МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт комплексной безопасности и специального приборостроения

Кафедра «Цифровые технологии обработки данных»

ОТЧЕТ
по практической работе
«Практическая работа №5. СУБД «Семейство столбцов» Apache Cassandra» по дисциплине «Нереляционные системы управления базами данных»

Выполнил Смирнов И.А.
фамилия, имя, отчество

21Б0700 шифр

БСБО-11-21 группа

Проверил к.т.н., доцент Ильин Д.Ю.
ученая степень, должность *фамилия, имя, отчество*

Москва 2023г.

Цель практической работы

Цель настоящей практической работы – научиться использовать СУБД «семейство столбцов» Apache Cassandra.

Задачи практической работы

Для достижения поставленной цели необходимо решить следующие задачи:

1. Спроектировать программное обеспечение и отразить в результирующих схемах применение Apache Cassandra.

2. В рамках проектирования программного обеспечения записать запросы пользователей и их последовательность при применении системы.

3. Спроектировать ER-диаграмму базы данных в нотации Чена.

4. Спроектировать логическую и физическую схему базы данных в нотации Чеботко.

5. Установить Apache Cassandra и осуществить к ней ручной доступ через любое доступное программное обеспечение.

6. Разработать программное обеспечение, использующее Apache Cassandra для данных предметной области, определенной вариантом задания.

7. Протестировать программное обеспечение и продемонстрировать корректность его работы.

8. Подготовить ответы на контрольные вопросы.

9. Составить отчет о проведенной работе.

Вариант: 2
нтернет-магазин

Скриншоты, показывающие ход выполнения практической работы

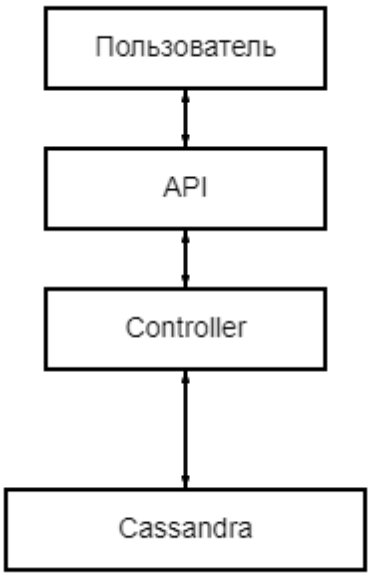

Рис.1 – Результирующая схема

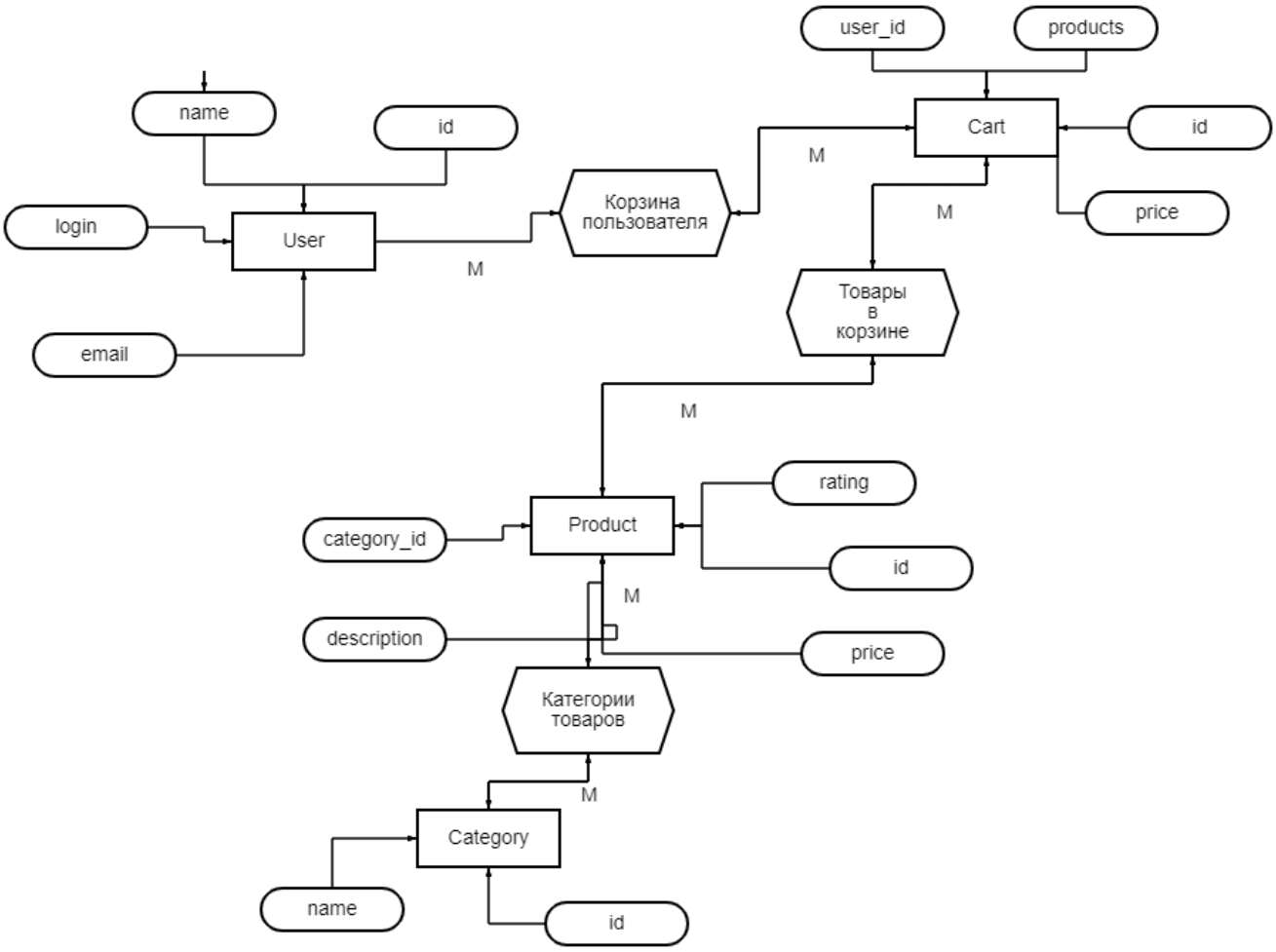

Рисунок 2 – ER-диаграмма базы данных в нотации Чена

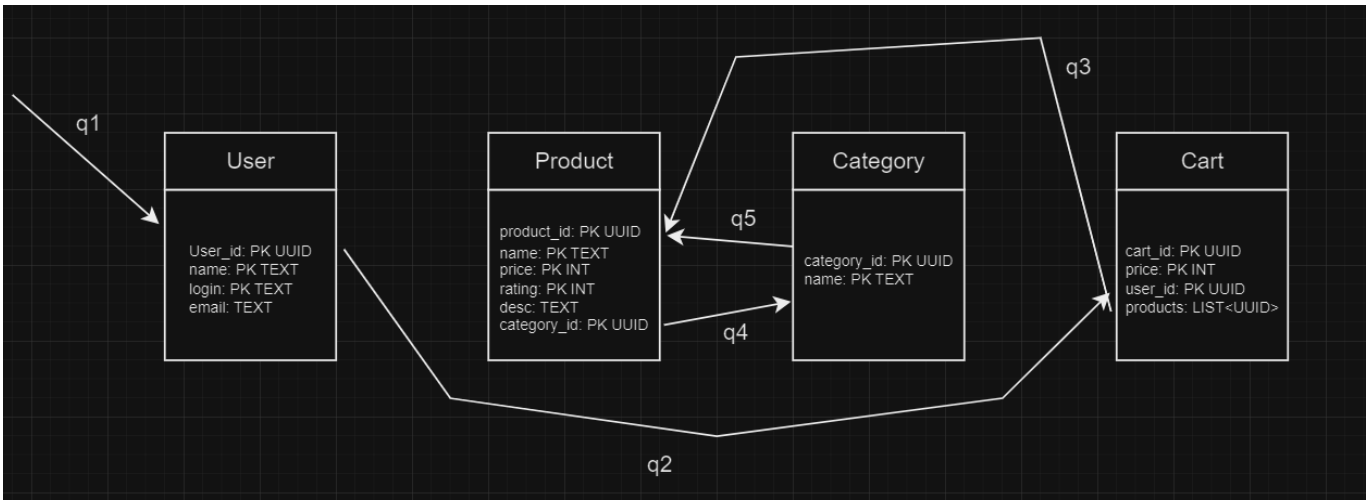

Рисунок 3 – Логическая схема базы данных в нотации Чеботко.

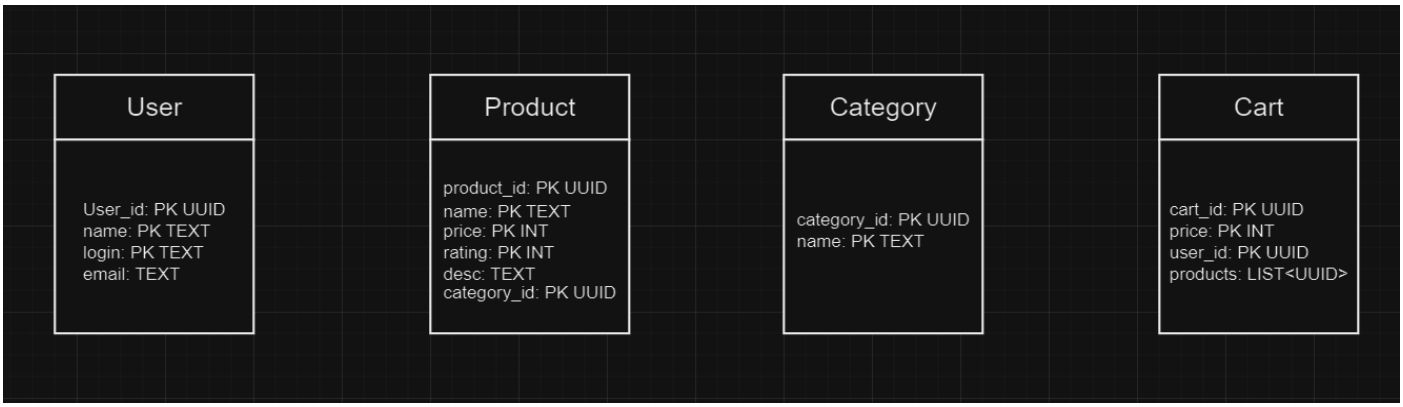

Рисунок 4 – Физическая схема базы данных в нотации Чеботко.

Для начала работы была создана сама БД, а также индексы для будущих запросов (листинг 1):

```
const createTableUser = `
    CREATE TABLE IF NOT EXISTS eshop.user (
    user_id UUID PRIMARY KEY,
    name TEXT,
    login TEXT,
    email TEXT,
);`;


const createTableProduct = `
    CREATE TABLE IF NOT EXISTS eshop.product (
    product_id UUID PRIMARY KEY,
    name TEXT,
    price INT,
    rating INT,
    description TEXT,
    category_id UUID
);`;

const createTableCategory = `
    CREATE TABLE IF NOT EXISTS eshop.category (
    category_id UUID PRIMARY KEY,
    name TEXT
);`;

const createTableCart = `
    CREATE TABLE IF NOT EXISTS eshop.cart (
    cart_id UUID PRIMARY KEY,
    price INT,
    user_id UUID,
    products SET<UUID>
);`;

await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.user (login)', [],
{prepare: true});

await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.product (name)', [],
{prepare: true});
await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.product (price)', [],
{prepare: true});
await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.product (rating)', [],
{prepare: true});
await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.product (category_id)',
[], {prepare: true});

await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.category (name)', [],
{prepare: true});

await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.cart (user_id)', [],
{prepare: true});
```

Листинг 1 – создание таблиц и индексов.

Далее были написаны CRUD операции для товаров.

```
app.post('/products', async (req, res) => {
    const {name, price, rating, description, category} = req.body;

    if (!name || !price || !rating || !description || !category) {
        return res.status(400).json({error: 'Invalid request. Please provide all
product details.'});
    }

    try {
        const productId = uuid.v4();

        const categoryId = await getCategoryByName(category);

        if (!categoryId) {
            return res.status(400).json({error: 'Нет такой категории'});
        }

        const query = 'INSERT INTO eshop.product (product_id, name, price, rating,
description, category_id) VALUES (?, ?, ?, ?, ?, ?)';
        await client.execute(query, [productId, name, price, rating, description,
categoryId], {prepare: true});

        res.status(201).json({message: 'Product created successfully'});
    } catch (error) {
        console.error('Error creating product:', error);
        res.status(500).json({error: 'Internal Server Error'});
    }
});

app.get('/products', async (req, res) => {
    try {
        const query = 'SELECT * FROM eshop.product';
        const result = await client.execute(query, []);

        const productsWithCategories = await Promise.all(
            result.rows.map(async (product) => {
                const categoryName = await getCategoryById(product.category_id);
                return {...product, category: categoryName};
            })
        );

        res.json(productsWithCategories);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.put('/products/:productId', async (req, res) => {
    const productId = req.params.productId;
    const {name, price, description, rating} = req.body;

    try {
        const query = 'UPDATE eshop.product SET name = ?, price = ?, description =
?, rating = ? WHERE product_id = ?';
        await client.execute(query, [name, price, description, rating, productId],
{prepare: true});
        res.json({message: 'Продукт обновлён!'});
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.delete('/products/:productId', async (req, res) => {
    const productId = req.params.productId;
```

```
    try {
        const query = 'DELETE FROM eshop.product WHERE product_id = ?';
        await client.execute(query, [productId], {prepare: true});
        res.json({message: 'Продукт удалён!'});
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});
```

Результат работы операций.

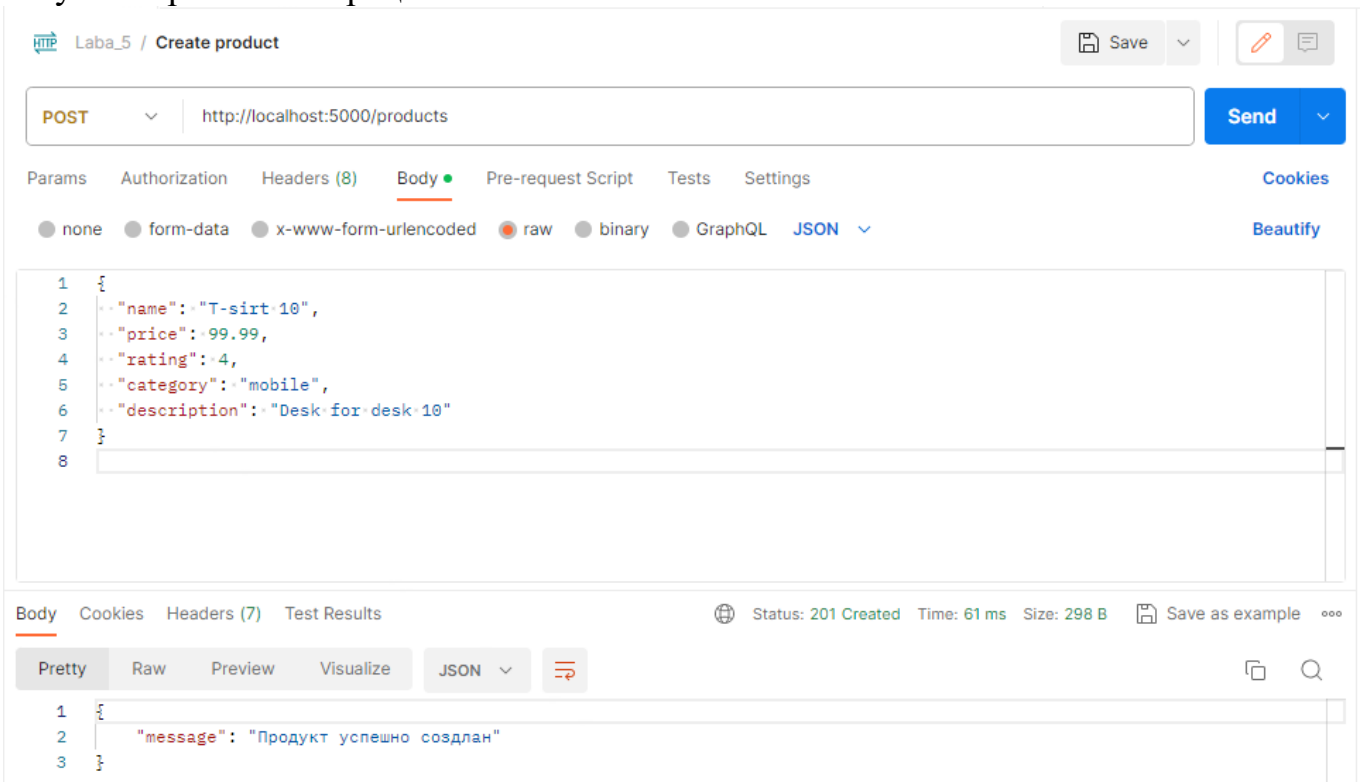

Рис. 5 - создание товара.

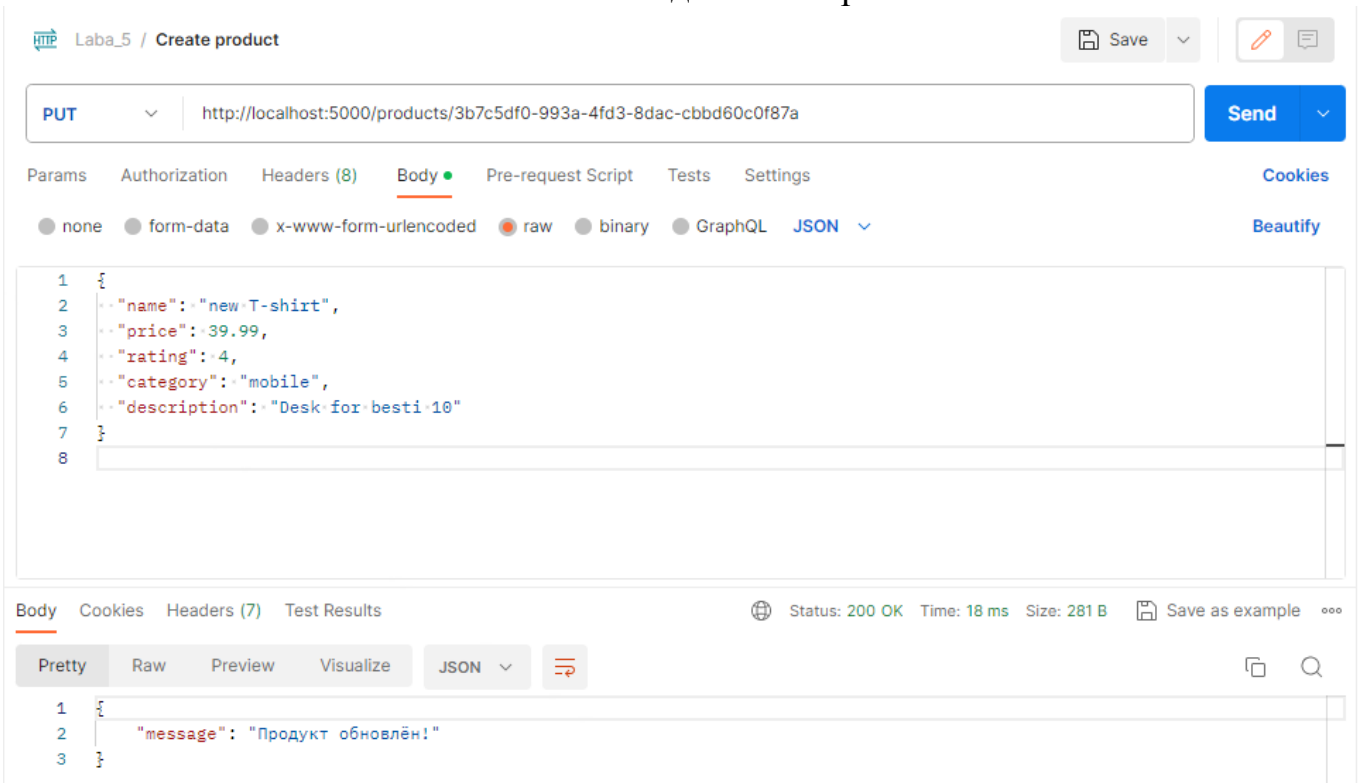

Рис. 6 – обновление товара.

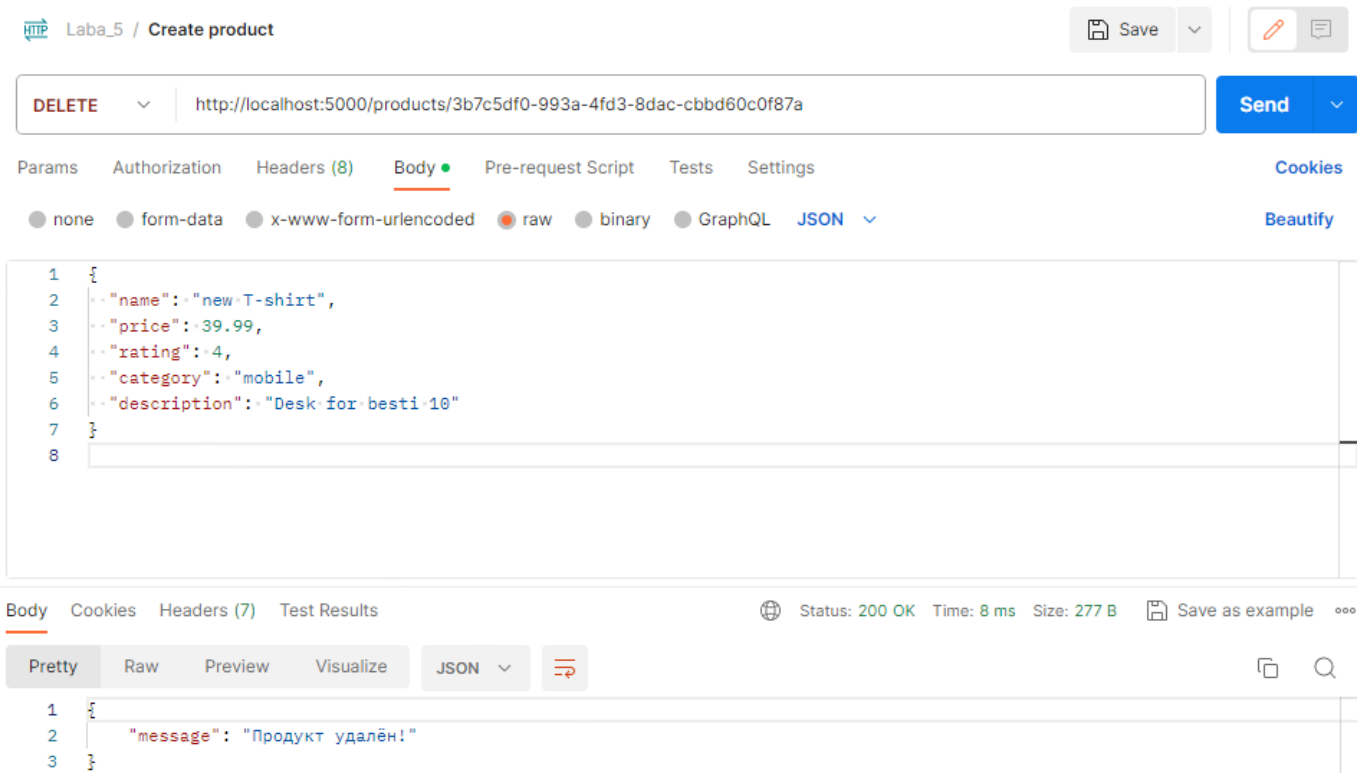

Рис. 7 – удаление товара.

Сформировал 5 пользовательских запросов.

- Пользователь заходит под своим логином.
- Пользователь переходит в свою корзину
- Пользователь выбирает продукт
- Пользователь переходит по категории товара, чтобы посмотреть остальные продукты данной категории
- Пользователь выбирает продукт на основе рейтинга

```
app.get('/users/by-login', async (req, res) => {
    const login = req.query.login;

    try {
        const query = 'SELECT * FROM eshop.user WHERE login = ?';
        const result = await client.execute(query, [login], {prepare: true});

        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});
```

```
app.get('/cart/:user_id', async (req, res) => {
    try {
        const user_id = req.params.user_id;

        const query = 'SELECT * FROM eshop.cart WHERE user_id = ?';
        const result = await client.execute(query, [user_id], {prepare: true});

        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
```

```
    }
});
```

```
app.get('/products/:productId', async (req, res) => {
    const productId = req.params.productId;
    try {
        const query = 'SELECT * FROM eshop.product WHERE product_id = ?';
        const result = await client.execute(query, [productId]);

        if (result.rows.length > 0) {
            const product = result.rows[0];
            const categoryName = await getCategoryById(product.category_id);
            console.log(categoryName);

            res.json({product, categoryName});
        } else {
            res.status(404).json({error: 'Продукт не найден'});
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.get('/by-category/:category_id', async (req, res) => {
    const category_id = req.params.category_id;
    try {
        const query = 'SELECT * FROM eshop.product WHERE category_id = ?';
        const result = await client.execute(query, [category_id], {prepare: true});

        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});
```

```
app.get('/by-category-rating/:category_id', async (req, res) => {
    const category_id = req.params.category_id;
    const rating = req.query.rating;

    try {
        const query = 'SELECT * FROM eshop.product WHERE category_id = ? AND rating
= ? ALLOW FILTERING';
        const result = await client.execute(query, [category_id, rating], {prepare:
true});

        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});
```

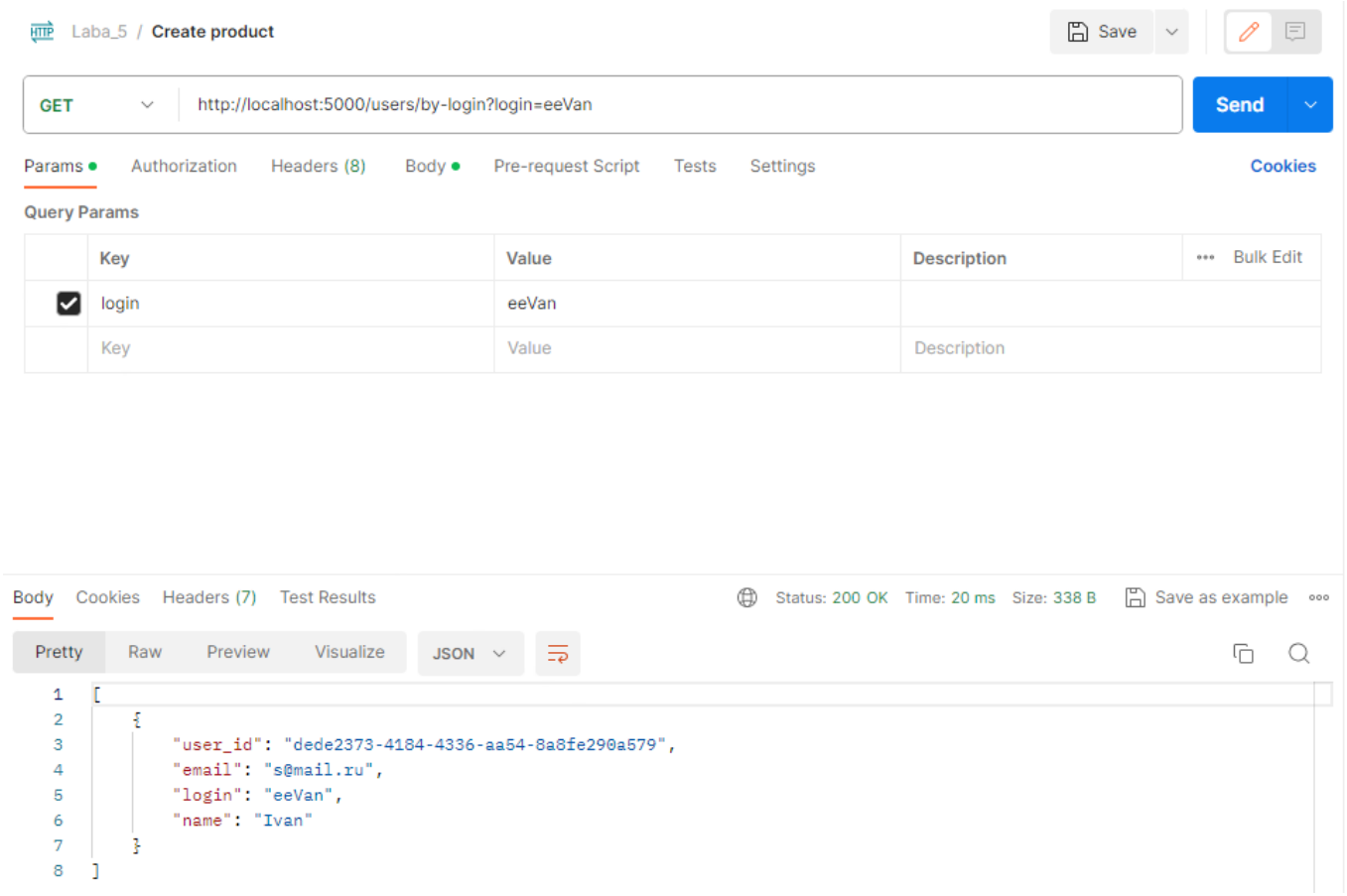

Рис. 8 – вход по логину.

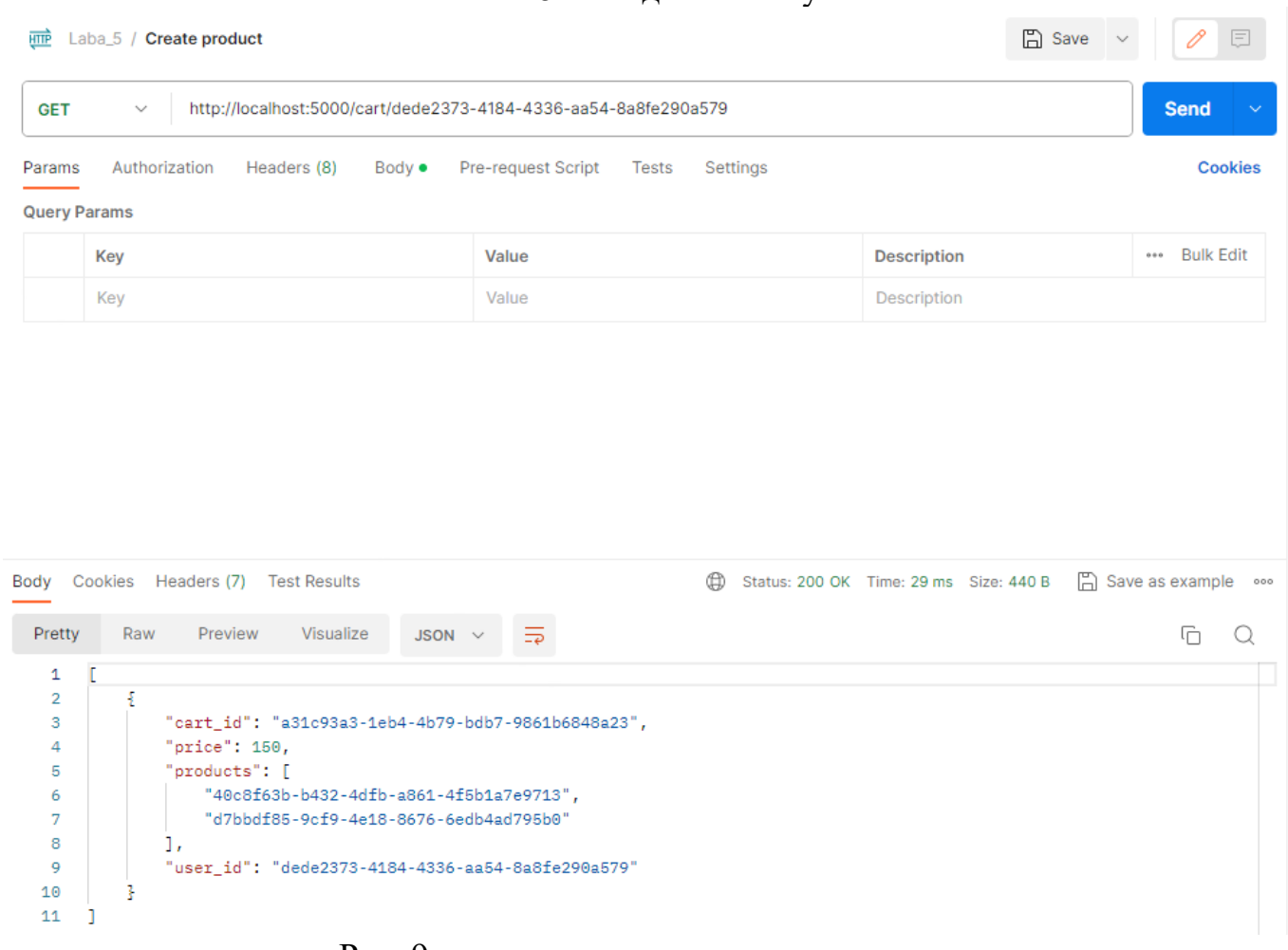

Рис. 9 - переход в корзину пользователя.

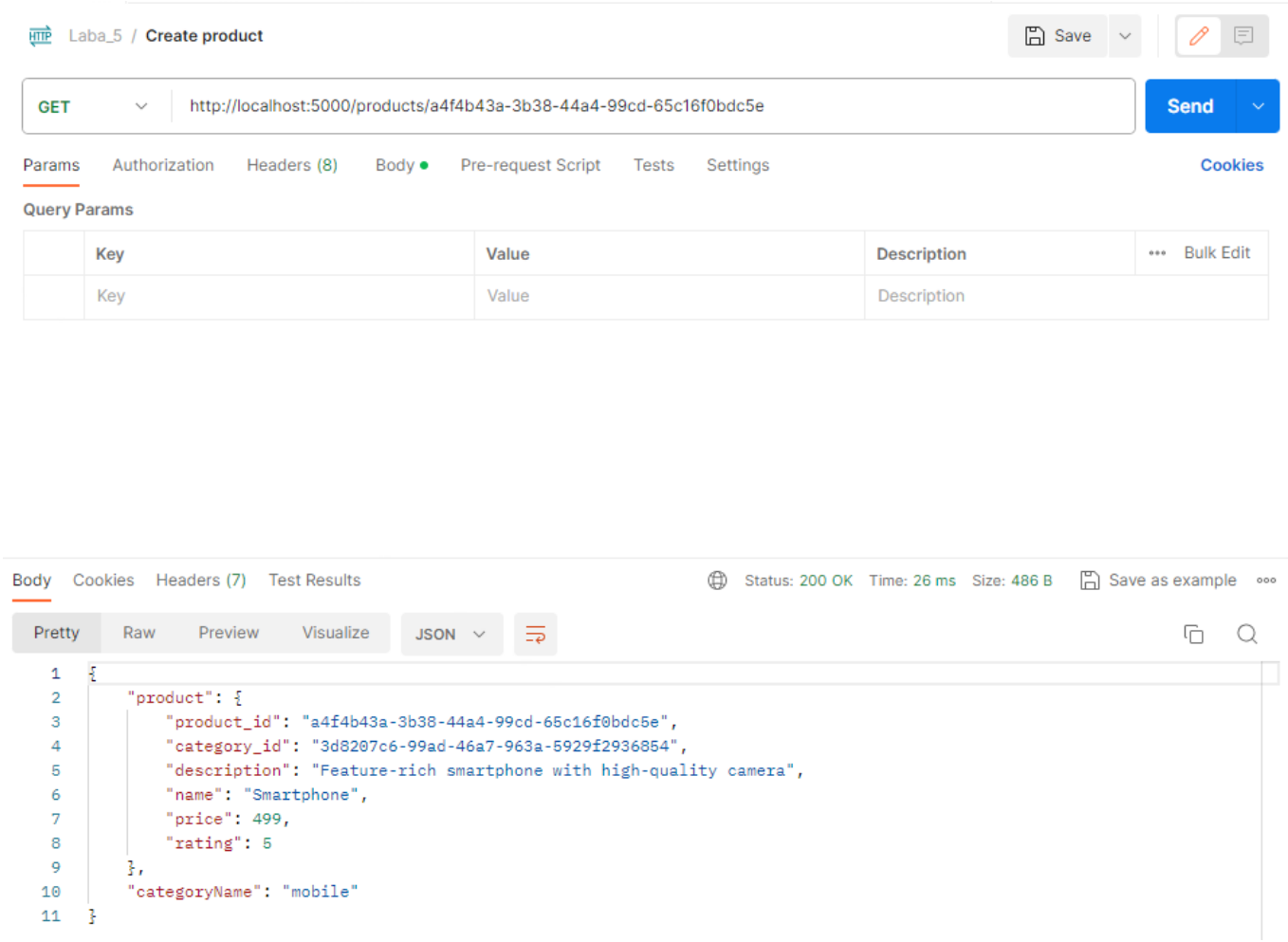

Рис. 10 – переход на страницу продукта.

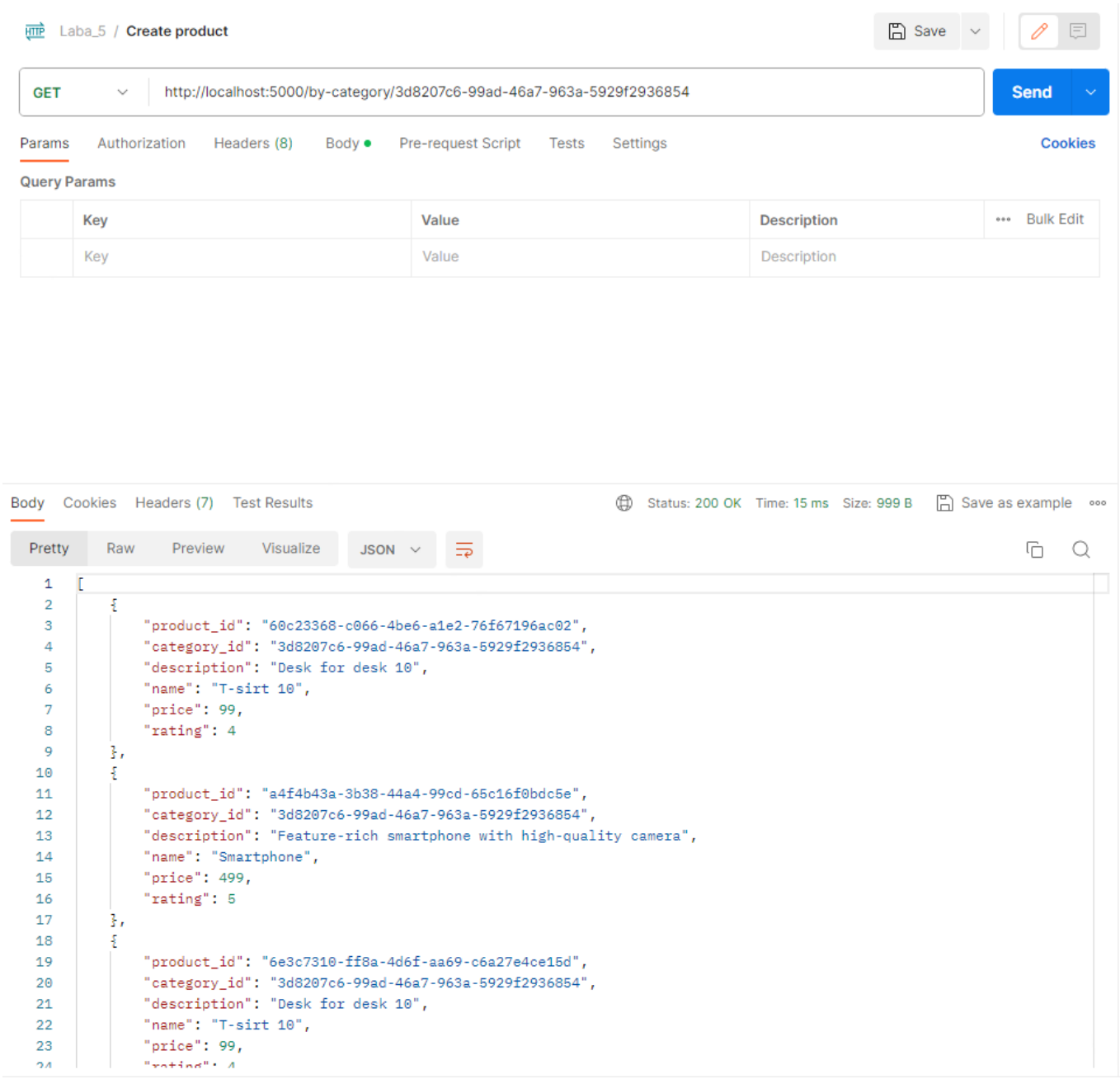

Рис. 11 – товары определенной категории товаров.

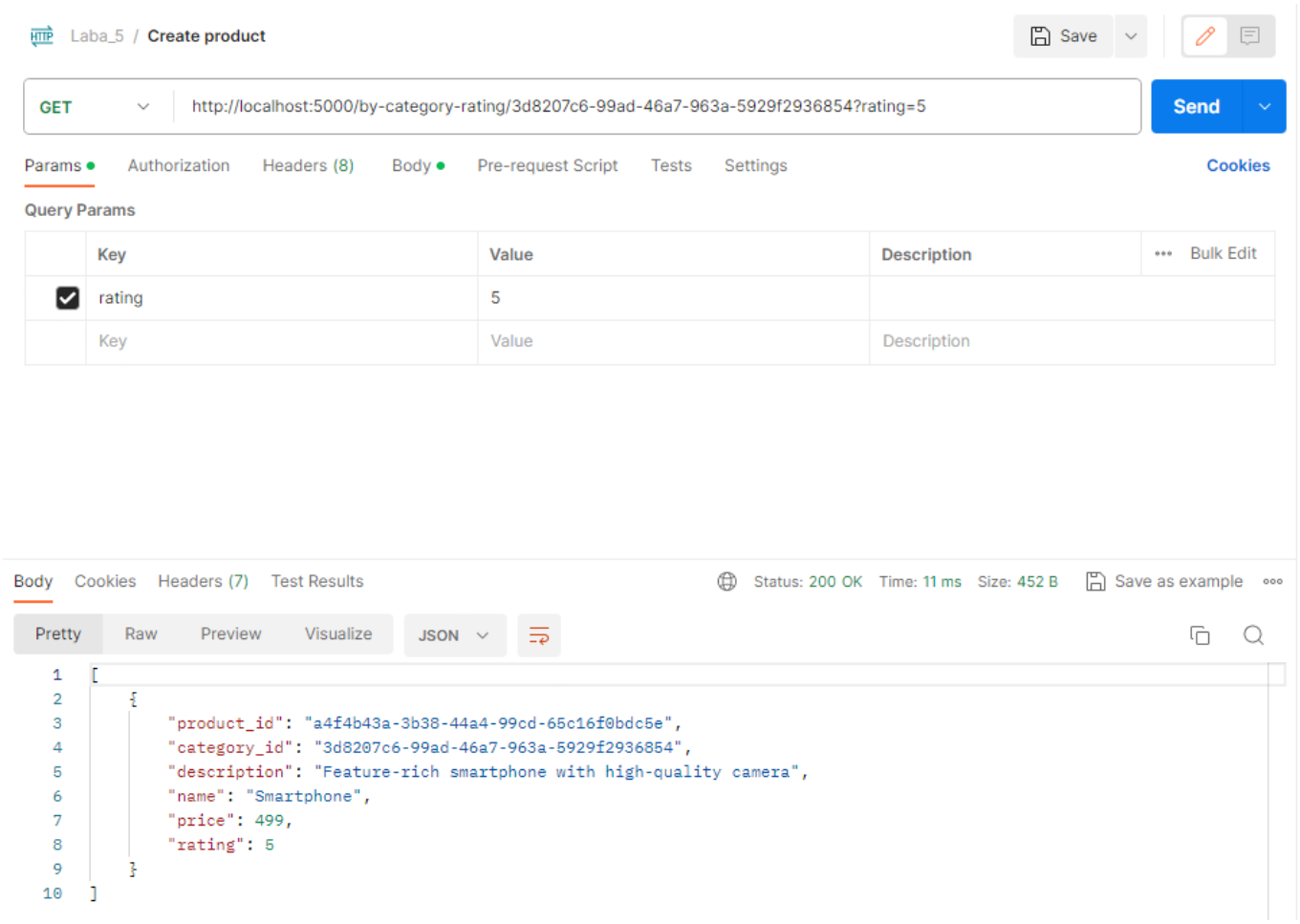

Рис. 12 – товары определённо категории и рейтинга.

Листинг приложения

```
const express = require('express');
const {Client} = require('cassandra-driver');
const bodyParser = require('body-parser');
const uuid = require('uuid');
// const productRoute = require('./routes/product');

const app = express();
const PORT = 5000;
const LINK = `http://localhost:${PORT}`;

const client = new Client({
    contactPoints: ['localhost'],
    localDataCenter: 'datacenter1',
    keyspace: 'eshop'
});


(async () => {
    await client.connect();
    console.log("Подлючение к Cossandra выполнено");
})();

const createTableUser = `
    CREATE TABLE IF NOT EXISTS eshop.user (
    user_id UUID PRIMARY KEY,
    name TEXT,
    login TEXT,
    email TEXT,
);`;


const createTableProduct = `
    CREATE TABLE IF NOT EXISTS eshop.product (
    product_id UUID PRIMARY KEY,
    name TEXT,
    price INT,
    rating INT,
    description TEXT,
    category_id UUID
);`;

const createTableCategory = `
    CREATE TABLE IF NOT EXISTS eshop.category (
    category_id UUID PRIMARY KEY,
    name TEXT
);`;

const createTableCart = `
    CREATE TABLE IF NOT EXISTS eshop.cart (
    cart_id UUID PRIMARY KEY,
    price INT,
    user_id UUID,
    products SET<UUID>
);`;


(async () => {

    await client.execute(createTableUser, [], {prepare: true});

    await client.execute(createTableCart, [], {prepare: true});

    // await client.execute(createTableUserProduct, [], {prepare: true});

    await client.execute(createTableCategory, [], {prepare: true});
```

```
    await client.execute(createTableProduct, [], {prepare: true});

    await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.user (login)', [],
{prepare: true});

    await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.product (name)', [],
{prepare: true});
    await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.product (price)', [],
{prepare: true});
    await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.product (rating)', [],
{prepare: true});
    await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.product
(category_id)', [], {prepare: true});

    await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.category (name)', [],
{prepare: true});

    await client.execute('CREATE INDEX IF NOT EXISTS ON eshop.cart (user_id)', [],
{prepare: true});


})();

async function getCategoryByName(categoryName) {
    try {
        const query = 'SELECT category_id FROM eshop.category WHERE name = ?';
        const result = await client.execute(query, [categoryName], {prepare: true});

        if (result.rows.length > 0) {
            return result.rows[0].category_id;
        } else {
            return null;
        }
    } catch (error) {
        console.error('Ошибка доступа к категории:', error);
    }
}

async function getCategoryById(categoryId) {
    try {
        if (categoryId === undefined || categoryId === null) {
            return null;
        }

        const query = 'SELECT name FROM eshop.category WHERE category_id = ?';
        const result = await client.execute(query, [categoryId], {prepare: true});

        if (result.rows.length > 0) {
            return result.rows[0].name;
        } else {
            return null;
        }
    } catch (error) {
        console.error('Ошибка доступа к категории:', error);
        return null;
    }
}


app.use(bodyParser.json());

app.post('/users', async (req, res) => {
    const {name, login, email} = req.body;

    if (!name || !login || !email) {
        return res.status(400).json({error: 'Введены не все данные'});
    }
```

```
    try {
        const userId = uuid.v4();

        const query = 'INSERT INTO eshop.user (user_id, name, login, email) VALUES
(?, ?, ?, ?)';
        await client.execute(query, [userId, name, login, email], {prepare: true});

        res.status(201).json({message: 'Пользователь добавлен!'});
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.get('/users', async (req, res) => {
    try {
        const query = 'SELECT * FROM eshop.user';
        const result = await client.execute(query, [], {prepare: true});

        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

// 1
app.get('/users/by-login', async (req, res) => {
    const login = req.query.login;

    try {
        const query = 'SELECT * FROM eshop.user WHERE login = ?';
        const result = await client.execute(query, [login], {prepare: true});

        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.get('/users/:userId', async (req, res) => {
    const userId = req.params.userId;

    try {
        const query = 'SELECT * FROM eshop.user WHERE user_id = ?';
        const result = await client.execute(query, [userId], {prepare: true});

        if (result.rows.length > 0) {
            res.json(result.rows[0]);
        } else {
            res.status(404).json({error: 'Пользователь не найден'});
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.post('/categories', async (req, res) => {
    const {name} = req.body;

    if (!name) {
        return res.status(400).json({error: 'Заполните все данные'});
    }

    try {
```

```
        const categoryId = uuid.v4();
        const query = 'INSERT INTO eshop.category (category_id, name) VALUES (?,
?)';
        await client.execute(query, [categoryId, name], {prepare: true});

        res.status(201).json({message: 'категория создана'});
    } catch (error) {
        console.error('Ошибка создания категории', error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.get('/categories', async (req, res) => {

    try {
        const getCategoryQuery = 'SELECT * FROM eshop.category';
        const categoryResult = await client.execute(getCategoryQuery, [], {prepare:
true});

        res.json(categoryResult.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.post('/products', async (req, res) => {
    const {name, price, rating, description, category} = req.body;

    if (!name || !price || !rating || !description || !category) {
        return res.status(400).json({error: 'Введите все данные'});
    }

    try {
        const productId = uuid.v4();

        const categoryId = await getCategoryByName(category);

        if (!categoryId) {
            return res.status(400).json({error: 'Нет такой категории'});
        }

        const query = 'INSERT INTO eshop.product (product_id, name, price, rating,
description, category_id) VALUES (?, ?, ?, ?, ?, ?)';
        await client.execute(query, [productId, name, price, rating, description,
categoryId], {prepare: true});

        res.status(201).json({message: 'Продукт успешно создлан'});
    } catch (error) {
        console.error('Ошибка создания продукта:', error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.get('/products', async (req, res) => {
    try {
        const query = 'SELECT * FROM eshop.product';
        const result = await client.execute(query, []);

        const productsWithCategories = await Promise.all(
            result.rows.map(async (product) => {
                const categoryName = await getCategoryById(product.category_id);
                return {...product, category: categoryName};
            })
        );

        res.json(productsWithCategories);
    } catch (error) {
```

```
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.put('/products/:productId', async (req, res) => {
    const productId = req.params.productId;
    const {name, price, description, rating} = req.body;

    try {
        const query = 'UPDATE eshop.product SET name = ?, price = ?, description =
?, rating = ? WHERE product_id = ?';
        await client.execute(query, [name, price, description, rating, productId],
{prepare: true});
        res.json({message: 'Продукт обновлён!'});
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.delete('/products/:productId', async (req, res) => {
    const productId = req.params.productId;
    try {
        const query = 'DELETE FROM eshop.product WHERE product_id = ?';
        await client.execute(query, [productId], {prepare: true});
        res.json({message: 'Продукт удалён!'});
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

// 3
app.get('/products/:productId', async (req, res) => {
    const productId = req.params.productId;
    try {
        const query = 'SELECT * FROM eshop.product WHERE product_id = ?';
        const result = await client.execute(query, [productId]);

        if (result.rows.length > 0) {
            const product = result.rows[0];
            const categoryName = await getCategoryById(product.category_id);
            console.log(categoryName);

            res.json({product, categoryName});
        } else {
            res.status(404).json({error: 'Продукт не найден'});
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

// 4
app.get('/by-category/:category_id', async (req, res) => {
    const category_id = req.params.category_id;
    try {
        const query = 'SELECT * FROM eshop.product WHERE category_id = ?';
        const result = await client.execute(query, [category_id], {prepare: true});

        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});
```

```
// 5
app.get('/by-category-rating/:category_id', async (req, res) => {
    const category_id = req.params.category_id;
    const rating = req.query.rating;

    try {
        const query = 'SELECT * FROM eshop.product WHERE category_id = ? AND rating
= ?';
        const result = await client.execute(query, [category_id, rating], {prepare:
true});

        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});


app.post('/cart', async (req, res) => {
    try {
        const {price, user_id, products} = req.body;

        if (!price || !user_id || !products) {
            return res.status(400).json({error: 'Введите все данные'});
        }

        const cartId = uuid.v4();

        const query = 'INSERT INTO eshop.cart (cart_id, price, user_id, products)
VALUES (?, ?, ?, ?)';
        await client.execute(query, [cartId, price, user_id, products], {prepare:
true});

        res.status(201).json({message: 'Корзина добавлена'});
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

// 2
app.get('/cart/:user_id', async (req, res) => {
    try {
        const user_id = req.params.user_id;

        const query = 'SELECT * FROM eshop.cart WHERE user_id = ?';
        const result = await client.execute(query, [user_id], {prepare: true});

        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.get('/by-price', async (req, res) => {
    const minPrice = parseFloat(req.query.min) || 0;
    const maxPrice = parseFloat(req.query.max) || Number.MAX_SAFE_INTEGER;

    try {
        const query = 'SELECT * FROM eshop.product WHERE price >= ? AND price <= ?
ALLOW FILTERING';
        const result = await client.execute(query, [minPrice, maxPrice], {prepare:
true});
        res.json(result.rows);
```

```
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.get('/by-rating', async (req, res) => {
    const minRating = parseInt(req.query.min) || 0;
    const maxRating = parseInt(req.query.max) || 5;

    try {
        const query = 'SELECT * FROM eshop.product WHERE rating >= ? AND rating <= ?
ALLOW FILTERING';
        const result = await client.execute(query, [minRating, maxRating], {prepare:
true});
        res.json(result.rows);
    } catch (error) {
        console.error(error);
        res.status(500).json({error: 'Ошибка сервера'});
    }
});

app.listen(PORT, (err) => {
    err ? console.log(err) : console.log(`Приложение запущено на порту: ${PORT}.
Перейдите на : ${LINK}`);
});
```

Ответы на контрольные вопросы.

1. Области применения Apache Cassandra в информационных системах:

    a. Большие данные (Big Data): Позволяет эффективно хранить и обрабатывать большие объемы данных.

    b. Интернет-магазины и электронная коммерция: Обеспечивает высокую доступность и масштабируемость для обработки транзакций.

    c. Интернет-платформы и социальные сети: Где важными являются высокая производительность и распределенная структура данных.

    d. Системы управления временными рядами: Используется для хранения и обработки данных временных рядов.

2. Ограничения Apache Cassandra в отношении согласованности данных:

    a. Eventual Consistency (Консистентность в конечном итоге): Apache Cassandra обеспечивает модель консистентности в конечном итоге, что означает, что в какой-то момент система приходит к консистентному состоянию после выполнения операций, но в промежуточный момент данные могут быть несогласованными.

    b. Несколько уровней консистентности: Cassandra позволяет выбирать между различными уровнями консистентности, но это связано с компромиссами в производительности.

3. Библиотеки для программного взаимодействия с Apache Cassandra:

    a. DataStax Java Driver: Официальный драйвер для взаимодействия с Cassandra из языка Java.

    b. Cassandra Python Driver: Драйвер для языка Python, предоставляющий интерфейс для работы с Cassandra

    c. .Node.js Driver for Apache Cassandra: Драйвер для платформы Node.js.

d. C# DataStax Driver: Драйвер для взаимодействия с Cassandra из языка C#.

4. Настройки пространства ключей в Apache Cassandra:

    a. Strategy Class (Класс стратегии): Определяет, как данные будут распределены по узлам кластера. Например, SimpleStrategy или NetworkTopologyStrategy

    b. Replication Factor (Фактор репликации): Количество узлов, на которых будет реплицироваться каждый ключ. Определяет надежность данных
    .
    c. Компоненты ключа (Key Components): Cassandra поддерживает составные ключи, состоящие из нескольких компонентов.

5. Отличия СУБД "семейство столбцов" от других видов СУБД:

    a. Модель данных: В "семействе столбцов" данные хранятся не в виде строк, а в виде колонок, что обеспечивает эффективность при работе с широкими наборами данных.
    b. Гибкая схема: В отличие от реляционных баз данных, "семейство столбцов" позволяет добавлять новые колонки без изменения существующей схемы.

    c. Горизонтальное масштабирование: Базы данных этого типа обеспечивают легкое горизонтальное масштабирование, что делает их подходящими для обработки больших объемов данных и высоких нагрузок.

Выводы о результатах выполнения работы.

В ходе выполнения лабораторной работы были получены навыки работы с СУБД «семейство столбцов» Apache Cassandra.