МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА



ОТЧЁТ

по курсовой работе

по дисциплине

«Сервис Ориентированные Системы»

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жевнерчук Д.В.

Выполнил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Докукин Д.В.

Группа:

20-ПО

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Содержание**

[Содержание 2](#_Toc1)

[Список сокращений 2](#_Toc2)

[Введение 2](#_Toc3)

[Постановка задачи 3](#_Toc4)

[Анализ предметной области и проектирование API 3](#_Toc5)

[6.1 Объектно-ориентированная модель ресурсов 3](#_Toc6)

[Разработанная модель данных 3](#_Toc7)

[Описание таблиц и связей БД 4](#_Toc8)

[Диаграмма разработанной БД 5](#_Toc9)

[6.2 Проектирование REST-API 5](#_Toc10)

[Описание разработанных точек доступа к сервису 6](#_Toc11)

[Реализация сервис-ориентированной системы 7](#_Toc12)

[7.1 Аналитический обзор платформ и средств построения сервис-ориентированных систем 7](#_Toc13)

[7.2 Компонентная модель сервис-ориентированной системы 8](#_Toc14)

[7.3 Настройка программного проекта 10](#_Toc15)

[7.4 Описание реализации сервисов 11](#_Toc16)

[7.5 Описание тестового примера 16](#_Toc17)

[Заключение 22](#_Toc18)

[Список литературы 22](#_Toc19)

[Приложения 23](#_Toc20)

[Приложение 1 23](#_Toc21)

[Приложение 2 23](#_Toc22)

[Приложение 3 24](#_Toc23)

[Приложение 4 29](#_Toc24)

[Приложение 5 32](#_Toc25)

**Список сокращений**

СУБД – система управления базами данных

СОА – сервис-ориентированная архитектура

БД – база данных

CRUD – create read update delete – акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание, чтение, модификация, удаление

ПО – программное обеспечение

**Введение**

На текущий момент, сервис-ориентированная архитектура широко применяется во многих современных проектах. Среди преимуществ, можно выделить возможность применить модульный подход при реализации программного обеспечения, что позволяет вести разработку различных частей проекта параллельно и независимо друг от друга. Обеспечивая связь между сервисами посредством стандартизированных интерфейсов, каждый сервис может развиваться независимо от других, с применением любых программных и аппаратных средств, наиболее оптимальных для решения поставленной задачи. Вытекающая из этого слабая связанность даёт возможность легко обновлять и заменять сервисы. Такая модульность также даёт возможность реализовать принцип избыточности, при котором сервисы могут дублироваться, что особенно полезно для популярных ресурсов и высоко-нагруженных систем.

С середины 2010-х годов, начала обретать развитие микросервисная архитектура – это вариант СОА, принцип которой заключается в применении минимальных по размеру сервисов, отвечающих строго определённым и узким перечням задач. Таким образом, можно заключить, что сервис-ориентированная архитектура актуальна и активно развивается.

Рассматривая поставленную задачу – создание сервис-ориентированной системы аналитической обработки результатов продаж в Интернет-магазине – можно обрисовать несколько признаков:

* Наличие баз данных пользователей, товаров, категорий, заказов
* Наличие клиентской части, отвечающей за отображение визуального интерфейса, понятного пользователям
* Наличие сервера, обрабатывающего запросы пользователей, отвечающего за управление данными, контроль доступа к ресурсам системы

Исходя из этого, можно разбить задачу на условные три части:

1. Фронтенд (клиентская часть)
2. Бекенд (серверная часть)
3. СУБД (база данных проекта)

Описанные части можно представить, как отдельные сервисы, общающиеся друг с другом посредством стандартизированных интерфейсов.

Аналитическая обработка результатов продаж предполагает наличие как минимум базы данных интернет-магазина. Чтобы изучить детали работы такого ресурса, для лучшего понимания, я ставлю себе задачей реализацию всех описанных частей интернет-магазина. По завершении, обработку данных по продажам можно будет интегрировать в клиентскую часть, настроив ограничение доступа к соответствующему разделу.

Целью курсовой работы можно определить задачу по созданию прототипа такой системы.

Среди сопутствующих задач в ходе выполнения работы, можно выделить разработку модели данных, выбор средств разработки, определение целевой платформы и документирование.

**Постановка задачи**

Вариант курсовой работы – 5:

Сервис-ориентированная система аналитической обработки результатов продаж в Интернет-магазине.

Анализ предметной области и проектирование API

6.1 Объектно-ориентированная модель ресурсов

Разработанная модель данных

В модели данных были спланированы следующие сущности:

* Пользователь (**User**) – соответствует таблице пользователей. Хранит целочисленный авто-инкрементируемый идентификатор (id), строковый электронный адрес (email), строковый хэшированный пароль (password) и строковую роль (role) – USER (по умолчанию) или ADMIN.
* Тип (**Type**) – соответствует таблице типов устройств. Хранит целочисленный авто-инкрементируемый идентификатор (id) и строковое имя (name).
* Бренд (**Brand**) – соответствует таблице брендов устройств. Хранит целочисленный авто-инкрементируемый идентификатор (id) и строковое имя (name).
* Устройство (**Device**) – соответствует таблице устройств. Хранит целочисленный авто-инкрементируемый идентификатор (id), строковое имя (name), целочисленную стоимость (price), строковое имя хранящегося на сервере изображения (img), целочисленный идентификатор типа (typeId), целочисленный идентификатор бренда (brandId).
* Таблица Тип-бренд (**TypeBrand**) – для обозначения связи «многие ко многим» между Типами и Брендами. Хранит целочисленный авто-инкрементируемый идентификатор (id), целочисленный идентификатор типа (typeId), целочисленный идентификатор бренда (brandId).
* Заказ (**Order**) – соответствует таблице заказов. Хранит целочисленный авто-инкрементируемый идентификатор (id), целочисленную общую стоимость заказа (price), целочисленный идентификатор пользователя (userId).
* Товар заказа (**OrderItem**) – соответствует таблице товаров из заказов пользователей. Хранит целочисленный авто-инкрементируемый идентификатор (id), целочисленное количество товаров (amount), целочисленный идентификатор заказа (orderId), целочисленный идентификатор устройства (deviceId).

У каждой из сущностей автоматически созданы два дополнительных поля – время создания записи (createdAt) и время обновления (updatedAt).

Описание таблиц и связей БД

Таблица «users», производная от модели User, хранит записи пользователей.

Таблица «devices», производная от модели Device, содержит список устройств.

Таблица «types», производная от модели Type, содержит список типов устройств.

Таблица «brands», производная от модели Brand, содержит список брендов устройств.

Таблица «type\_brands», производная от модели TypeBrand, создаётся для реализации связи «многих ко многим» между таблицами «types» и «brands». Никак не задействуется.

Таблица «orders», производная от модели Order, хранит заказы пользователей.

Таблица «order\_items», производная от модели OrderItem, хранит записи устройств и их количество из заказов пользователей.

Между таблицами «users» и «orders» связь «Один ко многим», поскольку у одного пользователя может быть много заказов, но у каждого заказа может быть только один пользователь.

Между таблицами «orders» и «order\_items» связь «Один ко многим», так как на одном заказе может быть несколько добавленных устройств, но каждая такая запись может относиться только к одному заказу.

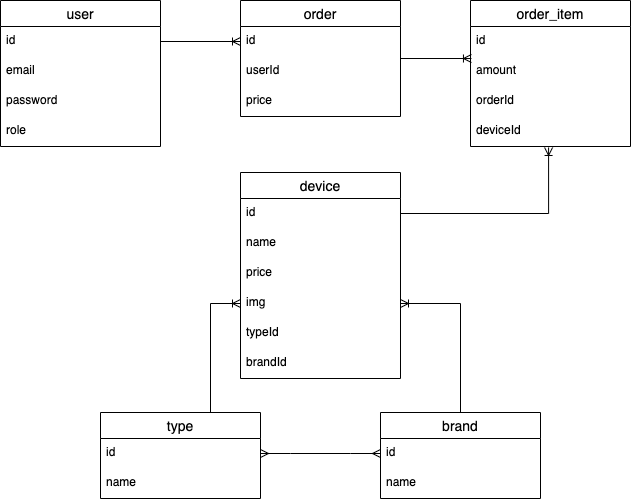
Между таблицами «devices» и «order\_items» связь «Один ко многим», потому, как может быть множество заказов одного устройства, но каждая запись ссылается только на одно конкретное устройство.

Между таблицами «types» и «devices» связь «Один ко многим», так как может быть множество устройств одного типа, но каждое устройство может принадлежать только одному типу.

Между таблицами «brands» и «devices» связь «Один ко многим», так как может быть множество устройств одного бренда, но каждое устройство может принадлежать только одному бренду.

Между таблицами «types» и «brands» связь «Многие ко многим», поскольку у каждого бренда может быть множество типов устройств, так же, как и для каждого типа может быть множество брендов.

Диаграмма разработанной БД



6.2 Проектирование REST-API

У серверной части предполагается пять групп точек доступа:

/types – принимает CRUD-запросы к модели Type

/brands – принимает CRUD-запросы к модели Brand

/devices – принимает CRUD-запросы к модели Device

/users – принимает CRUD-запросы к модели User

/orders – принимает запросы к моделям Order и OrderItem

Описание разработанных точек доступа к сервису

Ниже перечислены все конечные точки и http запросы, которые обрабатываются на этих конечных точках.

/types:

get – отправляет запрос на выдачу списка всех типов устройств;

post – отправляет запрос на создание новой записи типа;

put – отправляет запрос на обновление записи типа;

delete – отправляет запрос на удаление записи типа;

/types/{id}:

get – отправляет запрос на выдачу одного типа устройств по его идентификатору;

put – отправляет запрос на обновление записи типа по ID;

delete – отправляет запрос на удаление записи типа по ID;

/brands:

get – отправляет запрос на выдачу списка всех брендов устройств;

post – отправляет запрос на создание новой записи бренда;

put – отправляет запрос на обновление записи бренда;

delete – отправляет запрос на удаление записи бренда;

/brands/{id}:

get – отправляет запрос на выдачу одного бренда устройств по его идентификатору;

put – отправляет запрос на обновление записи бренда по ID;

delete – отправляет запрос на удаление записи бренда по ID;

/devices:

get – отправляет запрос на выдачу списка всех устройств;

post – отправляет запрос на добавление нового устройства;

/devices/{id}:

get – отправляет запрос на выдачу одного устройства по его идентификатору;

put – отправляет запрос на обновление устройства по его идентификатору;

delete – отправляет запрос на удаление устройства по его идентификатору;

/users:

get – отправляет запрос на выдачу списка всех пользователей;

put – отправляет запрос на обновление полей пользователя;

delete – отправляет запрос на удаление записи пользователя;

/users/registration:

post – отправляет запрос на создание нового пользователя и выдачу токена;

/users/login:

post – отправляет запрос на авторизацию пользователя и выдачу токена;

/users/auth:

get – отправляет запрос на проверку авторизации и обновление токена;

/users/admin:

post – отправляет запрос на создание нового пользователя и выдачу токена. Подразумевается возможность определения роли пользователя. Должно быть доступно только администраторам;

/orders:

post – отправляет запрос на создание нового заказа;

get – отправляет запрос на выдачу списка всех заказов;

/orders/order/{id}:

get – отправляет запрос на выдачу одного заказа по его идентификатору;

delete – отправляет запрос на удаление заказа по его идентификатору;

/orders/items:

get – отправляет запрос на выдачу всех заказанных устройств;

/orders/items/{orderId}:

get – отправляет запрос на выдачу всех заказанных устройств, относящихся к заказу, идентификатор которого указан в параметре в пути;

Реализация сервис-ориентированной системы

7.1 Аналитический обзор платформ и средств построения сервис-ориентированных систем

Существует большое множество вариаций создания сервис-ориентированных систем. Так, среди наиболее популярных инструментов разработки пользовательских интерфейсов можно выделить:

* React – является JavaScript-библиотекой для создания пользовательских интерфейсов. Имеет поддержку как одностраничных, так и мультистраничных приложений.
* Vue.js - фреймворк для создания пользовательских интерфейсов на JavaScript. Используется при написании простых и гибких приложений.
* Angular – фреймворк от компании Google. По сравнению с предыдущими, имеет более высокий порог вхождения в разработку.

Для создания бекенда используются следующие инструменты:

* Java – серверная часть разрабатывается на Java и выполняется на виртуальной машине Java. Используется во многих высоконагруженных корпоративных проектах.
* Node.js – платформа на основе движка V8 от Google, для разработки сервис-ориентированных приложений на JavaScript. Лёгок в освоении, позволяет очень быстро написать рабочий проект.
* Go – компилируемый язык программирования, разработанный компанией Google. Предназначается для разработки высокоэффективных программ. Его преимущество заключается в том, что программы, написанные на Go, транслируются в объектный код и исполняются без использования виртуальной машины.

Среди систем управления базами данных можно отметить:

* PostgreSQL – объектно-реляционная СУБД. Бесплатное ПО с открытым сходным кодом. Отличается надежностью и целостностью данных. Среди недостатков можно выделить определённую сложность при настройке, а также повышенное потребление ресурсов в сравнении с аналогами.
* MySQL – давно зарекомендовавшая себя реляционная СУБД. Однако, демонстрирует не лучшие показатели при работе с большими объёмами данных.
* MongoDB – документоориентированная СУБД. К преимуществам относятся легкость в масштабировании, скорость при поиске и чтении. Рекомендуется в случаях, когда нужно работать с большим количеством не связанных друг с другом данных.

Данную курсовую работу я решил разрабатывать на базе платформы node.js с использованием фреймворка Express для реализации серверной части, по причине низкого порога вхождения в разработку и высокой скорости написания проекта. Для фронтенда были выбраны библиотека React и фреймворк Bootstrap – для создания пользовательского интерфейса, а также библиотека axios для взаимодействия с серверной частью. В качестве СУБД была выбрана PostgreSQL.

7.2 Компонентная модель сервис-ориентированной системы

В данном разделе я опишу клиентскую часть проекта.

Начальной точкой служит файл index.js, внутри которого импортируется пакет ReactDOM, позволяющий управлять DOM-элементами, имеющимися на странице. Его метод createRoot() возвращает объект, метод render() которого инициирует отрисовку приложения. Внутрь него помещается основной компонент приложения, который также импортируется в текущий файл.

Компонент приложения, обычно именуемый App, является функцией, возвращающей пользовательские элементы интерфейса. В своём проекте я использовал одностраничную модель интерфейса, поэтому данный компонент возвращает компонент BrowserRouter, импортированный из пакета react-router-dom, который отвечает за настройку маршрутов в приложении.

Всего в клиентской части реализовано 5 маршрутов:

/ – корневой, главная страница интернет-магазина;

/login – страница авторизации;

/registration – страница регистрации;

/basket – страница корзины, доступна только авторизованным пользователям;

/admin – панель администратора, доступна только пользователям с ролью администратора.

Наборы маршрутов для авторизованных и неавторизованных пользователей определены в отдельном файле routes.js и импортируются в компоненте AppRouter. Этот компонент импортирует в себя три компонента из пакета react-roter-dom – Routes, Route и Navigate. Внутри компонента Routes добавляются компоненты Route, определяющие доступные маршруты. Чтобы, при переходе пользователя по несуществующему пути, его переадресовывало на главную страницу, в последний элемент Route передаётся элемент Navigate, отвечающий за переадресацию, в котором указывается требуемый путь.

У каждой из страниц, для которых был предусмотрен маршрут, имеется отдельный элемент:

* Shop – компонент главной страницы, на которой отображаются страницы товаров.
* Auth – компонент страниц регистрации и авторизации. В зависимости от требуемой операции, изменяет своё содержание, что позволяет обойтись одним элементом для двух маршрутов.
* Basket – компонент страницы корзины. Доступен только авторизованным пользователям.
* Admin – компонент страницы панели администратора. Доступен только администраторам.

Вышеуказанные компоненты были сгруппированы в каталоге pages.

Помимо компонентов страниц, дополнительно реализованы компоненты отдельных элементов интерфейса, сгруппированные в каталоге components:

* NavBar – компонент верхней навигационной панели;
* DeviceList – компонент, отрисовывающий заданный набор устройств, генерируя для каждого устройства компонент DeviceItem;
* DeviceItem – компонент карточки устройства. Включает изображение, имена типа и бренда устройства, его название, цену и кнопки добавления в корзину. Если пользователь не авторизован, при нажатии на кнопку, его перебросит на страницу авторизации и регистрации;
* TypeBar – компонент списка типов устройств. При выборе одного из типов, происходит фильтрация устройств по соответствию выбранному типу;
* BrandBar - компонент списка брендов устройств. При выборе одного из брендов, происходит фильтрация устройств по соответствию выбранному бренду;
* Pages – компонент, создающий кнопки страниц устройств на главной странице;
* BasketItem – компонент товара, добавленного в корзину. Содержит кнопки изменения количества товара в корзине, а также кнопку удаления товара из корзины;
* Statistics – компонент демонстрации аналитики продаж интернет-магазина. Содержит компоненты Plot из пакета react-plotly.js для отрисовки графиков. При загрузке компонента, подгружаются данные о совершенных заказах, обрабатываются и в подготовленном виде передаются в компоненты Plot.

Также, во вложенном каталоге modals, помещены три компонента, отвечающих за отрисовку модальных окон:

* CreateBrand – компонент модального окна для добавления нового бренда;
* CreateType - компонент модального окна для добавления нового типа;
* CreateDevice – компонент модального окна для добавления нового устройства.

7.3 Настройка программного проекта

В клиентской части, используется следующий перечень модулей, используемых для разработки:

Axios - библиотека JavaScript, которая позволяет делать HTTP-запросы из Node.js;

Bootstrap – открытый и бесплатный JS-фреймворк, используется для быстрой верстки адаптивных дизайнов сайтов и веб-приложений;

react-bootstap – компоненты bootstrap, построенные на react;

Dotenv – пакет, который позволяет определить глобальные значения в текстовом файле .env и использовать их в любом месте проекта;

Jwt-decode – пакет, отвечающий за расшифровку токена авторизации;

Mobx - это библиотека, которая даёт инструменты для глобального использования переменных и методов между разными компонентами;

Mobx-react-lite – пакет, связывающий react и mobx;

Plotly.js – библиотека, отвечающая за создание графиков;

React-plotly.js – компонент react из Plotly;

React-dom – фреймворк, который широко применяется и использует виртуальный DOM;

React-router / react-router-dom – библиотека для навигации между разными частями веб-приложения, созданными на React.

В серверной части представлен следующий перечень зависимостей:

Bcrypt - криптографическая библиотека для хеширования паролей;

Cors – пакет посредника (middleware) для Express-приложений, предназначенный для настройки HTTP-заголовков, связанных с CORS (Cross-Origin Resource Sharing – возможность использования ресурсов из другого источника);

Dotenv – пакет, который позволяет определить глобальные значения в текстовом файле .env и использовать их в любом месте проекта;

Express - фреймворк для Node.js, который реализовывает слой функций, необходимых для создания эффективных приложений и API;

Express-fileupload – для простого получения файла в запросе express;

Jsonwebtoken – библиотека для создания (подписания) и подтверждения (проверки) токенов, используемых для аутентификации/авторизации пользователей;

Uuid – библиотека методов для генерации уникальных идентификаторов;

Sequelize – ORM фреймворк для Node.js, который позволяет удобно работать с базами данных, используя объектно-ориентированный подход.

Запуск сервисов осуществлялся локально.

Подключение к клиентской части производится на порт 3000 по следующему адресу:

<http://localhost>:3000/

Подключение к серверной части выполняется на порт 5000 по адресу:

<http://localhost>:5000/api

Секретный ключ шифрования токенов авторизации: random\_secret\_key\_123\_doks\_edition

Подключение к СУБД производится на порт 5432 по адресу:

<http://localhost>:5432/

Название БД: online\_store

Имя пользователя БД: postgres

Пароль БД: 0

7.4 Описание реализации сервисов

В серверной части, управление данными происходит посредством классов-контроллеров. Каждый класс представляет собой обёртку для функций, внутри которых происходит обработка данных из входящего запроса к серверу.

В серверной части реализовано пять контроллеров:

brandController – отвечает за обработку запросов к модели данных Brand;

deviceController – отвечает за обработку запросов к модели данных Device;

orderController – отвечает за обработку запросов к моделям данных Order и OrderItem;

typeController – отвечает за обработку запросов к модели данных Type;

userController – отвечает за обработку запросов к модели данных User.

Методы класса brandController:

create() – отвечает за создание новой записи в таблице «brands»;

update() – отвечает за обновление существующих записей;

delete() – отвечает за удаление существующей записи по имени из query-параметров;

getAll() – возвращает все элементы таблицы;

getOne() – возвращает один элемент из таблицы по ID из параметра в пути;

updateByID() – обновляет элемент по ID из параметра в пути;

deleteByID() – удаляет элемент по ID из параметра в пути.

Методы класса deviceController:

create() – отвечает за создание новой записи в таблице «devices»;

change() – отвечает за обновление существующих записей;

getAll() – возвращает все элементы таблицы;

getOne() – возвращает один элемент из таблицы по ID из параметра в пути;

remove() – удаляет элемент по ID из параметра в пути.

Методы класса orderController:

create() – отвечает за создание новой записи в таблице «orders». Вместе с этим, создаёт записи в таблице «order\_items»;

getOrders() – возвращает все элементы таблицы «orders»;

getOneOrder() – возвращает один элемент из таблицы по ID из параметра в пути;

remove() – удаляет элемент по ID из параметра в пути. Вместе с этим, удаляет все записи из таблицы «order\_items», относящиеся к удаляемому заказу;

getOrderItems() – возвращает все записи устройств, относящихся к заказу, ID которого указан в параметре в пути;

getItems() – возвращает все записи устройств, добавленных в заказы пользователей.

Методы класса typeController:

create() – отвечает за создание новой записи в таблице «types»;

update() – отвечает за обновление существующих записей;

delete() – отвечает за удаление существующей записи по имени из query-параметров;

getAll() – возвращает все элементы таблицы;

getOne() – возвращает один элемент из таблицы по ID из параметра в пути;

updateByID() – обновляет элемент по ID из параметра в пути;

deleteByID() – удаляет элемент по ID из параметра в пути.

Методы класса userController:

create() – отвечает за создание новой записи в таблице «users». Даёт возможность указать роль пользователя;

registration() – отвечает за регистрацию пользователей и создаёт новые записи в таблице. Роль пользователей автоматически определяется, как ‘USER’ (простой пользователь без прав администратора);

login() – отвечает за авторизацию пользователей. Ищет запись по указанному email и сверяет пароль;

check() – отвечает за проверку (и обновление) авторизации;

getAll() – возвращает все элементы таблицы;

updateByID() – производит поиск элемента по email, указанному в теле запроса, и обновляет поля элемента значениями, определёнными так же в теле запроса;

delete() – отвечает за удаление существующей записи по ID из query-параметров.

Ниже приводится соответствие, какие http-запросы инициируют вызов тех или иных методов классов-контроллеров и какие ограничения при этом накладываются:

/types:

get – вызывает метод typeController.getAll

post – вызывает метод typeController.create. Требует роли Администратора.

put – вызывает метод typeController.update. Требует роли Администратора.

delete – вызывает метод typeController.delete. Требует роли Администратора.

/types/{id}:

get – вызывает метод typeController.getOne

put – вызывает метод typeController.updateByID. Требует роли Администратора.

delete – вызывает метод typeController.deleteByID. Требует роли Администратора.

/brands:

get – вызывает метод brandController.getAll

post – вызывает метод brandController.create. Требует роли Администратора.

put – вызывает метод brandController.update. Требует роли Администратора.

delete – вызывает метод brandController.delete. Требует роли Администратора.

/brands/{id}:

get – вызывает метод brandController.getOne

put – вызывает метод brandController.updateByID. Требует роли Администратора.

delete – вызывает метод brandController.deleteByID. Требует роли Администратора.

/devices:

get – вызывает метод deviceController.getAll

post – вызывает метод deviceController.create. Требует роли Администратора.

/devices/{id}:

get – вызывает метод deviceController.getOne

put – вызывает метод deviceController.change. Требует роли Администратора.

delete – вызывает метод deviceController.remove. Требует роли Администратора.

/users:

get – вызывает метод userController.getAll. Требует роли Администратора.

put – вызывает метод userController.update. Требует авторизации.

delete – вызывает метод userController.delete. Требует роли Администратора.

/users/registration:

post – вызывает метод userController.registration.

/users/login:

post – вызывает метод userController.login.

/users/auth:

get – вызывает метод userController.check. Требует авторизации.

/users/admin:

post – вызывает метод userController.create. Требует роли Администратора.

/orders:

post – вызывает метод orderController.create. Требует авторизации.

get – вызывает метод orderController.getOrders. Требует роли Администратора.

/orders/order/{id}:

get – вызывает метод orderController.getOneOrder. Требует роли Администратора.

delete – вызывает метод orderController.remove. Требует роли Администратора.

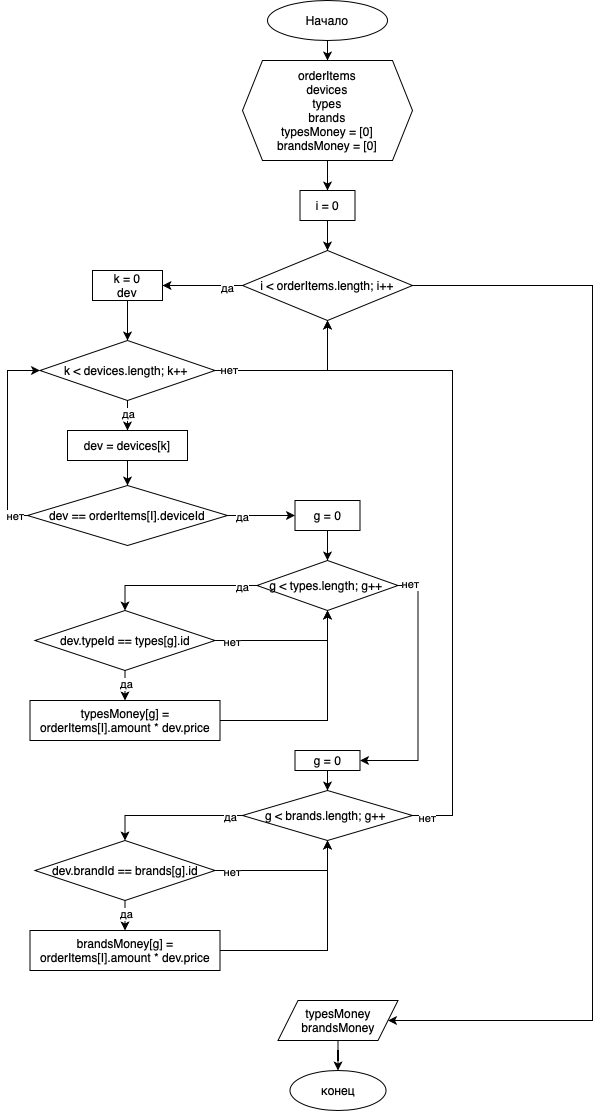
/orders/items:

get – вызывает метод orderController.getItems. Требует роли Администратора.

/orders/items/{orderId}:

get – вызывает метод orderController.getOrderItems. Требует роли Администратора.

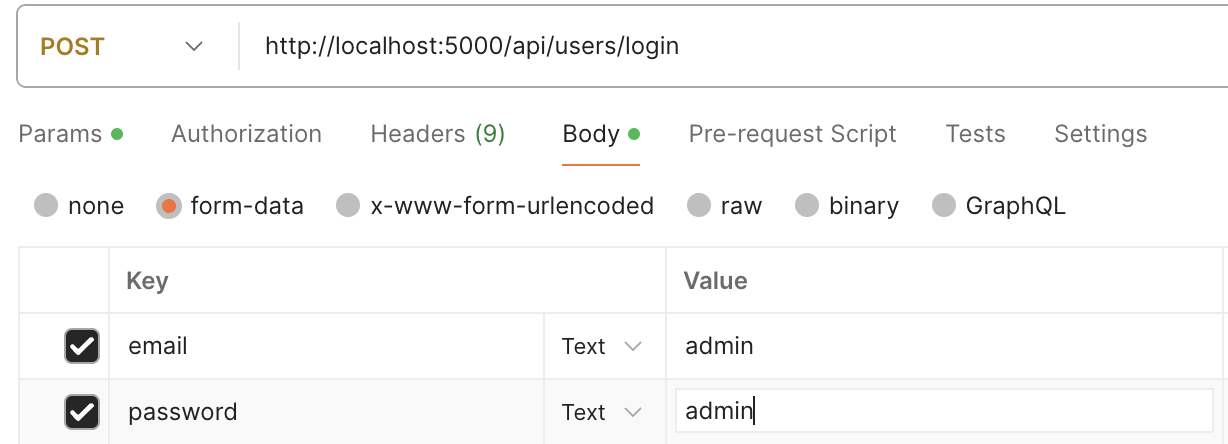
Далее будет приведена блок-схема алгоритма подсчета доходов по отношению к типам и брендам из клиентской части, который выполняется в момент открытия раздела статистики:



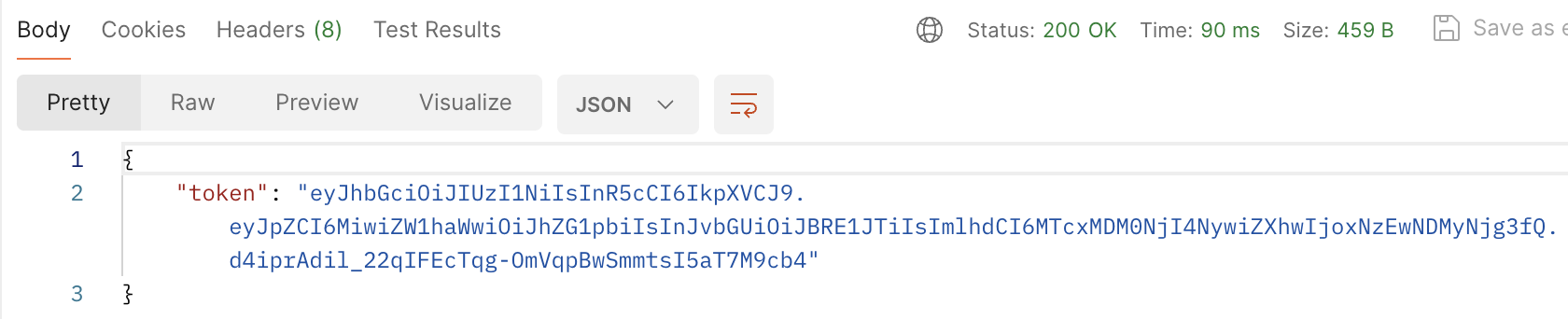
7.5 Описание тестового примера

Протестирую авторизацию, получение списка устройств, получение списка пользователей, создание заказа, получение списка заказов и списка заказанных товаров заказа, удаление заказа.

С помощью postman отправлю запрос на авторизацию:



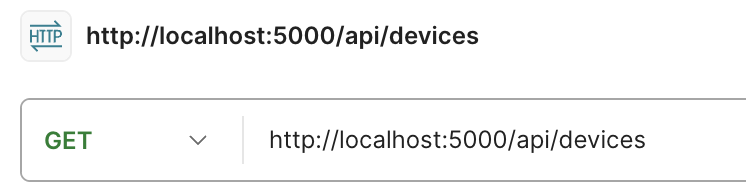
В ответ приходит токен:



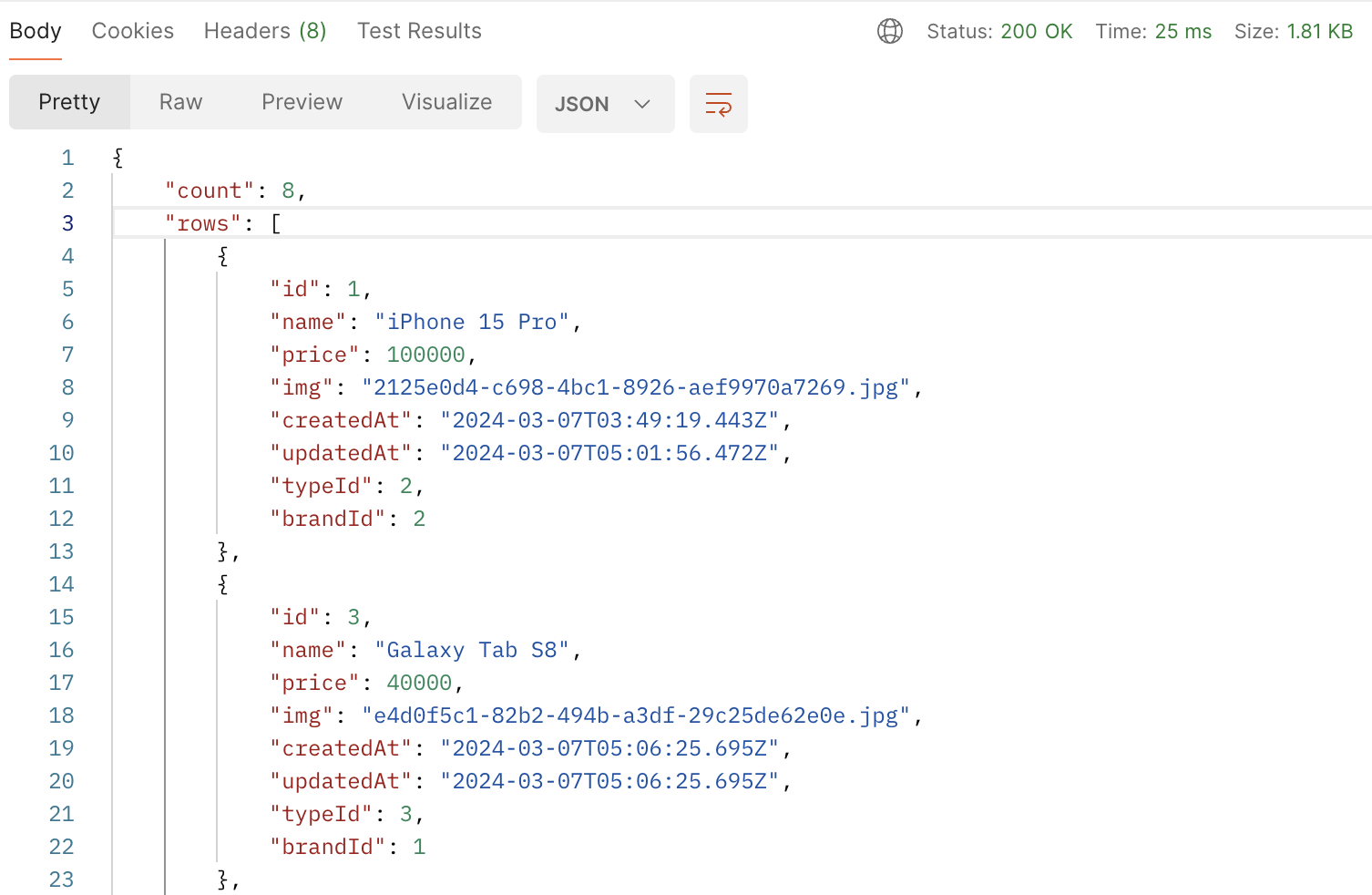
Скопирую токен и добавлю его в поле заголовка Authorization:



Отправлю get запрос для получения списка всех устройств:

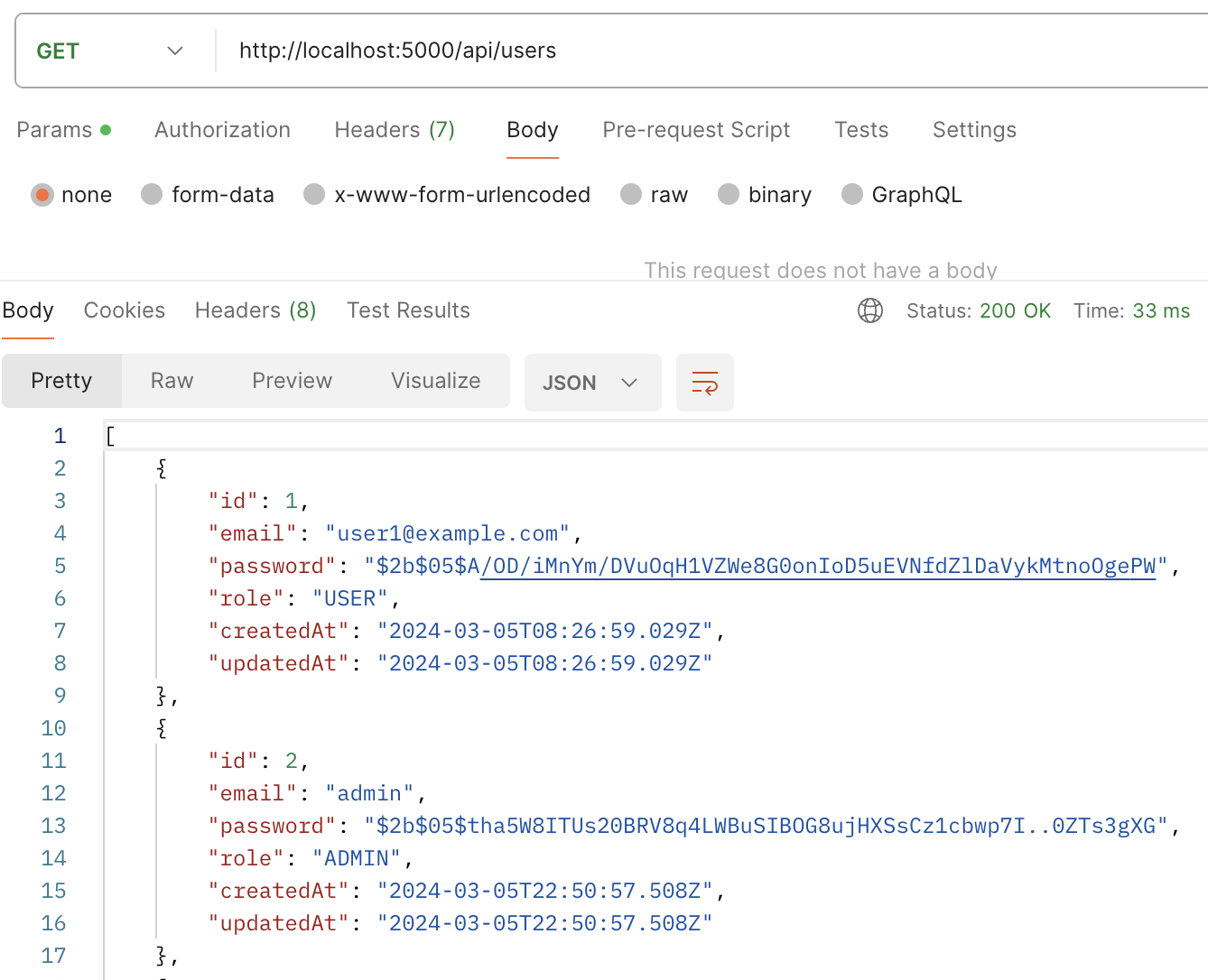


Результат ответа:



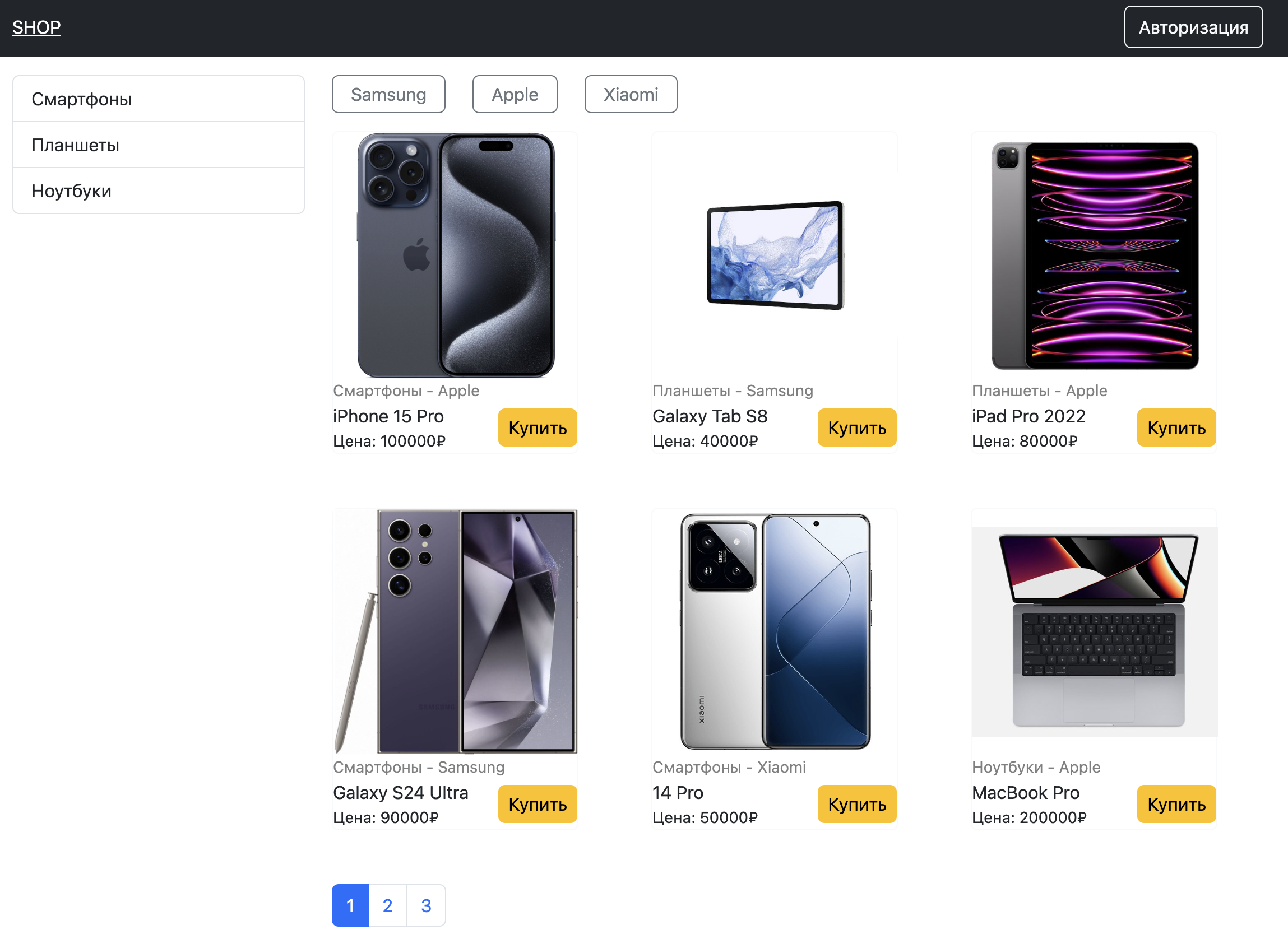
Первым полем значится общее число элементов для расчёта количества страниц товаров. Вторым полем идет список устройств.

Запрошу список пользователей:



Для оформления заказа, воспользуюсь клиентским сервисом.

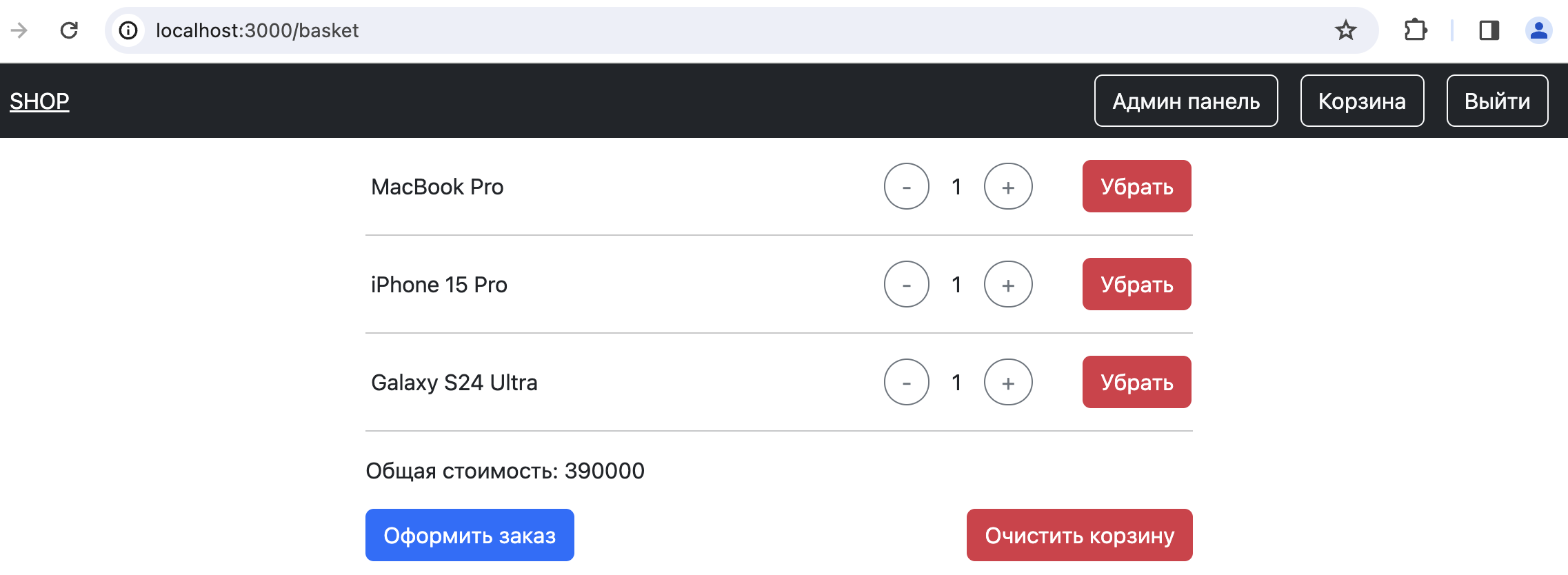
Главная страница сайта выглядит следующим образом:



Нажму кнопку навигационного бара «Авторизация». Появляется окно авторизации:

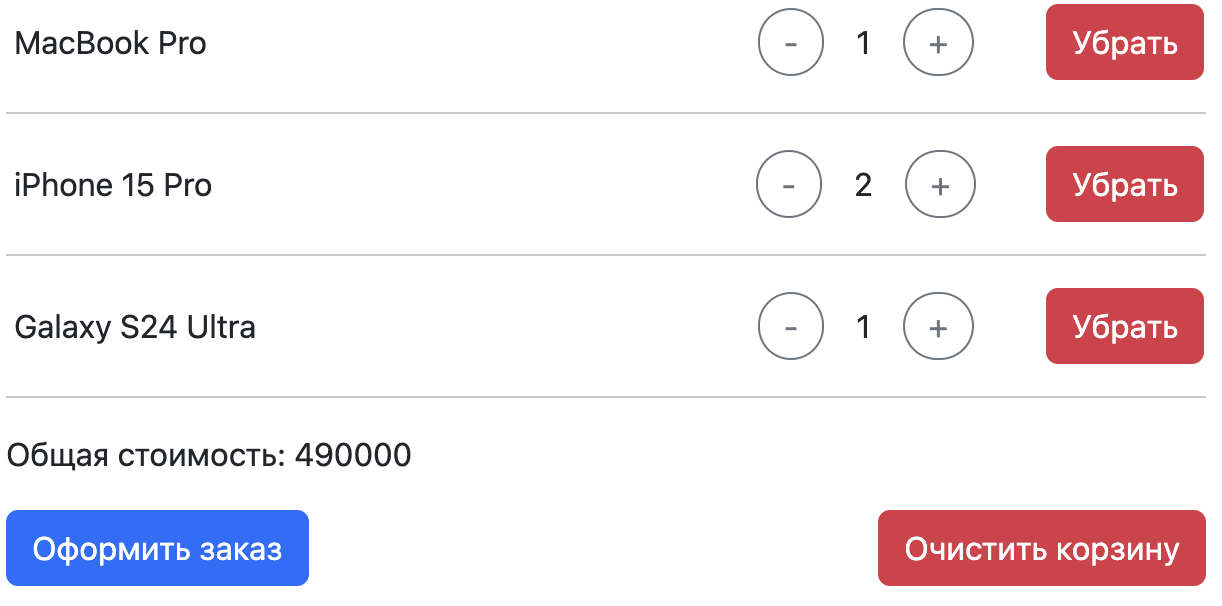


Добавлю несколько товаров в корзину и перейду в соответствующий раздел:

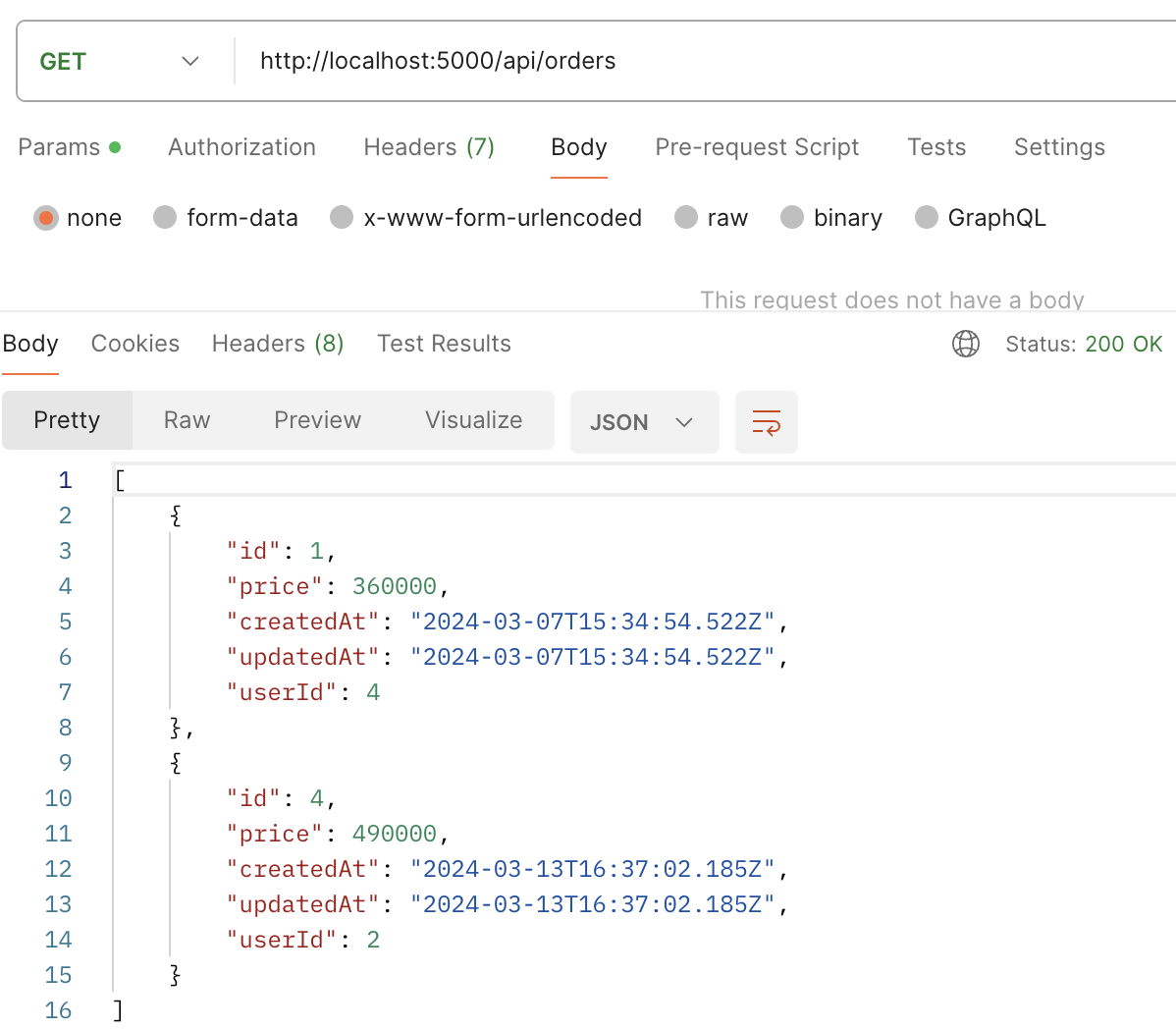


На выше приведённом изображении можно также наблюдать, как в навигационном баре появились разделы панели администратора и корзины.

При изменении количества товара, общая сумма также пересчитывается:



Нажму кнопку «Оформить заказ» и перейду в postman. Сделаю запрос на получение заказов:

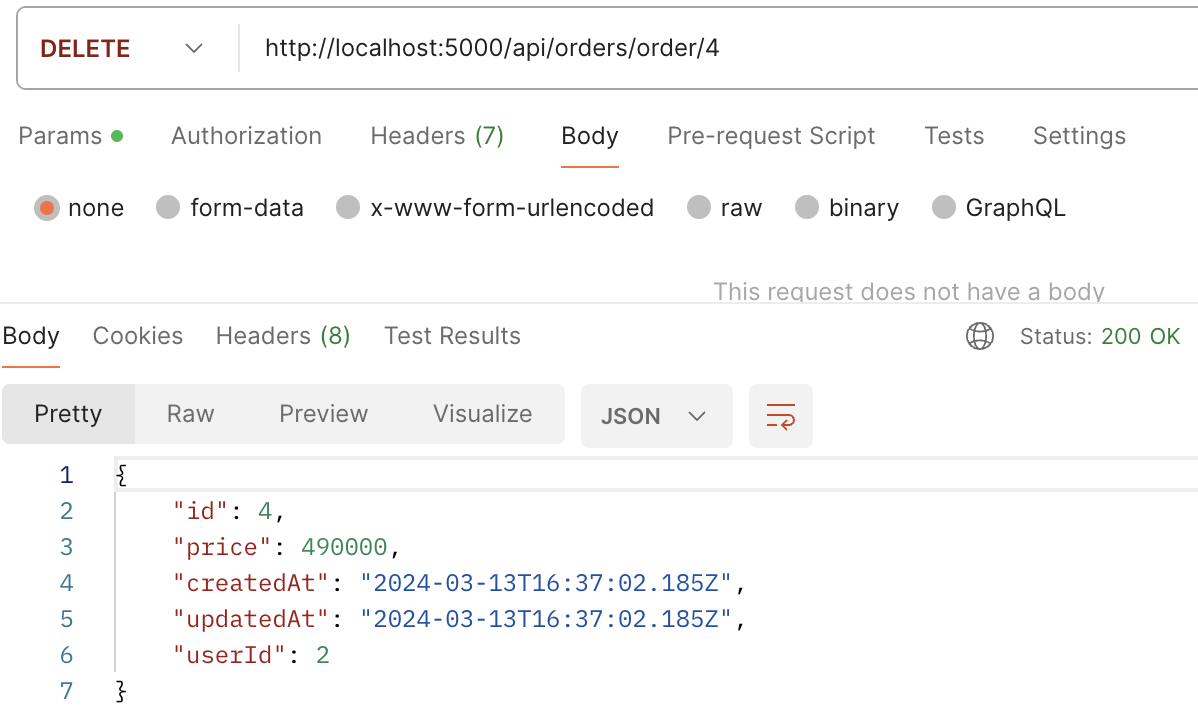


Вторым элементов в списке является созданный мной заказ. Запрошу список заказанных устройств для данного заказа:

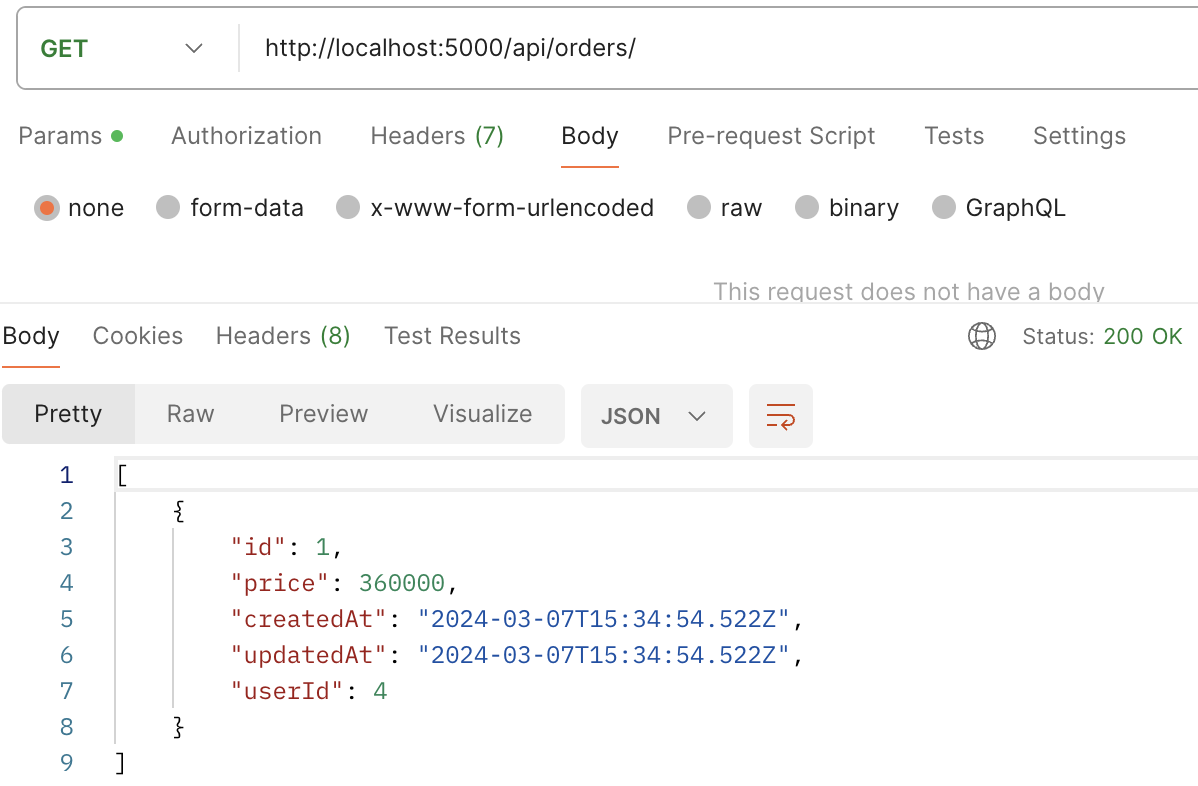


Здесь можно наблюдать ровно тот же состав, какой присутствовал в корзине.

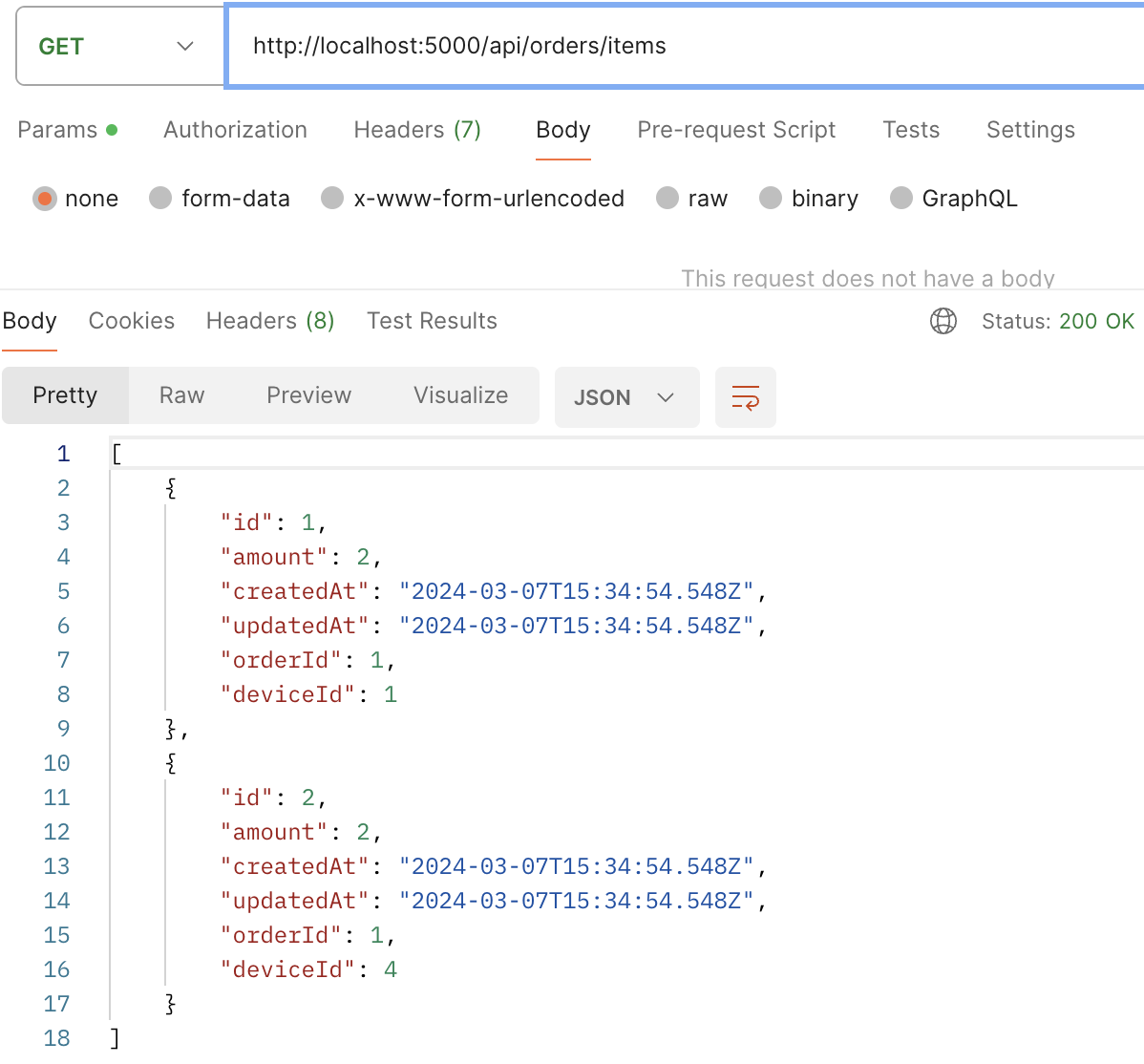
Теперь, отправлю запрос delete для удаления заказа:



Сделаю проверку, что заказ был удалён:

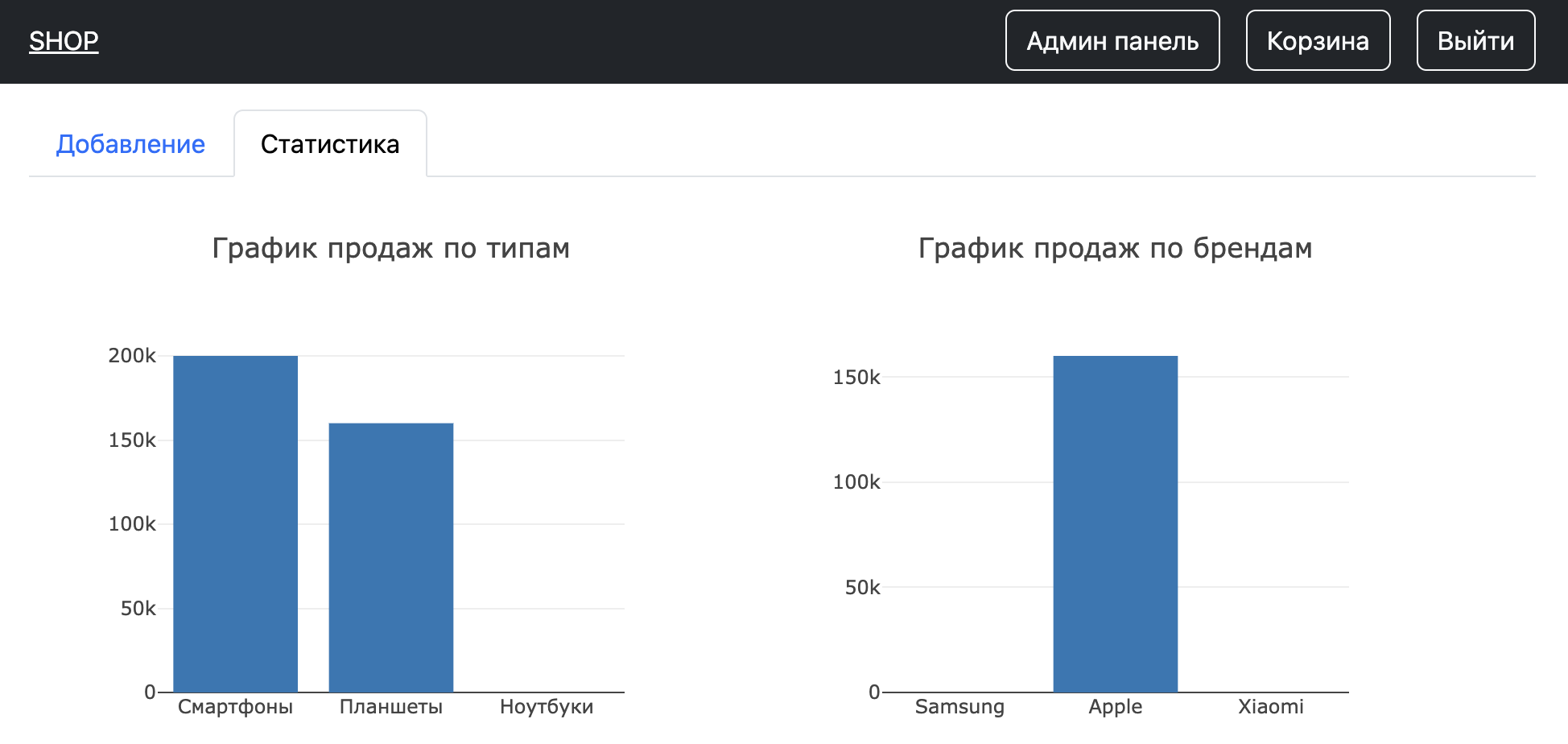


В списке заказов – не значится.



При запросе списка всех заказанных устройств, записи по удалённому заказу отсутствуют.

В дополнение, продемонстрирую раздел аналитики продаж в панели администратора, где, благодаря компоненту Plot пакета Plotly на основе данных строятся графики:



Заключение

Несмотря на скромную реализацию поставленной задачи, каждый разработанный элемент функционирует в соответствии с ожиданиями.

К явным недостаткам можно отнести незащищённую передачу логина и пароля и отсутствие поддержки защищённого соединения.

К менее значимым недостаткам можно отнести не кешируемую корзину, которая сбрасывается при каждой перезагрузке страницы. Однако, в перспективе, это решается добавлением поддержки cookie-файлов.

В список доработок и улучшений можно добавить такие вещи, как:

* Личный кабинет;
* Возможность просматривать совершенные заказы;
* Возможность отменять заказы;
* Запрос контактных данных для совершения заказа;
* Возможность редактировать категории и устройства в панели администратора из клиентского приложения (сервер имеет все необходимые интерфейсы);
* Аналитику продаж можно отделить в обособленный сервис, подключаемый к серверу.

На этом список не заканчивается.

В процессе выполнения данной работы я получил значимый опыт в разработке сервис-ориентированной системы и лучшее понимание деталей организации и взаимодействия сервисов, что поможет мне в дальнейшем углубляться в изучение данной темы и пробовать новые инструменты.

Список литературы

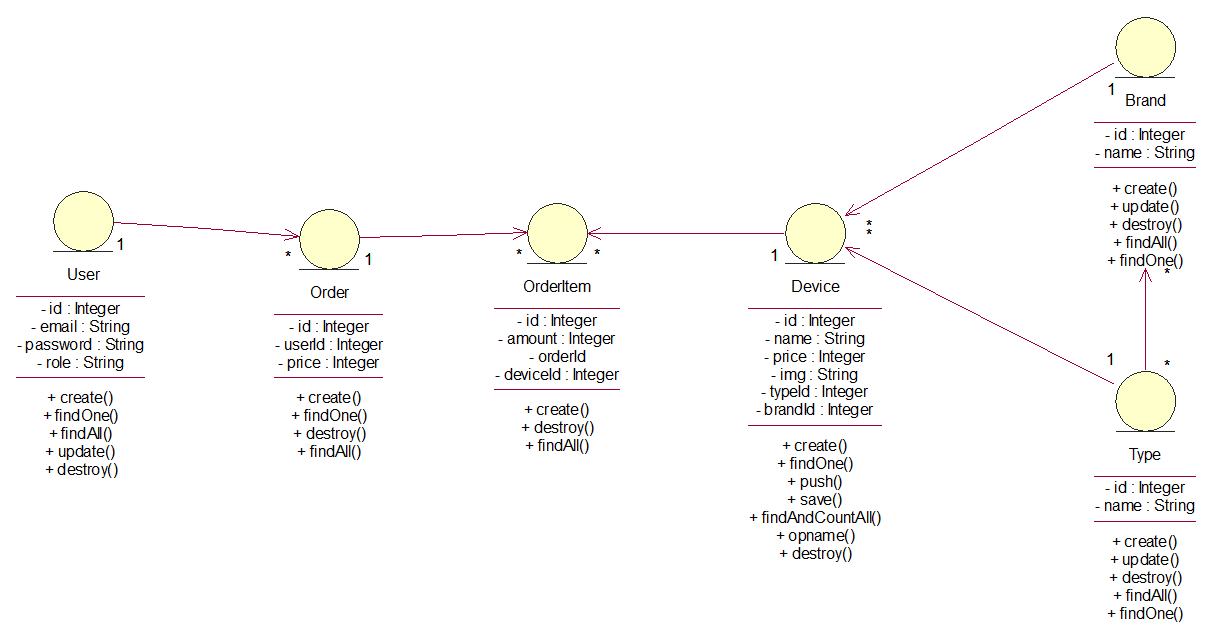
<https://google.com/>

https://youtube.com/

Приложения

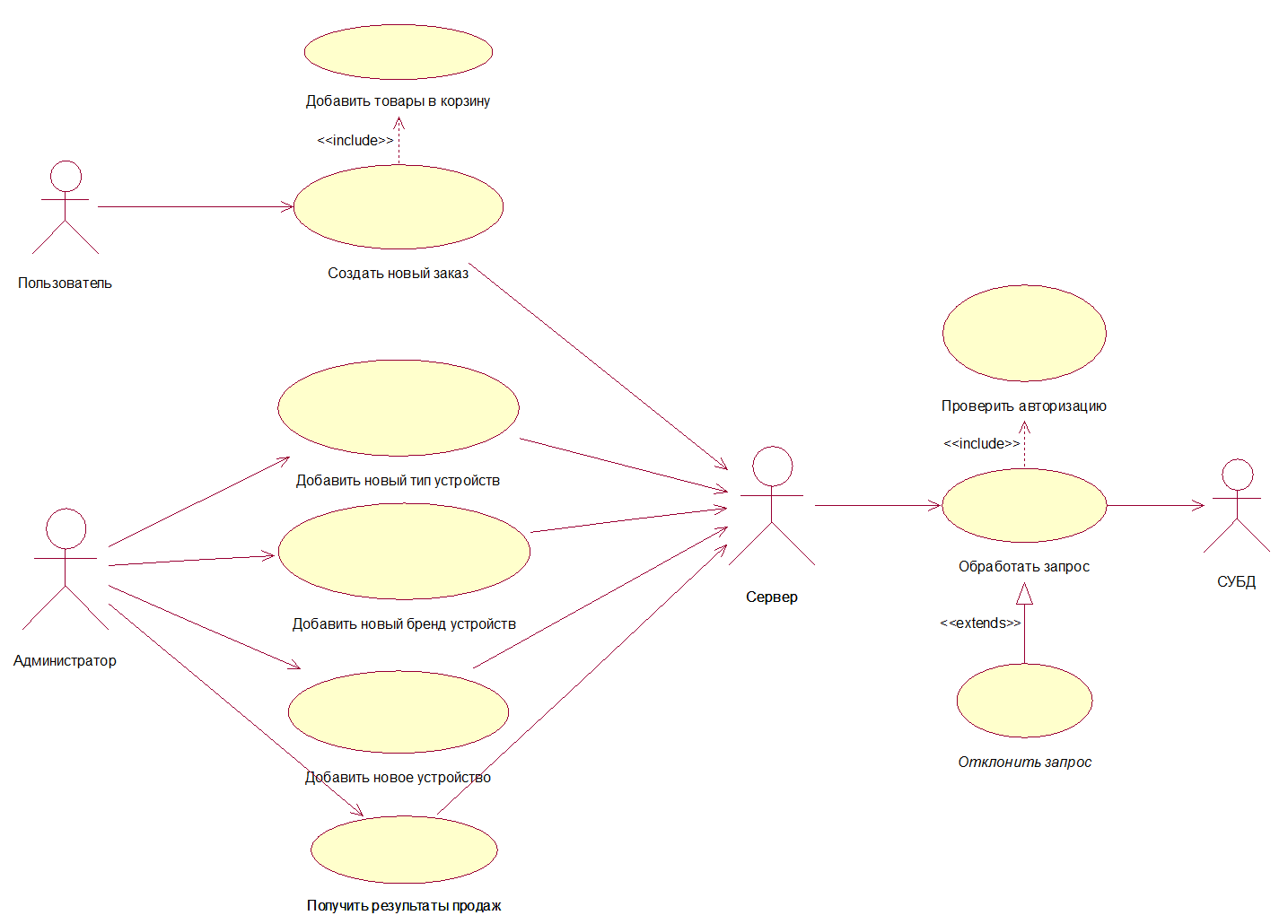
Приложение 1

Диаграмма классов сущностей



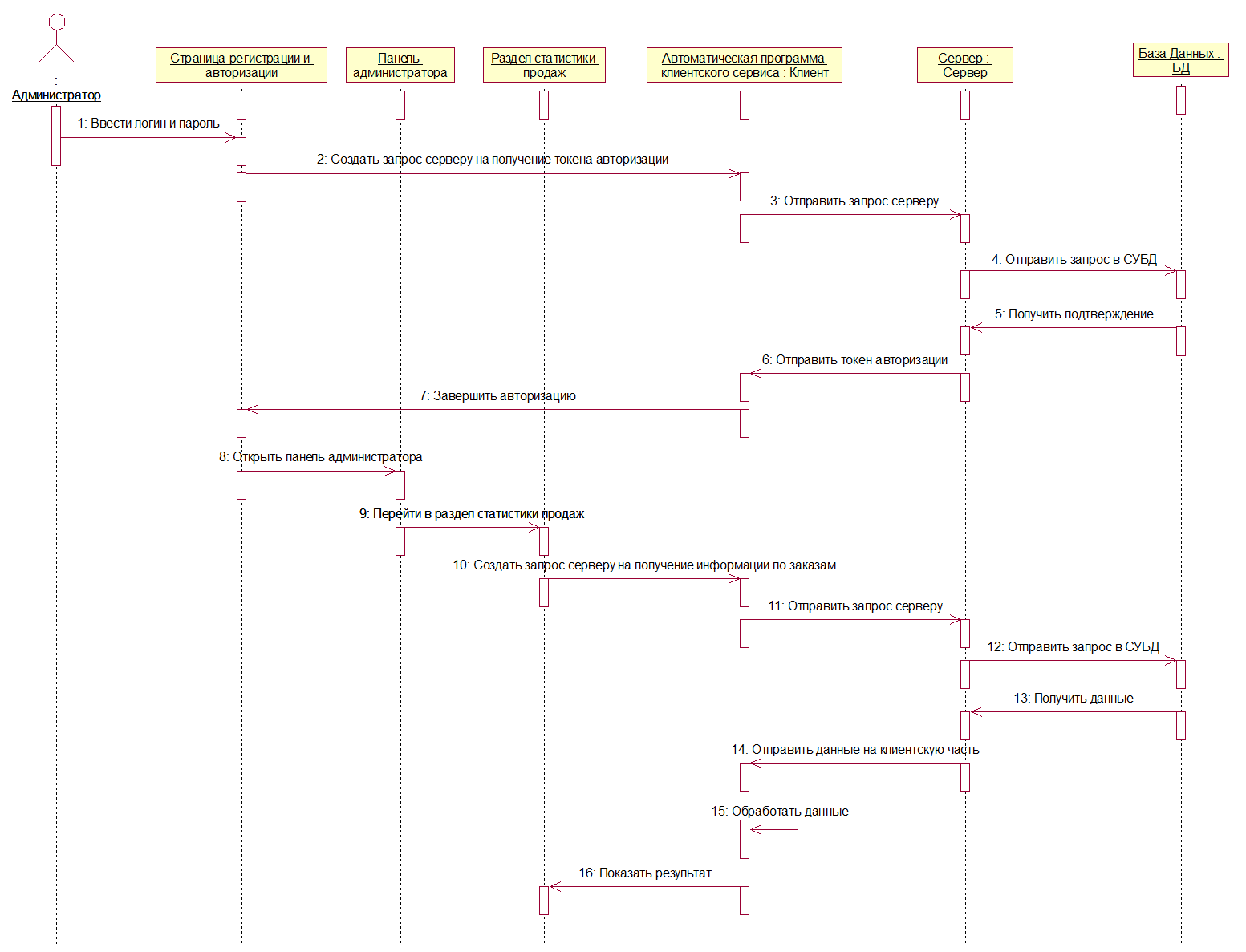
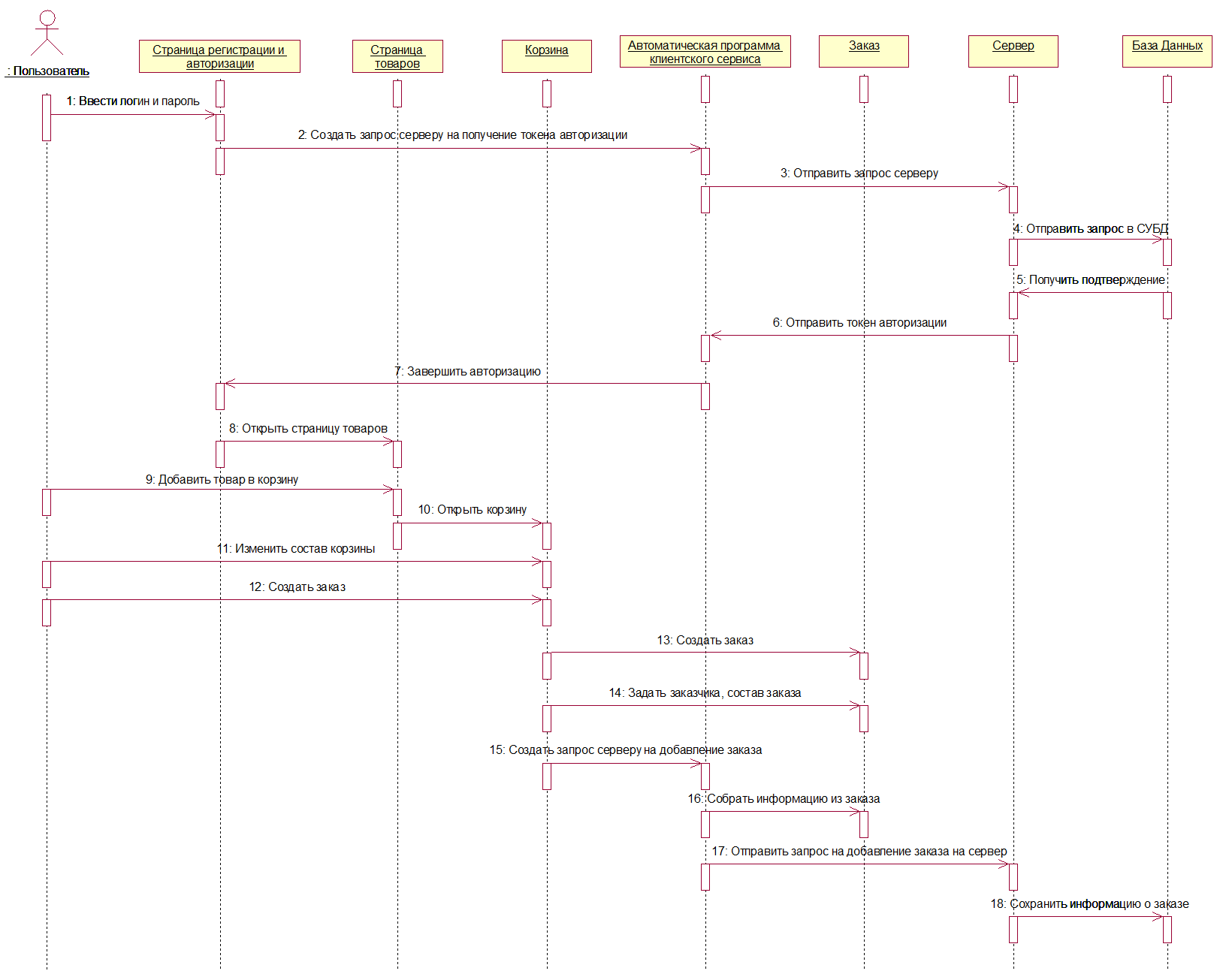
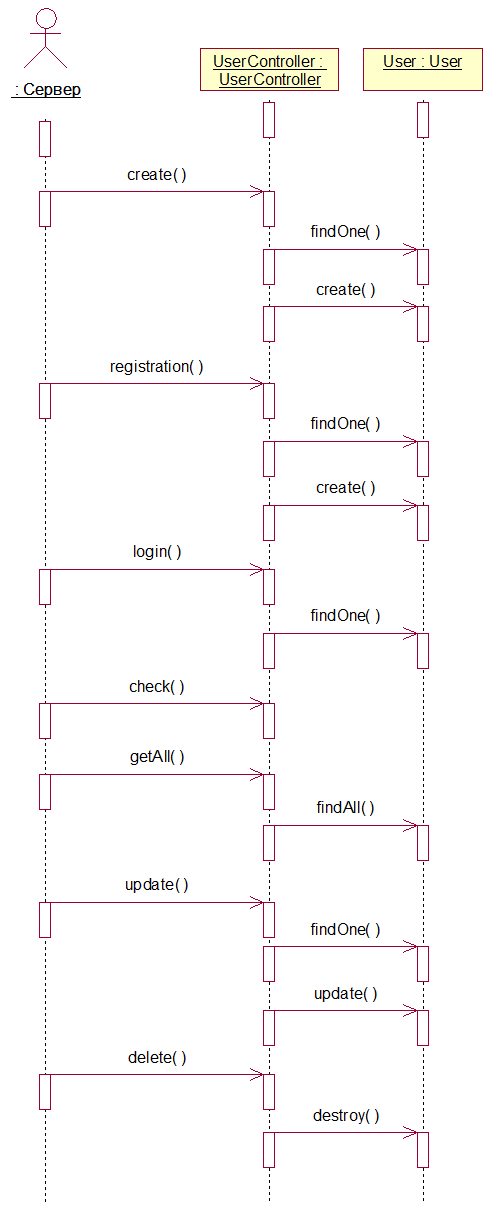
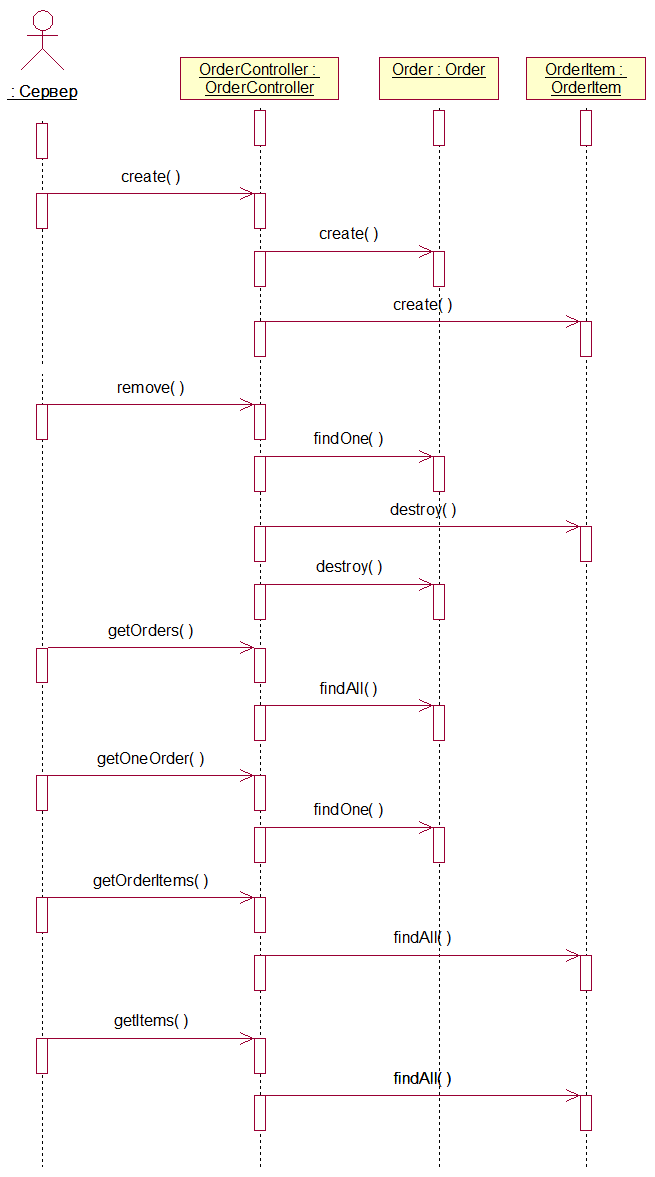
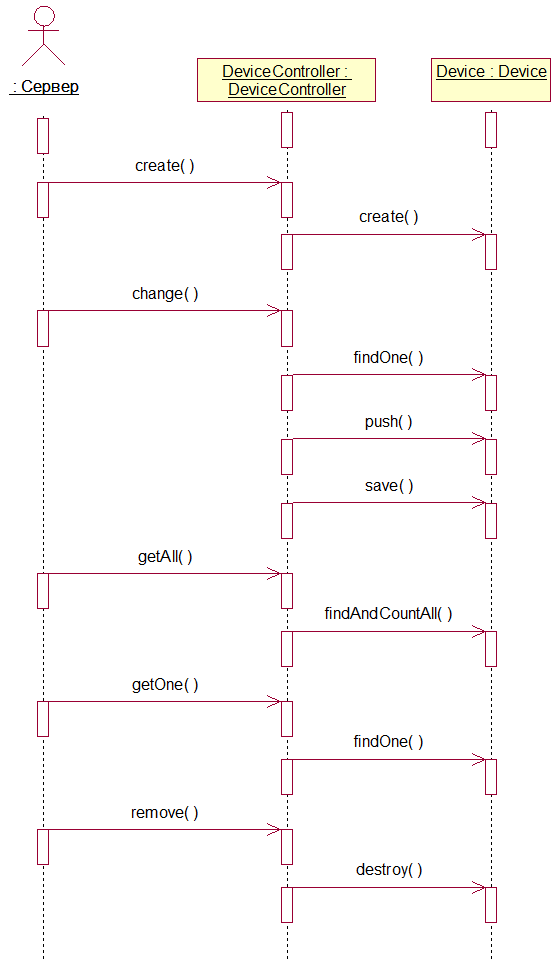
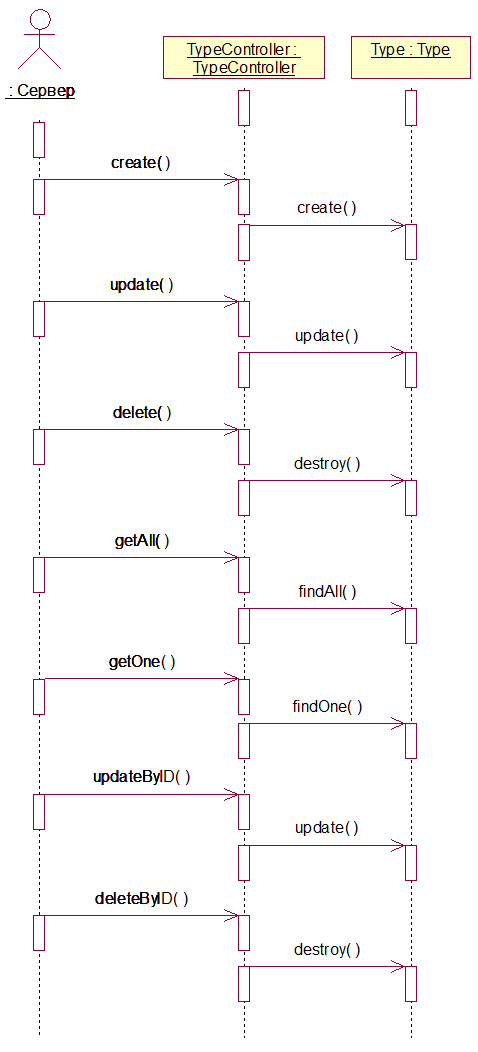
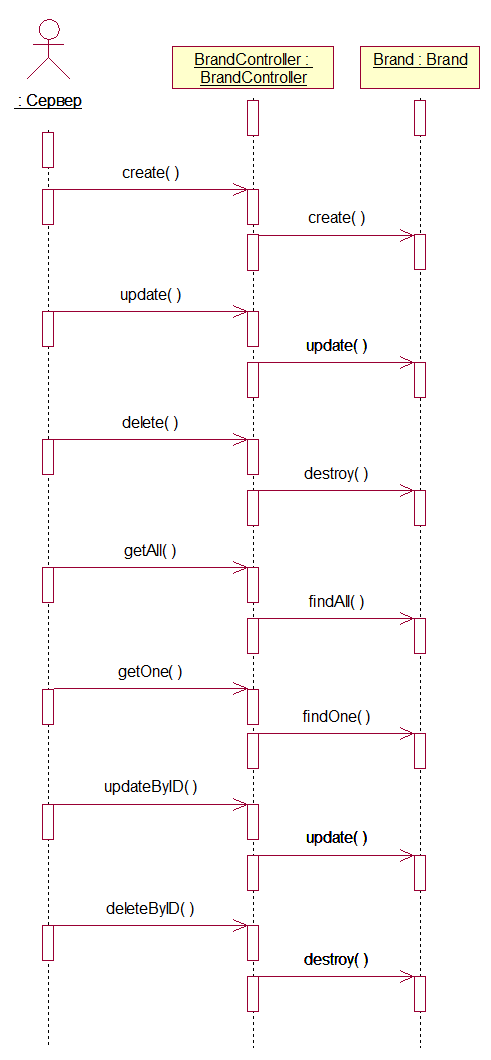
Приложение 2

Диаграмма вариантов использования



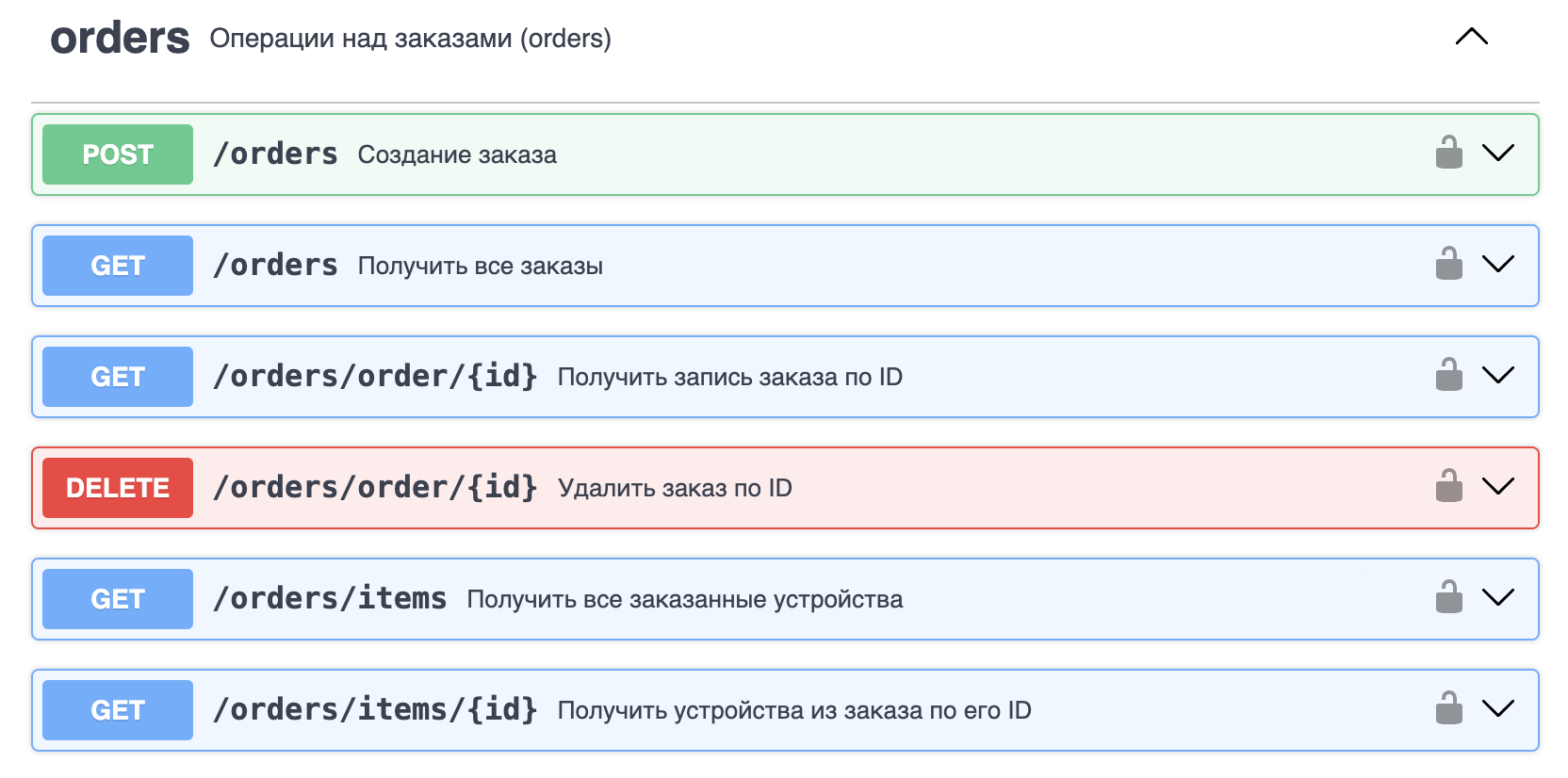
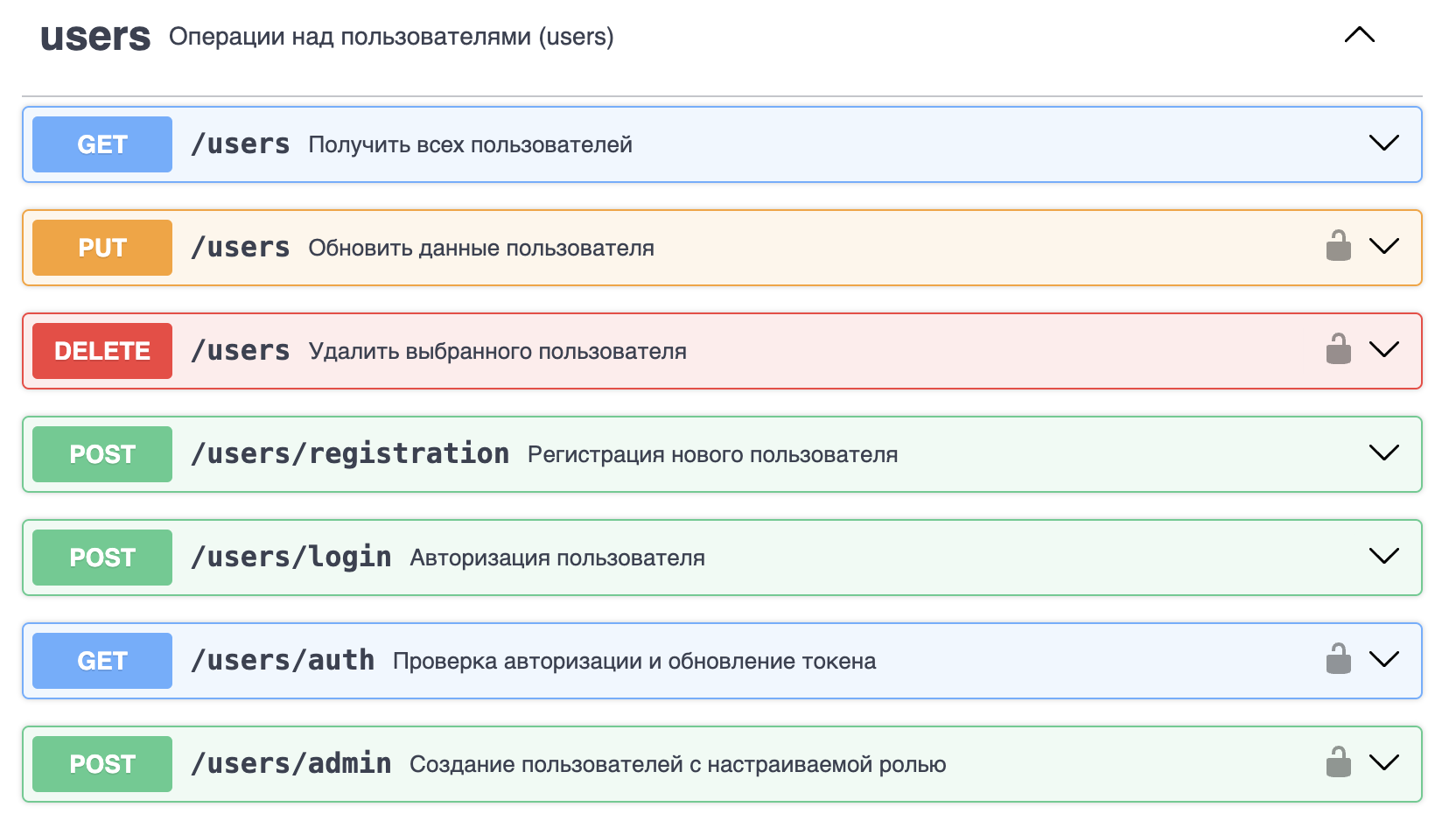
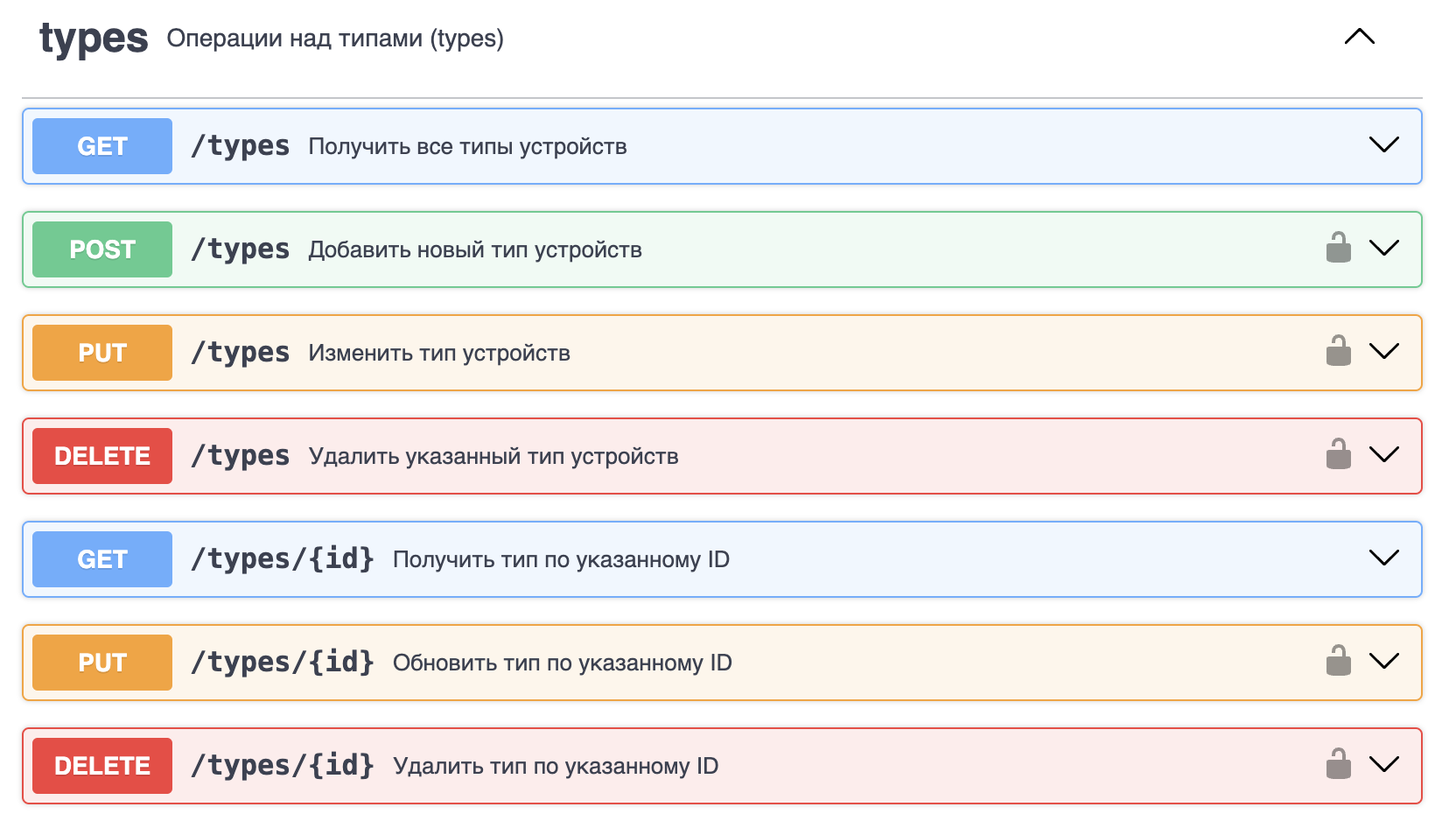
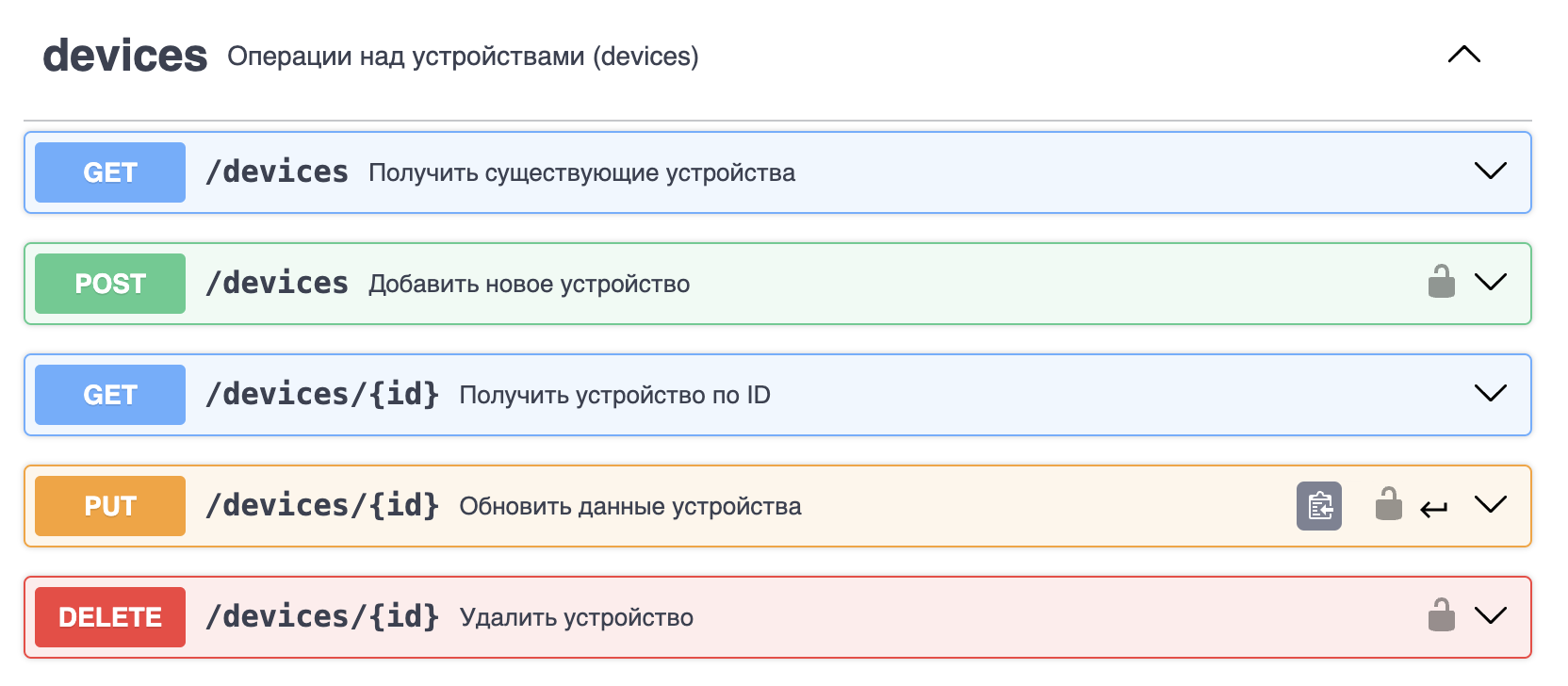
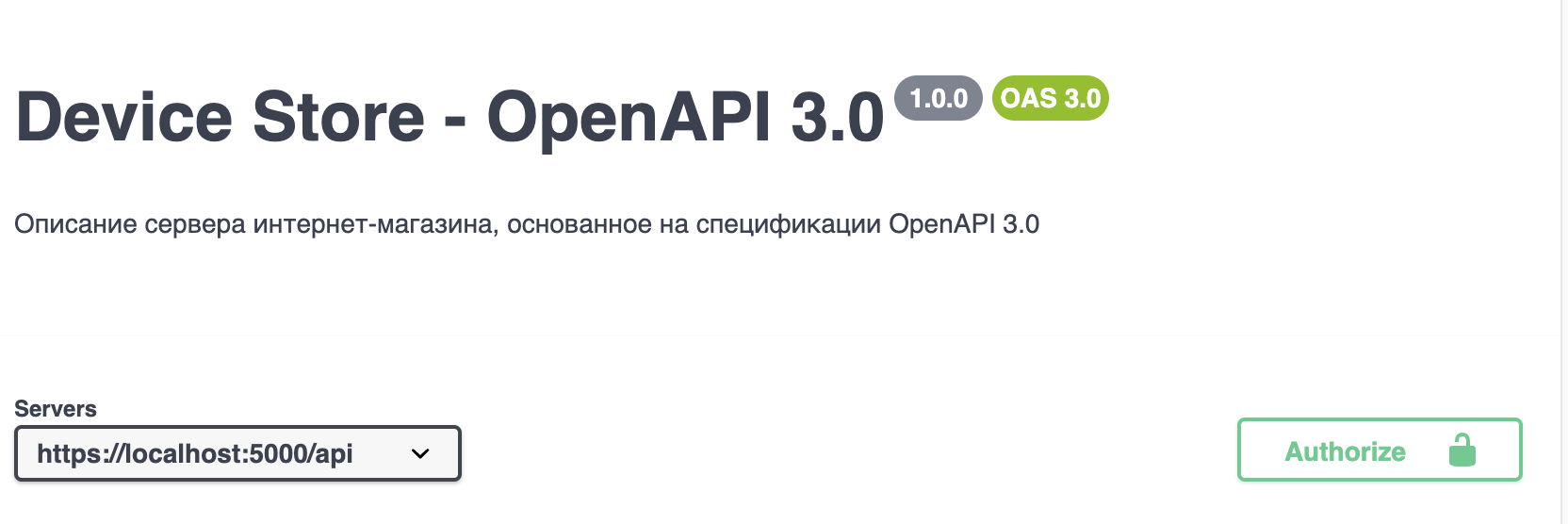
Приложение 3

Диаграммы взаимодействия (sequence diagrams)



Приложение 4

Swagger Editor



Приложение 5

Диаграмма компонентов клиентской части

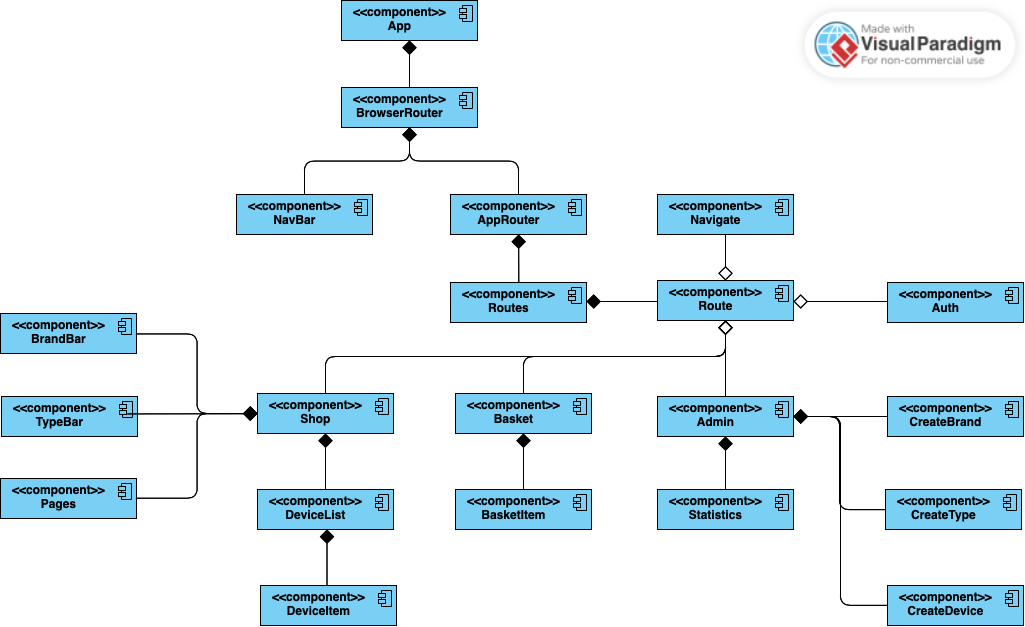


Диаграмма компонентов проекта

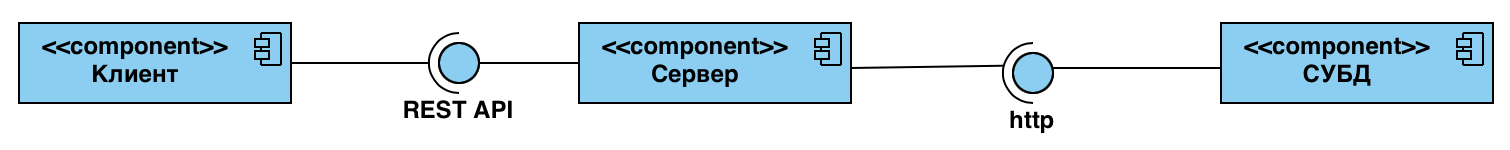


Диаграмма развертывания для интернет-магазина

