**Разработка базы данных «Интернет-магазин бытовой техники и электроники» и клиентского приложения для доступа к ней**

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc123035148)

[1. Описание реляционной модели данных 4](#_Toc123035149)

[2. Запросы к базе данных 7](#_Toc123035150)

[3. Создание хранимых функций и триггеров 12](#_Toc123035151)

[4. Клиентское приложение 15](#_Toc123035152)

[4.1 Обзор клиентского приложения 15](#_Toc123035153)

[4.2 SQL – инъекции 18](#_Toc123035154)

[ЗАКЛЮЧЕНИЕ 19](#_Toc123035155)

[СПИСОК ЛИТЕРАТУРЫ 21](#_Toc123035156)

[ПРИЛОЖЕНИЕ 1. СЦЕНАРИЙ СОЗДАНИЯ БАЗЫ ДАННЫХ 22](#_Toc123035157)

[ПРИЛОЖЕНИЕ 2. ПРОГРАММНЫЙ КОД 25](#_Toc123035158)

# **ВВЕДЕНИЕ**

Предметной областью настоящей работы является интернет-магазин бытовой техники и электроники. Магазин ведёт учет товаров, принимает заказы, собирает информацию о каждом устройстве и предоставляет всю информацию в удобном для пользователя форме. Для автоматизации и облегчения этих процессов, предоставления возможности людям искать и покупать желаемые им товары, было решено разработать web-приложение, которое состоит из клиентской и серверной части. В клиентской части происходит взаимодействие с пользователем и работником интернет-магазина. Пользователь имеет возможность просматривать необходимую информацию о товаре, фильтровать по нужным ему категориям и оставлять отзывы о продукте. У работника есть отдельная страница для редактирования данных. Серверная часть отвечает за обработку и хранения данных, полученных от клиентской части.

1. Описание реляционной модели данных

Схема данных приведено на рисунке 1.

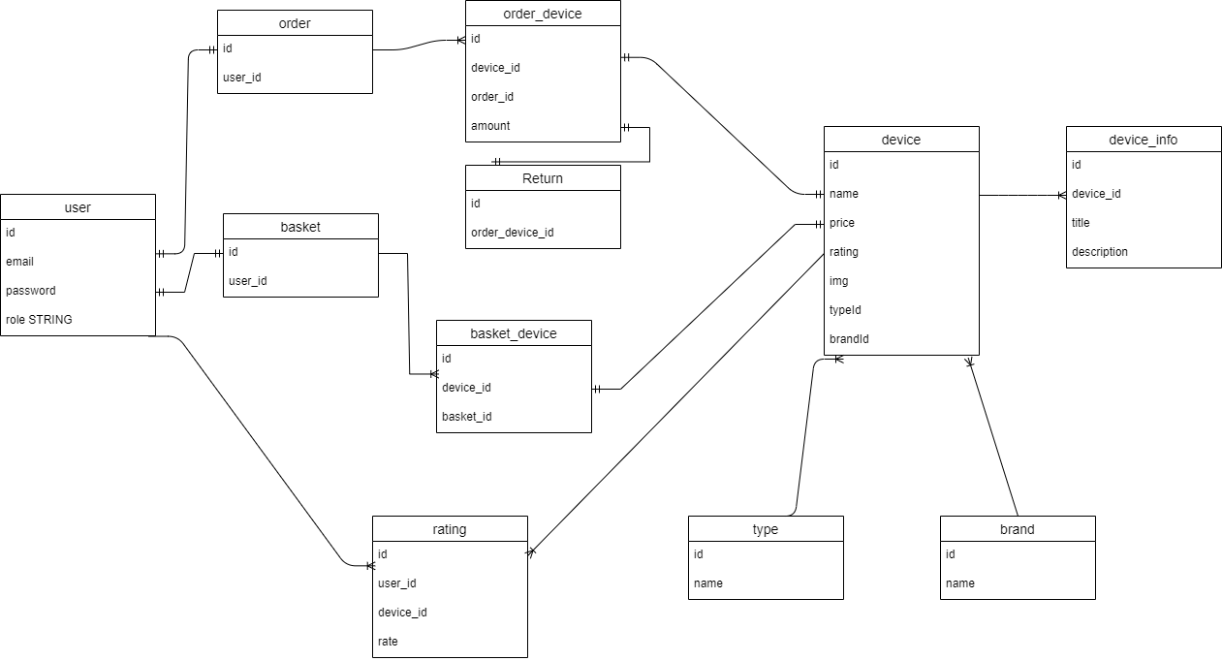


Рисунок 1 – Схема данных

Таблица «device» предназначена для хранения устройств и содержит столбцы:

* name – имя устройства
* price – цена устройства
* rating – рейтинг устройства
* img – изображение устройства
* typeId – id соответствующего типа устройства
* brandId – id соответствующего бренда устройства
* id – id устройства

Таблица «device\_info» предназначена для хранения описания устройства и содержит столбцы:

* title – заголовок описания
* description – само описание
* id – id описания
* device\_id – id устройства

Таблица «type» предназначена для хранения типа устройств:

* id – id типа
* name – название типа

Таблица «brand» предназначена для хранения бренда устройств:

* id – id бренда
* name – название бренда

Таблица «rating» предназначена для хранения рейтинга устройств:

* id – id рейтинга
* user\_id – id пользователя
* device\_id – id устройства
* rate – рейтинг

Таблица «rating» предназначена для хранения пользователей:

* id – id пользователя
* email – email пользователя
* password – пароль пользователя
* role – роль пользователя (ADMIN / USER)

Таблица «basket» предназначена для хранения корзины пользователя:

* id – id корзины
* user\_id – id пользователя

Таблица «basket\_device» предназначена для хранения устройств помещенных в корзину:

* basket\_id – id корзины
* device\_id – id устройства

Таблица «order» предназначена для хранения заказа пользователя:

* id – id корзины
* user\_id – id пользователя

Таблица «order\_device» предназначена для хранения заказанных устройств:

* device\_id – id устройства
* order\_id – id заказа
* amount – количество устройств

Таблица «return» предназначена для хранения информации о возвратах устройств:

* id – id возврата
* order\_device\_id – id соответствующего заказа

1. Запросы к базе данных

Запросы — это объект базы данных, который служит для извлечения данных из таблиц и предоставления их пользователю в удобном виде. Особенность запросов состоит в том, что они черпают данные из базовых таблиц и создают на их основе временную таблицу. Применение запросов позволяет избежать дублирования данных в таблицах и обеспечивает максимальную гибкость при поиске и отображении данных в базе данных. В качестве СУБД был использован PostgreSQL [1], а запросы были написаны в графическом клиенте pgAdmin. Далее будут приведены примеры написания различных запросов реляционной алгебры.

Селекция:

SELECT \* FROM device\_infos WHERE title = 'Экран'

Этот запрос позволяет получить описания устройств, у которых заголовок описания равен «Экран».

SELECT \* FROM devices WHERE price > 80000

Этот запрос позволяет получить устройства, цена которых больше 80000.

SELECT \* FROM users WHERE role = 'USER'

Данный запрос позволяет получить всех обычных пользователей.

Проекция:

SELECT name FROM brands;

Этот запрос позволяет получить название брендов.

SELECT rating FROM devices;

Этот запрос позволяет получить рейтинг устройств.

SELECT name FROM types;

Этот запрос позволяет получить типы устройств.

Соединение:

SELECT \* FROM devices, types

Этот запрос возвращает всю информацию из таблиц устройств и типов устройств.

SELECT \* FROM devices, brands

Этот запрос возвращает всю информацию из таблиц устройств и брендов устройств.

SELECT \* FROM users, ratings

Этот запрос возвращает всю информацию из таблиц пользователей и рейтингов.

Декартово произведение:

SELECT devices.name, devices.price, brands.name  FROM devices, brands

WHERE brands.id = devices."brandId" AND devices.price > 80000

Этот запрос позволяет получить название устройства, его цену и название бренда, где цена устройства больше 80000.

SELECT devices.name, devices.price, types.name  FROM devices, types

WHERE types.id = devices."typeId" AND devices.price < 50000

Этот запрос позволяет получить название устройства, его цену тип, где цена устройства меньше 50000.

SELECT users.email, ratings.rate FROM users, ratings

WHERE users.id = ratings."userId"

Этот запрос позволяет получить email пользователя и все рейтинги которые он поставил устройствам.

Группировка:

SELECT brands.name, COUNT(\*) AS amount FROM brands

JOIN devices ON devices."brandId" = brands.Id

GROUP BY brands.name

Этот запрос позволяет получить количество устройств определенного бренда.

SELECT types.name, COUNT(\*) AS amount FROM types

JOIN devices ON devices."typeId" = types.Id

GROUP BY types.name

Этот запрос позволяет получить количество устройств определенного типа.

SELECT users.email, ratings."userId", COUNT(\*) AS ratings\_amount FROM ratings

JOIN users ON users.id = ratings."userId"

GROUP BY ratings."userId", users.email HAVING COUNT(\*) = 1

Этот запрос позволяет получить пользователей, которые поставили рейтинг только одному устройству.

Объединение:

SELECT devices.name, brands.name FROM devices

JOIN brands ON brands.id = devices."brandId"

WHERE brands.name = 'LG' OR brands.name = 'Xiaomi'

Данный запрос позволяет получить название устройства и его бренд, где бренд равен LG или Xiaomi.

SELECT devices.name, device\_infos.title FROM device\_infos

JOIN devices ON device\_infos."deviceId" = devices.id

WHERE device\_infos.title = 'Экран' OR device\_infos.title = 'Процессор'

Данный запрос позволяет получить название устройства и заголовок его описания, где заголовок равен «Экран» или «Процессор».

SELECT devices.name, types.name FROM devices

JOIN types ON types.id = devices."typeId"

WHERE types.name = 'Холодильники' OR types.name = 'Смартфоны'

Данный запрос позволяет получить название устройства и его тип, где тип устройства равняется холодильнику или смартфону.

Пересечение:

SELECT name FROM devices

JOIN order\_devices ON devices.id = order\_devices."deviceId"

WHERE order\_devices.amount = 2 AND EXISTS (SELECT \* FROM order\_devices as od

WHERE od."deviceId" = order\_devices."deviceId" AND od.amount = 1)

Этот запрос позволяет получить название устройства, который заказывали в количестве 1 и 2.

SELECT devices.name FROM devices

JOIN types ON devices."typeId" = types.id

JOIN brands ON devices."brandId" = brands.id

WHERE types.name = 'Смартфоны' AND brands.name = 'Samsung'

Этот запрос позволяет получить все смартфоны бренда Samsung.

SELECT users.email, users.id FROM users

JOIN orders ON users.id = orders."userId"

JOIN order\_devices ON orders.id = order\_devices."orderId"

WHERE order\_devices."deviceId" = 33

AND EXISTS (SELECT \* FROM order\_devices as od

WHERE od."orderId" = order\_devices."orderId" AND od."deviceId" = 40)

Данный запрос позволяет получить пользователя, который заказал устройство под id 33 и 40.

Разность:

SELECT devices.name, ratings.rate FROM devices

JOIN ratings ON ratings."deviceId" = devices.id

WHERE ratings.rate = '4' AND NOT EXISTS (SELECT \* FROM ratings as r

WHERE r."deviceId" = devices.id AND r.rate = '3' )

Этот запрос позволяет получить название устройства и его рейтинг, где рейтинг устройства равняется 4 и не равняется 3.

SELECT devices.name, brands.name FROM devices

JOIN types ON devices."typeId" = types.id

JOIN brands ON devices."brandId" = brands.id

WHERE types.name = 'Смартфоны' AND brands.name != 'Samsung'

Этот запрос позволяет получить все смартфоны, бренд которых не равняется Samsung.

SELECT users.email, users.id FROM users

JOIN orders ON users.id = orders."userId"

JOIN order\_devices ON orders.id = order\_devices."orderId"

WHERE order\_devices."deviceId" = 42

AND

NOT EXISTS (SELECT \* FROM order\_devices as od

WHERE od."orderId" = order\_devices."orderId" AND od."deviceId" = 47)

Этот запрос позволяет получить пользователей, которые заказали устройство под id 42 но не под id 47.

1. Создание хранимых функций и триггеров

Для обеспечения правильной работы серверной части проекта были реализованы по две хранимые и триггерные функции, ответственные за проверку вводимых данных и бизнес-логику приложения.

Первая функция предполагается использовать для классификации устройств по их рейтингу.

CREATE FUNCTION GetRating (integer) RETURNS text AS $$

DECLARE res text;

BEGIN

    IF $1 BETWEEN 1 AND 2 THEN

        SELECT 'УЖАСНЫЙ' INTO res;

    ELSE IF $1 BETWEEN 2 AND 3 THEN

        SELECT 'ПЛОХОЙ' INTO res;

    ELSE IF $1 BETWEEN 3 AND 4 THEN

        SELECT 'ХОРОШИЙ' INTO res;

    ELSE SELECT 'ОТЛИЧНЫЙ' INTO res;

    END IF; END IF; END IF ;

    RETURN res;

END;

$$ LANGUAGE plpgsql

Вторая функция используется для проверки существования устройства и пользователя при вводе информации в таблицу рейтингов.

CREATE FUNCTION IsCorrect(integer, integer)

RETURNS BOOLEAN AS $$

BEGIN

      RETURN EXISTS (SELECT \* FROM ratings

                 JOIN users ON users.id = ratings."userId"

                 JOIN devices ON devices.id = ratings."deviceId"

                 WHERE devices.id = $1 AND users.id = $2);

END;

$$ LANGUAGE plpgsql;

Первый триггер отвечает за создание заказа только тем пользователя, которые заказали одинаковые устройства не больше 39 штук. Он проверяет количество устройств, что они находятся в диапазоне от 0 до 39 включительно, до добавления их в таблицу. Если количество устройств выходит за этот диапазон будет выводиться ошибка корректности вводимых данных.

CREATE FUNCTION trigger\_order\_devices\_insert() RETURNS trigger AS $$

BEGIN

  IF (NEW.amount > 0 and NEW.amount < 40)

      THEN

      ELSE

         -- генерация исключительной ситуации

         RAISE EXCEPTION 'Ошибка корректности данных';

      END IF;

   RETURN NEW;

END;

$$ LANGUAGE  plpgsql;

CREATE TRIGGER tr\_order\_devices\_insert

BEFORE INSERT ON "order\_devices" FOR EACH ROW

EXECUTE PROCEDURE trigger\_order\_devices\_insert();

Второй триггер после введения данных в таблицу рейтингов меняет среднее значение рейтинга соответствующего устройства. Он, с помощью ранее описанной функции, проверяет корректность данных и изменяет значение в столбце rating соответствующего устройства. В случае ошибки введенных данных будет генерироваться ошибка.

CREATE FUNCTION trigger\_results\_insert() RETURNS trigger AS $$

BEGIN

  IF IsCorrect(NEW."deviceId", NEW."userId")

      THEN

        UPDATE devices SET rating = (select avg(rate) from ratings

                              where ratings."deviceId" = NEW."deviceId")

                   WHERE devices.id = NEW."deviceId";

      ELSE

         -- генерация исключительной ситуации

         RAISE EXCEPTION 'Ошибка корректности данных';

      END IF;

   RETURN NEW;

END;

$$ LANGUAGE  plpgsql;

CREATE TRIGGER tr\_results\_insert

AFTER INSERT ON "ratings" FOR EACH ROW

EXECUTE PROCEDURE trigger\_results\_insert();

1. Клиентское приложение

## Обзор клиентского приложения

Для работы с базой данных было создано web-приложение с помощью React [2] и Node.js. Разграничение прав доступа реализовано с помощью пользовательских ролей. В зависимости от прав пользователя текущей сессии происходит предоставление доступа к различному функционалу приложения.

Главная страница приложения, отображающая товары выглядит следующим образом: есть навигационные поля для фильтрации товаров по названиям брендов, по типу устройства. Имеется кнопка для авторизации. Все это можно увидеть на рисунке 2.

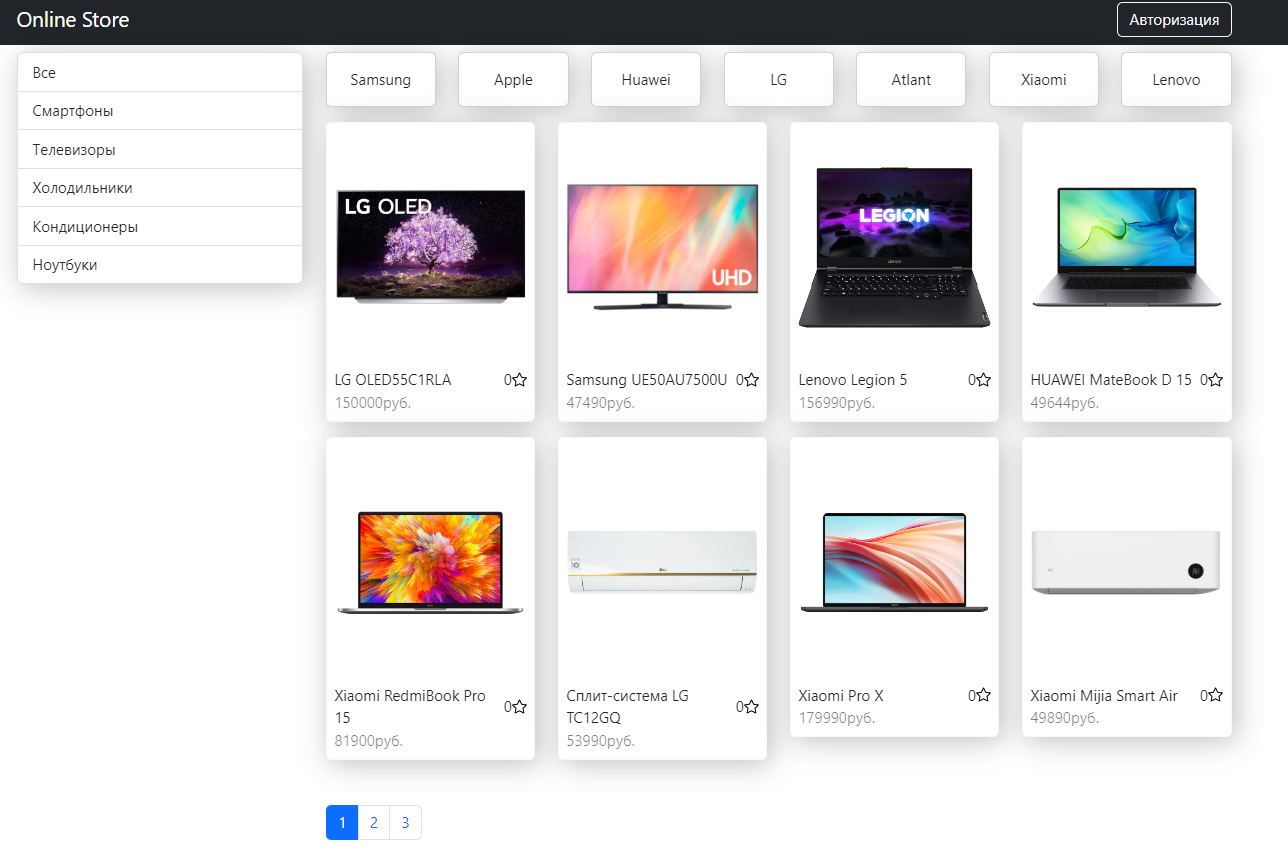


Рисунок – Главная страница

Для авторизации необходимо перейти на соответствующую страницу нажав на кнопку «Авторизация». Затем заполнить все необходимые данные и нажать на кнопку «Войти», который можно увидеть на рисунке 3, и откроется главная страница.

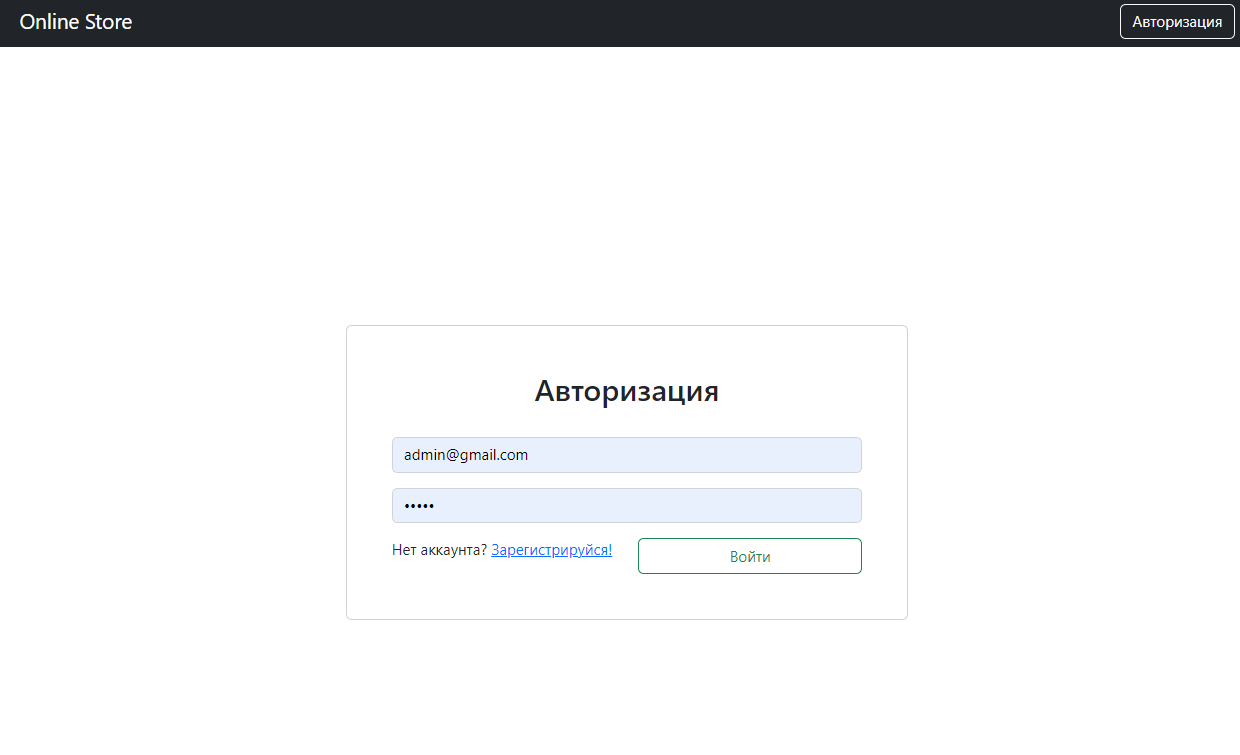


Рисунок – Авторизация

Для получения подробной информации об устройстве можно нажать на изображение товара и откроется его страница как на рисунке 4. Тут же можно и добавить устройство в корзину.

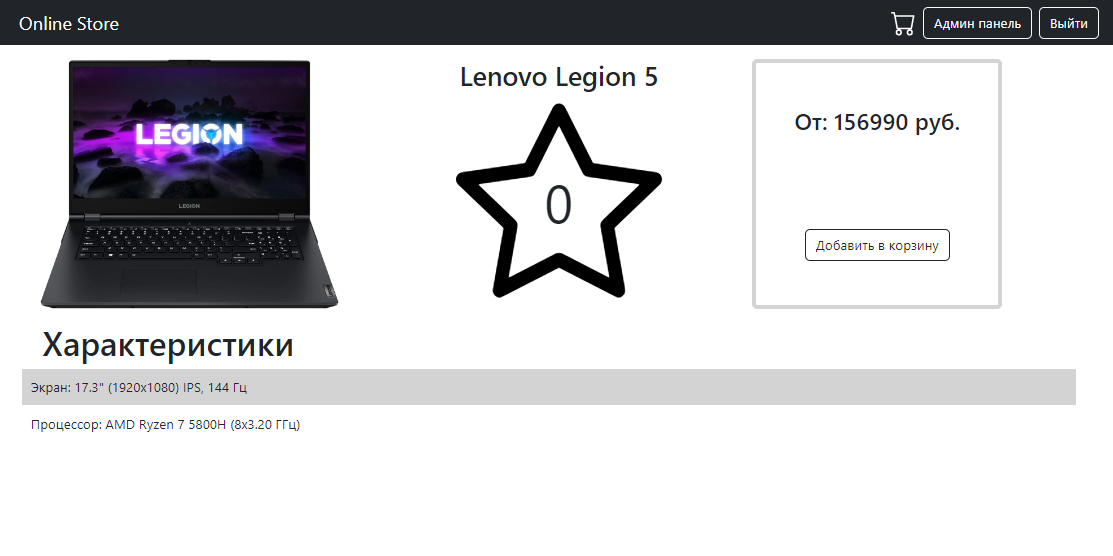


Рисунок – Страница товара

Если нажать на значок корзины, то откроется корзина пользователя, которую можно увидеть на рисунке 5.

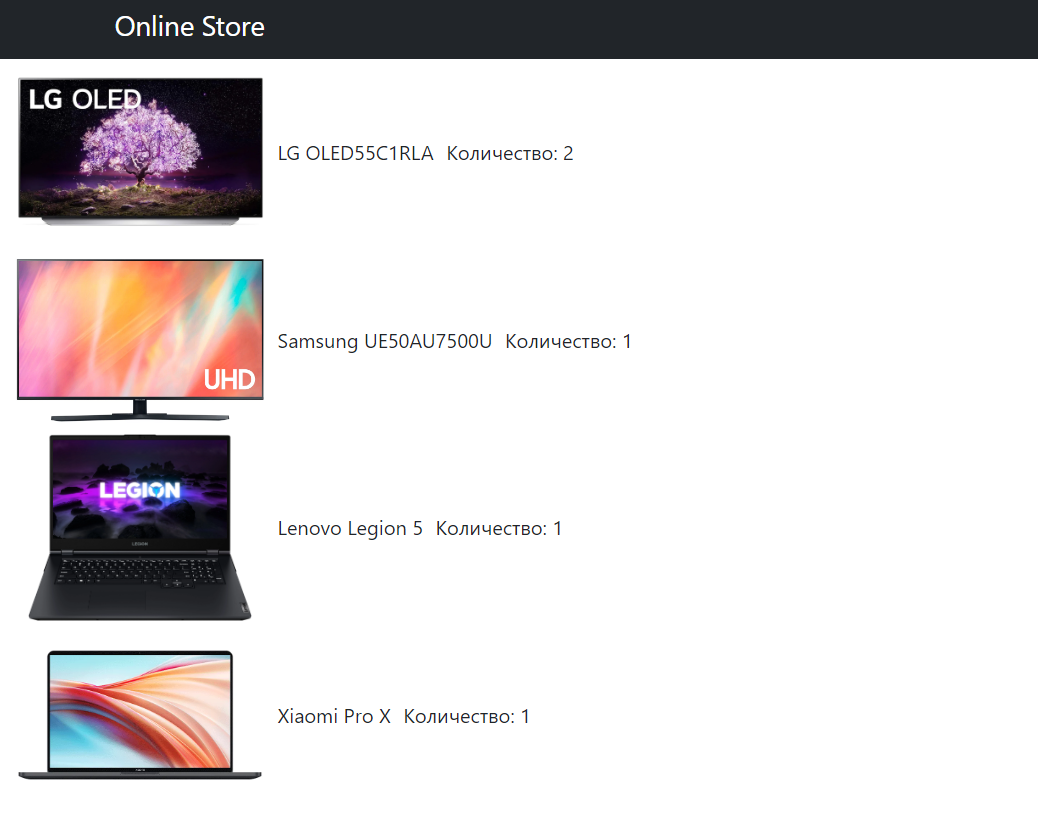


Рисунок – Корзина пользователя

У администратора склада есть возможность редактирования устройств, типов устройств и брендов, как на рисунке 6.

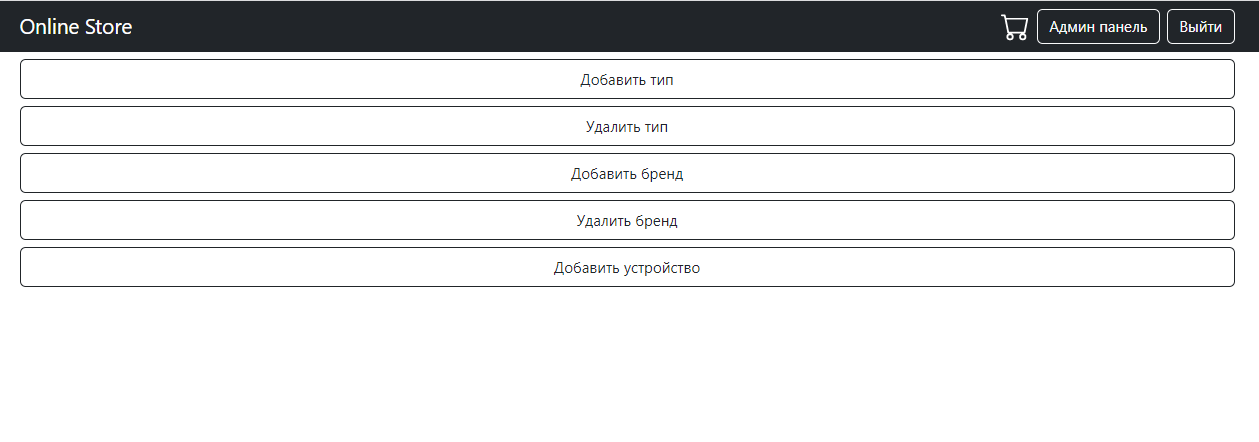


Рисунок – Панель администратора

## 4.1 SQL – инъекции

Внедрение SQL-кода — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода.

Внедрение SQL, в зависимости от типа используемой СУБД и условий внедрения, может дать возможность атакующему выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

Атака типа внедрения SQL может быть возможна из-за некорректной обработки входных данных, используемых в SQL-запросах.

Разработчик прикладных программ, работающих с базами данных, должен знать о таких уязвимостях и принимать меры противодействия внедрению SQL.

В данном проекте использовалась ORM Sequalize для написания пользовательских запросов. Он избегает замен, что позволяет обходить проблемы, лежащей в основе атак SQL-инъекций: неэкранированные строки. Он также поддерживает параметры привязки при использовании SQLite или PostgreSQL, что дополнительно снижает риск, отправляя параметры в базу данных отдельно к запросу.

ЗАКЛЮЧЕНИЕ

В ходе работы было изучено множество технологий разработки клиент-серверных приложений. Большая часть работы приходится на изучение языка запросов SQL и СУБД PostgreSQL. А также удалось познакомиться с такими инструментами как: Postman и Figma. С клиентской части были изучены: языки разметок и стилей – HTML и CSS, JavaScript-библиотека для создания пользовательских интерфейсов React, который благодаря эффективному взаимодействию с DOM-деревом ускоряет работу нашего приложения, платформа Node.js для разработки REST API. Конечным результатом можно считать успешное освоение указанных технологий, а также созданное приложение, которое отвечает исходным требованиям по хранению и обработке данных.

За время выполнения курсовой работы были реализованы следующие компетенции:

|  |  |  |
| --- | --- | --- |
| Шифр  компетенции | Расшифровка  приобретаемой компетенции | Расшифровка освоения компетенции |
| ОПК-2 | Способен применять компьютерные/суперкомпьютерные методы, современное программное обеспечение, в том числе отечественного происхождения, для решения задач профессиональной деятельности | Изучены средства СУБД, PostgreSQL и средства разработки web-приложений JavaScript, React, Node.js |
| ОПК-3 | Способен к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям | Изучена реляционная модель баз данных и алгоритмы работы клиентского приложения этой моделью. |
| ОПК-5 | Способен инсталлировать и сопровождать программное обеспечение информационных систем и баз данных, в том числе отечественного происхождения, с учетом информационной безопасности | Изучены и применены средства СУБД, PostgreSQL и средства разработки web-приложений JavaScript, React, Node.js |

СПИСОК ЛИТЕРАТУРЫ

1. PostgreSQL 15.1 Documentation [Электронный ресурс]. URL: https://www.postgresql.org/docs/ (дата обращения 12.11.2022)
2. JavaScript-библиотека React [Электронный ресурс]. URL: https://ru.react.js.org/?ref=dtf.ru (дата обращения 12.12.2022)

ПРИЛОЖЕНИЕ 1. СЦЕНАРИЙ СОЗДАНИЯ БАЗЫ ДАННЫХ

create table users  
(  
 id serial,  
 email varchar(255),  
 password varchar(255),  
 role varchar(255),  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone  
);

create table types  
(  
 id serial,  
 name varchar(255),  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone  
);

create table returns  
(  
 id serial,  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone,  
 "orderDeviceId" integer  
);

create table ratings  
(  
 id serial,  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone,  
 "userId" integer,  
 "deviceId" integer,  
 rate integer  
);

create table orders  
(  
 id serial,  
 date date,  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone,  
 "userId" integer  
);

create table order\_devices  
(  
 id serial,  
 amount integer,  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone,  
 "orderId" integer,  
 "deviceId" integer  
);

create table devices  
(  
 id serial,  
 name varchar(255),  
 price integer,  
 rating real,  
 img varchar(255),  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone,  
 "typeId" integer,  
 "brandId" integer  
);

create table device\_infos  
(  
 id serial,  
 title varchar(255),  
 description varchar(255),  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone,  
 "deviceId" integer  
);

create table brands  
(  
 id serial,  
 name varchar(255),  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone  
);

create table baskets  
(  
 id serial,  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone,  
 "userId" integer  
);

create table basket\_devices  
(  
 id serial,  
 "createdAt" timestamp with time zone,  
 "updatedAt" timestamp with time zone,  
 "basketId" integer,  
 "deviceId" integer,  
 amount integer  
);

ПРИЛОЖЕНИЕ 2. ПРОГРАММНЫЙ КОД

<https://github.com/Ka7im/Online_Store>

// DevicePage.js

import React, { useEffect } from 'react';

import { Col, Container, Image, Row, Card, Button } from 'react-bootstrap';

import star from '../assets/star.svg';

import { useParams } from 'react-router-dom';

import { useState } from 'react';

import { fetchOneDevice } from '../api/axios/deviceApi';

import {

    createDeviceInBasket,

    increaseDeviceInBasket,

    decreaseDeviceInBasket,

    getDevicesInBasket,

} from '../api/deviceSlice';

import { shallowEqual, useDispatch, useSelector } from 'react-redux';

const DevicePage = () => {

    const dispatch = useDispatch();

    const [device, setDevise] = useState({ info: [] });

    const { id } = useParams();

    const amount = useSelector(

        (state) =>

            state.device.basket.find((item) => {

                return item.id == id;

            })?.amount

    );

    useEffect(() => {

        fetchOneDevice(id).then((data) => setDevise(data));

        dispatch(getDevicesInBasket());

    }, []);

    return (

        <Container className='mt-3'>

            <Row>

                <Col md={4}>

                    <Image

                        height={300}

                        width={410}

                        style={{ objectFit: 'contain' }}

                        src={'http://localhost:5000/' + device.img}

                    />

                </Col>

                <Col md={4}>

                    <Row className='d-flex flex-column align-items-center'>

                        <h2 style={{ textAlign: 'center' }}>{device.name}</h2>

                        <div className='d-flex align-items-center justify-content-center'>

                            <Image src={star} width={248} height={248} />

                            <div style={{ position: 'absolute', fontSize: 64 }}>

                                {device.rating}

                            </div>

                        </div>

                    </Row>

                </Col>

                <Col md={4}>

                    <Card

                        className='d-flex flex-column align-items-center justify-content-around'

                        style={{

                            width: 300,

                            height: 300,

                            fontSize: 32,

                            border: '5px solid lightgray',

                        }}

                    >

                        <h3>{`От: ${device.price} руб.`}</h3>

                        {amount ? (

                            <>

                                <div>

                                    <Button

                                        variant='outline-dark'

                                        onClick={() => {

                                            dispatch(

                                                increaseDeviceInBasket(id)

                                            );

                                        }}

                                    >

                                        +

                                    </Button>{' '}

                                    {amount}{' '}

                                    <Button

                                        variant='outline-dark'

                                        onClick={() => {

                                            dispatch(

                                                decreaseDeviceInBasket(id)

                                            );

                                        }}

                                    >

                                        -

                                    </Button>

                                </div>

                            </>

                        ) : (

                            <Button

                                variant='outline-dark'

                                onClick={() => {

                                    dispatch(

                                        createDeviceInBasket({

                                            ...device,

                                            amount: 1,

                                        })

                                    );

                                }}

                            >

                                Добавить в корзину

                            </Button>

                        )}

                    </Card>

                </Col>

            </Row>

            <Row className='d-flex flex-column m-3'>

                <h1>Характеристики</h1>

                {device.info.map((info, index) => (

                    <Row

                        key={info.id}

                        style={{

                            background:

                                index % 2 === 0 ? 'lightgray' : 'transparent',

                            padding: 10,

                        }}

                    >

                        {info.title}: {info.description}

                    </Row>

                ))}

            </Row>

        </Container>

    );

};

export default DevicePage;

// deviceSlice.js

import { createAsyncThunk, createSlice } from '@reduxjs/toolkit';

import {

    createBrand,

    createType,

    fetchBrands,

    fetchTypes,

    fetchDevices,

    createDevice,

    deleteType,

    deleteBrand,

    createBasketDevice,

    increaseBasketDevice,

    getBasketDevices,

    decreaseBasketDevice,

} from './axios/deviceApi';

const initialState = {

    types: [],

    brands: [],

    devices: [],

    basket: [],

    selectedType: {},

    selectedBrand: {},

    currentPage: 1,

    totalCount: 0,

    limit: 8,

};

export const getDevicesInBasket = createAsyncThunk(

    'device/getDevicesInBasket',

    async () => {

        return await getBasketDevices();

    }

);

export const createDeviceInBasket = createAsyncThunk(

    'device/createDeviceInBasket',

    async (device) => {

        return await createBasketDevice(device);

    }

);

export const increaseDeviceInBasket = createAsyncThunk(

    'device/increaseDeviceInBasket',

    async (deviceId) => {

        return await increaseBasketDevice({ deviceId });

    }

);

export const decreaseDeviceInBasket = createAsyncThunk(

    'device/decreaseDeviceInBasket',

    async (deviceId) => {

        return await decreaseBasketDevice({ deviceId });

    }

);

export const addType = createAsyncThunk('device/addType', async (type) => {

    return await createType(type);

});

export const removeType = createAsyncThunk('device/removeType', async (id) => {

    return await deleteType(id);

});

export const getTypes = createAsyncThunk('device/getTypes', async () => {

    return await fetchTypes();

});

export const addBrand = createAsyncThunk('device/addBrand', async (brand) => {

    return await createBrand(brand);

});

export const removeBrand = createAsyncThunk(

    'device/removeBrand',

    async (id) => {

        return await deleteBrand(id);

    }

);

export const getBrands = createAsyncThunk('device/getBrands', async () => {

    return await fetchBrands();

});

export const getDevices = createAsyncThunk(

    'device/getDevices',

    async ({ typeId, brandId, page, limit }) => {

        return await fetchDevices(typeId, brandId, page, limit);

    }

);

export const addDevice = createAsyncThunk('device/addDevice', async (data) => {

    return await createDevice(data);

});

const deviceSlice = createSlice({

    name: 'device',

    initialState,

    reducers: {

        typeSelected: (state, action) => {

            state.selectedType = action.payload;

            state.currentPage = 1;

        },

        brandSelected: (state, action) => {

            state.selectedBrand = action.payload;

            state.currentPage = 1;

        },

        pageSelected: (state, action) => {

            state.currentPage = action.payload;

        },

    },

    extraReducers: (builder) => {

        builder

            .addCase(addType.fulfilled, (state, action) => {

                state.types.push(action.payload);

            })

            .addCase(getTypes.fulfilled, (state, action) => {

                state.types = action.payload;

            })

            .addCase(addBrand.fulfilled, (state, action) => {

                state.brands.push(action.payload);

            })

            .addCase(getBrands.fulfilled, (state, action) => {

                state.brands = action.payload;

            })

            .addCase(getDevices.fulfilled, (state, action) => {

                state.devices = action.payload.rows;

                state.totalCount = action.payload.count;

            })

            .addCase(addDevice.fulfilled, (state, action) => {

                state.devices.push(action.payload);

            })

            .addCase(removeType.fulfilled, (state, action) => {

                const index = state.types.indexOf(action.payload);

                state.types.splice(index, 1);

            })

            .addCase(removeBrand.fulfilled, (state, action) => {

                const index = state.brands.indexOf(action.payload);

                state.brands.splice(index, 1);

            })

            .addCase(createDeviceInBasket.fulfilled, (state, action) => {

                state.basket.push(action.payload);

            })

            .addCase(increaseDeviceInBasket.fulfilled, (state, action) => {

                state.basket = state.basket.map((item) => {

                    if (action.payload.deviceId == item.id) {

                        return { ...item, amount: action.payload.amount };

                    }

                    return item;

                });

            })

            .addCase(getDevicesInBasket.fulfilled, (state, action) => {

                state.basket = action.payload;

            })

            .addCase(decreaseDeviceInBasket.fulfilled, (state, action) => {

                state.basket = state.basket.map((item) => {

                    if (action.payload.deviceId == item.id) {

                        return { ...item, amount: action.payload.amount };

                    }

                    return item;

                });

            });

    },

});

const { reducer, actions } = deviceSlice;

export default reducer;

export const { typeSelected, brandSelected, pageSelected } = actions;

// deviceApi.js

import { $authHost, $host } from './index';

export const getBasketDevices = async () => {

    const { data } = await $authHost.get('api/basket');

    return data;

};

export const increaseBasketDevice = async (deviceId) => {

    const { data } = await $authHost.patch('api/basket/increase', deviceId);

    return data;

};

export const decreaseBasketDevice = async (deviceId) => {

    const { data } = await $authHost.patch('api/basket/decrease', deviceId);

    return data;

};

export const createBasketDevice = async (device) => {

    const { data } = await $authHost.post('api/basket', device);

    return data;

};

export const createType = async (type) => {

    const { data } = await $authHost.post('api/type', type);

    return data;

};

export const deleteType = async (id) => {

    const { data } = await $authHost.delete('api/type', { data: { id } });

    return data;

};

export const fetchTypes = async () => {

    const { data } = await $host.get('api/type');

    return data;

};

export const createBrand = async (brand) => {

    const { data } = await $authHost.post('api/brand', brand);

    return data;

};

export const deleteBrand = async (id) => {

    const { data } = await $authHost.delete('api/brand', { data: { id } });

    return data;

};

export const fetchBrands = async () => {

    const { data } = await $host.get('api/brand');

    return data;

};

export const createDevice = async (device) => {

    const { data } = await $authHost.post('api/device', device);

    return data;

};

export const fetchDevices = async (typeId, brandId, page, limit = 5) => {

    const { data } = await $host.get('api/device', {

        params: {

            typeId,

            brandId,

            page,

            limit,

        },

    });

    return data;

};

export const fetchOneDevice = async (id) => {

    const { data } = await $host.get('api/device/' + id);

    return data;

};

// basketController.js

const { BasketDevice, Device } = require('../models/models');

const ApiError = require('../error/ApiError');

class BasketController {

    async create(req, res) {

        const { basketId } = req.user;

        const deviceId = req.body.id;

        await BasketDevice.create({ basketId, deviceId, amount: 1 });

        return res.json(req.body);

    }

    async increase(req, res) {

        const { deviceId } = req.body;

        let { amount } = await BasketDevice.findOne({ where: { deviceId } });

        if (amount < 50) {

            amount++;

        }

        await BasketDevice.update(

            { amount },

            {

                where: {

                    deviceId,

                },

            }

        );

        return res.json({ deviceId, amount });

    }

    async getAll(req, res) {

        const basketDevices = await BasketDevice.findAll({

            include: Device,

        });

        return res.json(

            basketDevices.map((item) => {

                return { ...item.device.dataValues, amount: item.amount };

            })

        );

    }

    async decrease(req, res) {

        const { deviceId } = req.body;

        let { amount } = await BasketDevice.findOne({ where: { deviceId } });

        if (amount > 0) {

            amount--;

        }

        await BasketDevice.update(

            { amount },

            {

                where: {

                    deviceId,

                },

            }

        );

        return res.json({ deviceId, amount });

    }

}

module.exports = new BasketController();