	2	2	2	5	5	https://example.com/computer/3	0
Computer 4	Description for Computer 4	4320.00	5	5	2	3	4	5	5	5	https://example.com/computer/4	10
Computer 7	Description for Computer 7	10740.00	4	4	1	5	3	4	1	5	https://example.com/computer/7	10
Computer 8	Description for Computer 8	2630.00	1	5	4	3	1	2	2	1	https://example.com/computer/8	3
Computer 9	Description for Computer 9	4840.00	1	3	3	2	3	2	4	5	https://example.com/computer/9	6
Computer 10	Description for Computer 10	7720.00	2	2	3	3	1	3	3	2	https://example.com/computer/10	0
Computer 11	Description for Computer 11	8290.00	2	4	3	5	1	2	4	4	https://example.com/computer/11	19
Computer 12	Description for Computer 12	10080.00	3	3	3	5	1	1	1	4	https://example.com/computer/12	6
Computer 13	Description for Computer 13	2220.00	3	4	4	1	3	2	3	1	https://example.com/computer/13	10
Computer 14	Description for Computer 14	3980.00	3	3	2	3	3	3	4	3	https://example.com/computer/14	14
Computer 15	Description for Computer 15	10290.00	1	1	3	1	4	4	3	2	https://example.com/computer/15	1
Computer 16	Description for Computer 16	10660.00	1	3	2	2	1	5	2	2	https://example.com/computer/16	16
Computer 17	Description for Computer 17	4920.00	2	1	5	5	2	4	2	3	https://example.com/computer/17	16
Computer 18	Description for Computer 18	4480.00	2	4	4	2	4	3	4	2	https://example.com/computer/18	2
Computer 19	Description for Computer 19	6500.00	1	3	2	1	3	1	1	5	https://example.com/computer/19	14
Computer 20	Description for Computer 20	5080.00	1	1	3	4	2	3	4	1	https://example.com/computer/20	12
Computer 5	Description for Computer 5	1000.00	3	1	3	1	1	3	2	5	https://example.com/computer/5	8
Computer 6	Description for Computer 6	9410.00	2	4	2	1	2	1	5	5	https://example.com/computer/6	3
Computer 1	Description for Computer 1	3780.00	3	4	4	1	1	3	4	5	https://example.com/computer/1	3
Computer 2	Description for Computer 2	7530.00	1	2	1	4	4	3	1	3	https://example.com/computer/2	5

Рисунок 11. Панель администратора (Таблица «Users»)

Заключение

В условиях стремительно развивающегося рынка компьютерной техники необходимость в удобных и эффективных инструментах для покупки готовых компьютеров становится все более актуальной. Разработанное приложение для покупки компьютеров предоставляет пользователям возможность легко и быстро находить и приобретать готовые решения, что значительно упрощает процесс выбора и покупки.

В ходе работы была создана информационная система, которая включает в себя функционал для регистрации пользователей, авторизации, просмотра каталога товаров, а также оформления заказов. Система была разработана с учетом потребностей пользователей, что позволяет обеспечить интуитивно понятный интерфейс и удобную навигацию между страницами приложения.

Основные задачи, поставленные перед проектом, включали анализ требований целевой аудитории, проектирование структуры приложения, реализацию функционала и тестирование системы на предмет ее работоспособности и удобства использования. Результатом работы стало полностью функционирующее настольное приложение, которое соответствует всем заявленным требованиям.

Создание данного приложения не только улучшает процесс покупки готовых компьютеров, но и способствует повышению удовлетворенности клиентов за счет автоматизации процессов учета заказов и управления данными. В результате пользователи получают доступ к актуальной информации о товарах и могут быстро оформлять заказы без лишних усилий.

В процессе выполнения проекта также была подготовлена документация, которая поможет пользователям ознакомиться с основными функциями системы и эффективно использовать приложение. Таким образом, разработанное решение отвечает современным требованиям рынка и может стать надежным инструментом для пользователей при покупке компьютерной техники.

Список литературы

1. Документация sql URL: <https://dev.mysql.com/doc/> (дата обращения: 06.11.2024).
Режим доступа: свободный
2. Документация Java URL: <https://docs.oracle.com/en/java/> (дата обращения: 06.11.2024). Режим доступа: свободный
3. Документация JavaFX URL: <https://www.oracle.com/java/technologies/javase/javafx-docs.html> (дата обращения 06.11.2024).