**ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**«ТОМСКИЙ ЭКОНОМИКО-ПРОМЫШЛЕННЫЙ КОЛЛЕДЖ»**

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА REST API СЕРВЕРА ИНТЕРНЕТ-МАГАЗИНА «ANIBLOOM» В ФРЕЙМВОРКЕ LARAVEL**

***КУРСОВОЙ ПРОЕКТ***

*СПЕЦИАЛЬНОСТЬ 09.02.07 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ*

Выполнил:

Студент группы 1912с

\_\_\_\_\_\_\_\_\_\_\_\_ / Заикин В.А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2024г.

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_/ Уляхин В.А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2024г.

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

Томск 2024

**Оглавление**

[ВВЕДЕНИЕ 3](#_Toc163861656)

[ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc163861657)

[ТЕХНОЛОГИИ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ 5](#_Toc163861658)

[Используемые языка программирования и фреймворки 5](#_Toc163861659)

[Используемое программное обеспечение 5](#_Toc163861660)

[ГЛАВА 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ИНФОРМАЦИОННОГО ПРОДУКТА. 7](#_Toc163861661)

[Описание проекта и задач 7](#_Toc163861662)

[Специфические требования к API 9](#_Toc163861663)

[Функционал авторизированного пользователя 12](#_Toc163861664)

[Функционал администратора 17](#_Toc163861665)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ 22](#_Toc163861666)

[Концептуальная модель данных 22](#_Toc163861667)

[Нормализация данных 23](#_Toc163861668)

[Логическая модель данных 24](#_Toc163861669)

[Физическая модель данных 25](#_Toc163861670)

[Словарь данных 27](#_Toc163861671)

[Диаграмма базы данных 29](#_Toc163861672)

[ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФУНКЦИОНАЛА ИНФОРМАЦИОННОГО ПРОДУКТА 30](#_Toc163861673)

[Установка и настройка фреймворка Laravel 30](#_Toc163861674)

[Создание миграций 30](#_Toc163861675)

[Создание класса исключений 32](#_Toc163861676)

[Создание класса посредника 33](#_Toc163861677)

[Генерация классов моделей 34](#_Toc163861678)

[Валидация запроса формы 38](#_Toc163861679)

[Генерация классов контроллера 46](#_Toc163861680)

[Маршрутизация 56](#_Toc163861681)

[ГЛАВА 4. АВТОТЕСТЫ 58](#_Toc163861682)

[ГЛАВА 5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 71](#_Toc163861683)

[Функционал пользователя 71](#_Toc163861684)

[Функционал авторизированного пользователя 77](#_Toc163861685)

[Функционал администратора 86](#_Toc163861686)

[ЗАКЛЮЧЕНИЕ 94](#_Toc163861687)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 95](#_Toc163861688)

# ВВЕДЕНИЕ

API представляет собой механизмы, которые позволяют двум программным компонентам взаимодействовать друг с другом, используя определенные наборы определений и протоколов.

API расшифровывается как Application Programming Interface, что означает программный интерфейс приложения. В контексте API термин "приложение" применяется к любому программному обеспечению с определенной функциональностью. Интерфейс можно рассматривать как сервисный контракт между двумя приложениями, определяющий, как они взаимодействуют друг с другом с помощью запросов и ответов. Документация API содержит информацию о том, как разработчики должны структурировать эти запросы и ответы.

Архитектура API обычно рассматривается с точки зрения клиента и сервера. Приложение, отправляющее запрос, называется клиентом, а приложение, отправляющее ответ, - сервером. Таким образом, в примере с погодными данными, база данных службы погоды является сервером, а мобильное приложение - клиентом.

Одной из ключевых особенностей REST API является отсутствие сохранения состояния. Это означает, что серверы не хранят данные о клиенте между запросами. Запросы клиента к серверу подобны URL-адресам, которые вы вводите в браузере для посещения веб-сайтов. Ответ от сервера представляет собой простые данные, не включающие типичное графическое представление веб-страницы.

Целью курсового проекта является проектирование и разработка REST API интернет-магазина «Anibloom» в фреймворке Laravel.

Для достижения поставленной цели необходимо было решить следующие задачи:

1. Произвести анализ предметной области.
2. Написать техническое задание на разработку информационного продукта.
3. Спроектировать и разработать базу данных.
4. Спроектировать и разработать функционал информационного продукта.

# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Интернет-магазин «Anibloom» специализируется на продаже аниме-товаров и сопутствующих аксессуаров. Магазин предоставляет ассортимент товаров, включая мангу, кружки, дакимакуры, значки и другие предметы, популярные среди поклонников аниме и манги.

Клиенты магазина «Anibloom» имеют возможность оформлять заказы на сайте, просматривать каталог товаров, добавлять товары в корзину, отслеживать статус своих заказов. Кроме того, пользователи могут оставлять отзывы и оценки на товары, что помогает другим покупателям принять информированное решение при покупке.

# ТЕХНОЛОГИИ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

## Используемые языка программирования и фреймворки

**PHP** — C-подобный скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов.

**SQL** — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

**Laravel** — бесплатный веб-фреймворк с открытым кодом, предназначенный для разработки с использованием архитектурной модели MVC (англ. Model View Controller — модель-представление-контроллер). Laravel выпущен под лицензией MIT.

## Используемое программное обеспечение

**Draw.io** — инструмент для создания диаграмм, блок-схем, интеллект-карт, бизнес-макетов, отношений сущностей, программных блоков и другого. Сервис распространяется на бесплатной основе с открытым исходным кодом. Draw.io обладает богатым набором функций для визуализации большинства задач пользователя.

При входе на сервис пользователь сразу попадает в рабочий интерфейс. У пользователя нет возможности для авторизации или регистрации, есть только опция выбора места для экспорта проекта. Процесс создания проекта выглядит следующим образом: пользователь перетаскивает из левой панели фигуры или элементы на рабочую поверхность, затем изменяет их — изменяет цвет, размер, шрифт текста, свойства фигуры (прозрачность, форма и т. д.). Draw.io позволяет отслеживать и восстанавливать изменения готовых проектов, импортировать и экспортировать в PDF, PNG, XML, VSDX, HTML, а также автоматически публиковать и делиться работами.

Инструмент работает с Google Диск, Google Workspace и Dropbox, глубоко интегрирован и удобен для работы с продуктами Confluence и Jira от Atlassian. Пользователи также могут работать с диаграммами в автономном режиме и сохранять их локально, используя настольное приложение для персональных компьютеров.

Инструмент позволяет создавать: графики, диаграммы, таблицы, презентации, блок-схемы, планы помещений, воронки продаж, ментальный карты, карты сайтов.

**Open Server Panel** — это портативная программная среда, созданная специально для веб-разработчиков с учётом их рекомендаций и пожеланий. Данный программный комплекс включает в себя тщательно подобранный набор серверного программного обеспечения, а также невероятно удобную и продуманную управляющую утилиту, которая обладает мощными возможностями по администрированию и настройке всех доступных компонентов. OSPanel широко используется с целью разработки, отладки и тестирования веб-проектов, а также для предоставления веб-сервисов в локальных сетях.

**PhpStorm** — коммерческая кроссплатформенная интегрированная среда разработки для PHP, обладающая широкими возможностями анализа кода и поддержкой основных фреймворков.

**Postman** — это платформа API, позволяющая разработчикам проектировать, создавать, тестировать и повторять свои API. По состоянию на февраль 2023 года Postman сообщает о более чем 25 миллионах зарегистрированных пользователей и 75 000 открытых API, которые, по его словам, представляют собой крупнейший в мире публичный API-хаб. Штаб-квартира компании находится в Сан-Франциско и имеет офис в Бангалоре, где она была основана.

# ГЛАВА 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ИНФОРМАЦИОННОГО ПРОДУКТА.

## Описание проекта и задач

**ОБЩЕЕ ОПИСАНИЕ ПРОЕКТА**

Все пользователи системы подразделяются на две группы:

1. Администраторы
2. Пользователи

Категории товаров подразделяются на несколько групп

1. Манга
2. Кружки
3. Дакимакуры
4. Значки

Функционал пользователя:

* Регистрация
* Авторизация
* Просмотр категорий товаров
* Просмотр товаров определенной категории

Функционал авторизированного пользователя

* Просмотр своего профиля
* Выход
* Добавление товара в корзину
* Просмотр своей корзины
* Оформление заказа
* Оставление отзыва для определённого товара
* Редактирование корзины
* Редактирование своего профиля

Функционал администратора

* Добавление категории
* Добавление товара
* Редактирование категории
* Редактирование товара
* Удаление категории
* Удаление товара

Задача курсового проекта – реализовать REST API заданной структуры

В примерах будет использоваться переменная {{host}} которая обозначает адрес информационного продукта.

**ОБЩИЕ ТРЕБОВАНИЯ**

API должно быть доступно с других доменов.

Идентификация пользователя организуется посредством Bearer Token.

При попытке доступа авторизованным пользователем к функциям недоступным для своей группы во всех запросах необходимо возвращать ответ следующего вида:

Status: 403

Content-Type: application/json

Body:

{  
 "error": {  
 "code": 403,  
 "message": "Forbidden for you"  
 }  
}

При попытке доступа неавторизированным пользователем к функциям требующей авторизации во всех запросах необходимо возвращать ответ следующего вида:

Status: 403

Content-Type: application/json

Body:

{  
 "error": {  
 "code": 401,  
 "message": "Unauthorized"  
 }  
}

В случае ошибок связанных с валидацией данных во всех запросах необходимо возвращать следующее тело ответа:

{  
 "error": {  
 "code": <code>,  
 "message": <message>,  
 "errors": {  
 <key>: [<error message>]  
 }  
 }  
}

Вместо <code> и <message> необходимо указывать соответствующее значение, определенное в описании ответа на соответствующий запрос. В свойстве error.errors необходимо перечислить те свойства, которые не прошли валидацию, а в их значениях указать массив с ошибками валидации.

Например, если отправить пустой запрос на сервер, где проверяется следующая валидация:

* phone – обязательно поле
* password – обязательное поле

то тело ответа будет следующим:

{  
 "error": {  
 "code": 422,  
 "message": "Validation error",  
 "errors": {  
 phone: ["field phone can not be blank"],  
 password: ["field password can not be blank"]  
 }  
 }  
}

## Специфические требования к API

**Регистрация**

Запрос для регистрации нового пользователя в системе. При отправке запроса необходимо передать FormData со следующими полями:

* name – обязательное поле, строка;
* surname – обязательное поле, строка;
* patronymic – не обязательное поле, строка;
* login – обязательное и уникальное поле, строка;
* password – обязательное поле, строка;
* birth – обязательное поле, дата, формат Y-m-d;
* telephone – обязательное поле, формат число;
* email – обязательное поле, строка;

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/register  **Method:** POST  **Headers:** Content-Type: application/json  **Body:**  {  "surname": "Usertest",  "name": " Usertest",  "patronymic": " Usertest",  "login": " Usertest",  "password": " Usertest1!",  "birth": "12.02.2003",  "telephone": "7930573565",  "email": "usertest@mail.ru"  } | Успешная регистрация  **Status:** 201  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Регистрация прошла успешно"  }  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Авторизация**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/login  **Method:** POST  **Headers:** Content-Type: application/json  **Body:**  {  "login": "log1",  "password": "pass1"  } | Успешная авторизация  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "api\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  }  }  Ошибка аутентификации  **Status:** 401  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 401,      "message": "Ошибка аутентификации"  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Просмотр категорий товаров**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/categories  **Method:** GET  **Headers:**  **Body:**  {  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": [  {  "id": 1,  "name": "Манга",  "created\_at": null,  "updated\_at": null,  },  {  "id": 2,  "name": "Кружки"  "created\_at": null,  "updated\_at": null,  }  // Другие категории  ]  }  Ответ при отсутствии категорий  **Status:** 404  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 404,      "message": "Категории не найдены"  } |
|  |  |

**Просмотр товаров определенной категории**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/category/{id}  **Method:** GET  **Headers:**  **Body:**  {  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {      "data": [          {              "id": 1,              "name": "Берсерк. Том 1.",              "description": "",              "price": "1174.00",              "quantity": 10,              "photo": "uploads/1/1.webp",              "category\_id": 1,              "created\_at": null,              "updated\_at": "2024-03-08T00:00:00.000000Z"          },          {              "id": 2,              "name": "Берсерк. Том 2.",              "description": "",              "price": "1245.00",              "quantity": 10,              "photo": "uploads/1/2.webp",              "category\_id": 1,              "created\_at": null,              "updated\_at": "2024-03-08T00:00:00.000000Z"          },  // Другие товары  ]  }  Ответ при отсутствии товаров  **Status:** 404  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 404,      "message": "Товары не найдены"  } |

**Просмотр конкретного товара**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/product/{id}  **Method:** GET  **Headers:**  **Body:**  {  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {      "data": [          {              "id": 1,              "name": "Берсерк. Том 1.",              "description": "",              "price": "1174.00",              "quantity": 10,              "photo": "uploads/1/1.webp",              "category\_id": 1,              "created\_at": null,              "updated\_at": "2024-03-08T00:00:00.000000Z"          }  ] }  Ответ при отсутствии товаров  **Status:** 404  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 404,      "message": "Товар не найден"  } |
|  |  |

## Функционал авторизированного пользователя

**Просмотр своего профиля**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/profile  **Method:** GET  **Headers:** Authorization: Bearer {{api\_token}}  **Body:**  {  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {      "data": {          "id": 1,          "name": "Иван",          "surname": "Иванов",          "patronymic": "Иванович",          "login": "ivan",          "password": "Ivan123@",          "birth": "1990-01-01",          "email": "ivan@example.com",          "telephone": 1234567890,      }  } |

**Выход**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/logout  **Method:** GET  **Headers:** Authorization: Bearer {{api\_token}}  **Body:**  {  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Вы успешно вышли из системы"  }  } |

**Просмотр своей корзины**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/cart  **Method:** GET  **Headers:** Authorization: Bearer {{api\_token}}  **Body:**  {  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {      "cart\_items": [],      "total": "0.00"  } |

**Добавление товара в корзину**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/product/{id}  **Method:** POST  **Headers:** Authorization: Bearer {{api\_token}},  Content-Type: application/json  **Body:**  {  "product\_id": 1,  "quantity": 2  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Товар успешно добавлен в корзину"  }  }  Ответ при ошибке  **Status:** 404  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 404,      "message": "Товар не найден"  }  Ответ при ошибке добавить количество больше, чем есть  **Status:** 400  **Content-Type:** application/json  **Body:**  {      "error": "Недостаточное количество товара в наличии"  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Оформление заказа**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/checkout  **Method:** POST  **Headers:** Authorization: Bearer {{api\_token}},  Content-Type: application/json  **Body:**  {    "cart\_items": [      {        "product\_id": 1,        "quantity": 2      }    ],    "address": "ул. Новая 5-15",    "payment\_id": 1  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {      "message": "Заказ успешно оформлен"  }  Ответ при ошибке  **Status:** 404  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 404,      "message": "Не удалось оформить заказ, возможно, ваша корзина пуста"  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Оставление отзыва для определённого товара**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/product/{id}/review  **Method:** POST  **Headers:** Authorization: Bearer {{api\_token}},  Content-Type: application/json  **Body:**  {  "rating": 4,  "textReview": "Отличный товар!"  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {      "message": "Отзыв успешно сохранен"  }  Ответ при попытке оставить отзыв на не купленный товар  **Status:** 404  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 404,      "message": "Вы не можете оставить отзыв на товар, который вы не покупали"  }  Ответ при попытке оставить отзыв на уже созданный отзыв по данному товару  **Status:** 403  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 403,      "message": "Вы уже оставили отзыв на этот товар"  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Редактирование корзины**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/cart  **Method:** PATCH  **Headers:** Authorization: Bearer {{api\_token}},  Content-Type: application/json  **Body:**  {  "product\_id": 1,  "quantity": 5  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {      "message": "Количество товара в корзине успешно обновлено"  }  Ответ при попытке ввести отрицательное количество  **Status:** 400  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 400,      "message": "Количество товара должно быть положительным числом"  }  Ответ при попытке ввести количество больше имеющегося  **Status:** 400  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 400,      "message": "Количество товара должно быть положительным числом"  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Редактирование своего профиля**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/profile  **Method:** PATCH  **Headers:** Authorization: Bearer {{api\_token}},  Content-Type: application/json  **Body:**  {  "name": "Новое Имя",  "surname": "Новая Фамилия"  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Профиль успешно обновлен"  }  }  Ошибки валидации полей  **Формат ответа из общих требований** |

## Функционал администратора

**Добавление категории**

Запрос для добавления новой категории в системе. При отправке запроса необходимо передать FormData со следующими полями:

* name – обязательное поле, строка;

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/admin/category/create  **Method:** POST  **Headers:** Authorization: Bearer {{admin\_api\_token}},  Content-Type: application/json  **Body:**  {  "name": "Новая Категория"  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Категория успешно создана"  }  }  Ответ при попытке добавить категорию с уже существующем именем  **Status:** 422  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 422,      "message": "Категория с таким именем уже существует"  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Добавление товара**

Запрос для регистрации нового пользователя в системе. При отправке запроса необходимо передать FormData со следующими полями:

* name – обязательное поле, строка;
* description – необязательное поле, строка;
* price – обязательное поле, положительное вещественное число;
* quantity – обязательное поле, числовой;
* photo – необязательное поле, фото формата jpeg, png, jpg, webp;
* category\_id – обязательное поле, число, категория с данным идентификатором должна быть.

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/admin/product/create  **Method:** POST  **Headers:** Authorization: Bearer {{admin\_api\_token}},  Content-Type: application/json  **Body:**  {    "name": "Новый Товар",    "price": 1500,    "quantity": 10,    "description": "new product",    "category\_id": 1  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Товар успешно создан"  }  }  Ответ при попытке добавить продукт с уже существующем именем  **Status:** 422  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 422,      "message": "Продукт с таким именем уже существует"  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Редактирование категории**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/admin/category/{id}/edit  **Method:** PATCH  **Headers:** Authorization: Bearer {{admin\_api\_token}}, Content-Type: application/json  **Body:**  {    "name": "Измененная манга"  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Категория успешно отредактирована"  }  }  Ответ при ошибке  **Status:** 422  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 422,      "message": "Категория с таким именем уже существует"  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Редактирование товара**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/admin/product/{id}/edit  **Method:** POST  **Headers:** Authorization: Bearer {{admin\_api\_token}}, Content-Type: application/json  **Body:**  {  "name": "Новый Смартфон",  "price": 12000  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Товар успешно отредактирован"  }  }  Ответ при ошибке  **Status:** 400  **Content-Type:** application/json  **Body:**  {  "error": {  "message": "Не удалось отредактировать товар. Пожалуйста, проверьте данные запроса."  }  }  Ответ при попытке изменить продукт с уже существующем именем  **Status:** 422  **Content-Type:** application/json  **Body:**  {      "success": false,      "code": 422,      "message": "Продукт с таким именем уже существует"  }  Ошибки валидации полей  **Формат ответа из общих требований** |

**Удаление категории**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/admin/category/{id}/delete  **Method:** DELETE  **Headers:** Authorization: Bearer {{admin\_api\_token}}  **Body:**  {  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Категория успешно удалена"  }  }  Ответ при ошибке  **Status:** 400  **Content-Type:** application/json  **Body:**  {  "error": {  "message": "Не удалось удалить категорию. Пожалуйста, проверьте данные запроса."  }  } |

**Удаление товара**

|  |  |
| --- | --- |
| Request | Response |
| **URL:** {{host}}/admin/product/{id}/delete  **Method:** DELETE  **Headers:** Authorization: Bearer {{admin\_api\_token}}  **Body:**  {  } | Успешный ответ  **Status:** 200  **Content-Type:** application/json  **Body:**  {  "data": {  "message": "Товар успешно удален"  }  }  Ответ при ошибке  **Status:** 400  **Content-Type:** application/json  **Body:**  {  "error": {  "message": "Не удалось удалить товар. Пожалуйста, проверьте данные запроса."  }  } |

# ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ

## Концептуальная модель данных

Концептуальная модель – это отражение предметной области, для которой разрабатывается база данных. Не вдаваясь в теорию, отметим, что это некая диаграмма с принятыми обозначениями элементов. Все объекты, обозначающие вещи, обозначаются в виде прямоугольника. Атрибуты, характеризующие объект - в виде овала, а связи между объектами - ромбами. Мощность связи обозначаются стрелками (в направлении, где мощность равна многим - двойная стрелка, а со стороны, где она равна единице - одинарная).

Анализируя предметную область, можно выделить следующие сущности с их атрибутами (см. рисунок 1):

1. Сущность «Роль» с атрибутом «Название».
2. Сущность «Пользователь» с атрибутами «Имя», «Фамилия», «Отчество», «Логин», «Пароль», «Дата рождения», «Email», «Телефон», «Токен».
3. Сущность «Отзыв» с атрибутом «Текст», «Оценка».
4. Сущность «Заказ» с атрибутом «Адрес», «Дата».
5. Сущность «Статус» с атрибутом «Название».
6. Сущность «Состав заказа» с атрибутами «Количество», «Общая стоимость».
7. Сущность «Способ оплаты» с атрибутами «Название».
8. Сущность «Товар» с атрибутами «Название», «Описание», «Цена», «Количество», «Фото».
9. Сущность «Категория» с атрибутами «Название».
10. Сущность «Корзина» с атрибутами «Количество», «Общая стоимость».

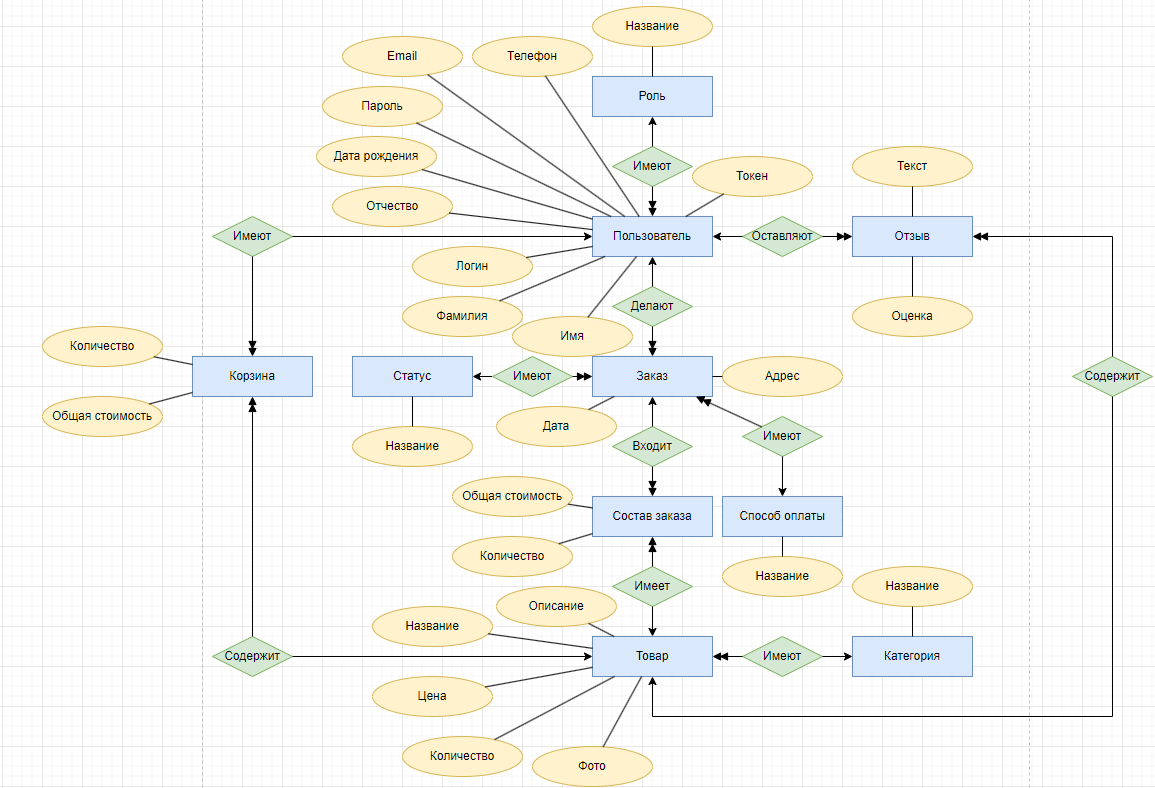


Рисунок 1 – Концептуальная модель данных

## Нормализация данных

Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

Процесс преобразования отношений базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры БД к виду, обеспечивающему минимальную логическую избыточность, и не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение физического объёма базы данных. Конечной целью нормализации является уменьшение потенциальной противоречивости, хранимой в базе данных информации.

Устранение избыточности производится, как правило, за счёт декомпозиции отношений таким образом, чтобы в каждом отношении хранились только первичные факты (то есть факты, не выводимые из других хранимых фактов).

***Первая нормальная форма (1NF)***

Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

***Вторая нормальная форма (2NF)***

Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый не ключевой атрибут неприводимо (функционально полно) зависит от её потенциального ключа. Функционально полная зависимость означает, что если потенциальный ключ является составным, то атрибут зависит от всего ключа и не зависит от его частей.

***Третья нормальная форма (3NF)***

Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости не ключевых атрибутов от ключевых.

При проектировании БД третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

## Логическая модель данных

Логическая модель базы данных — схема базы данных, выраженная в понятиях модели данных. Этим отличается от концептуальной модели, описывающей семантику предметной области без указания технологии (конкретных методов реализации), и от физической модели, которая описывает конкретные физические механизмы, применяемые для хранения данных в накопителях.

Используя нашу концептуальную модель данных и выполнив нормализацию данных до 3 нормальной формы включительно, построим логическую модель данных, в которой мы можем выделить новые атрибуты, поля, являющиеся первичными и внешними ключами, необходимые для построения связей.

Получившиеся логическая модель данных представлена на рисунке 2.

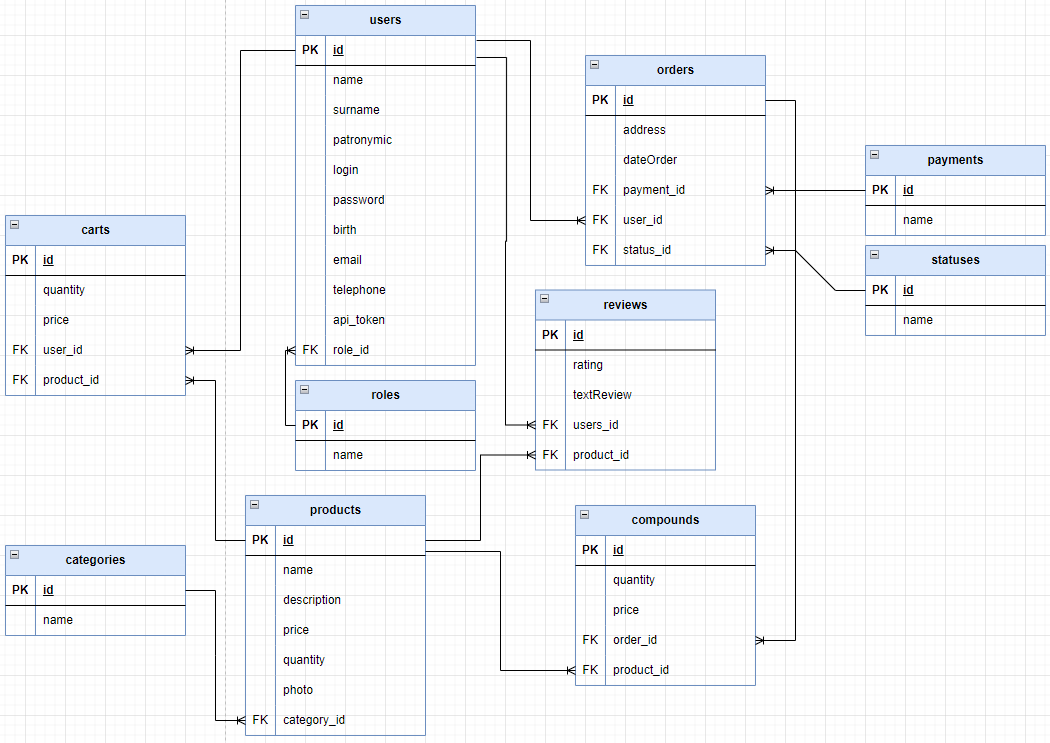


Рисунок 2 - Логическая модель данных

## Физическая модель данных

Физическая модель данных – это модель данных, описанная с помощью средств конкретной системы управления базами данных (СУБД). Физическая модель данных строится на базе даталогической, путем добавления особенностей конкретной СУБД. К таким особенностям могут относиться поддерживаемые СУБД типы данных, соглашения о присвоении имен таблицам, атрибутам и т.д. Физическая модель данных фактически является готовым заданием на создание БД, имея которое можно реализовать БД в выбранной СУБД.

На рисунке 3 представленная физическая модель данных для информационной системы «Информационная система аниме интернет-магазина»

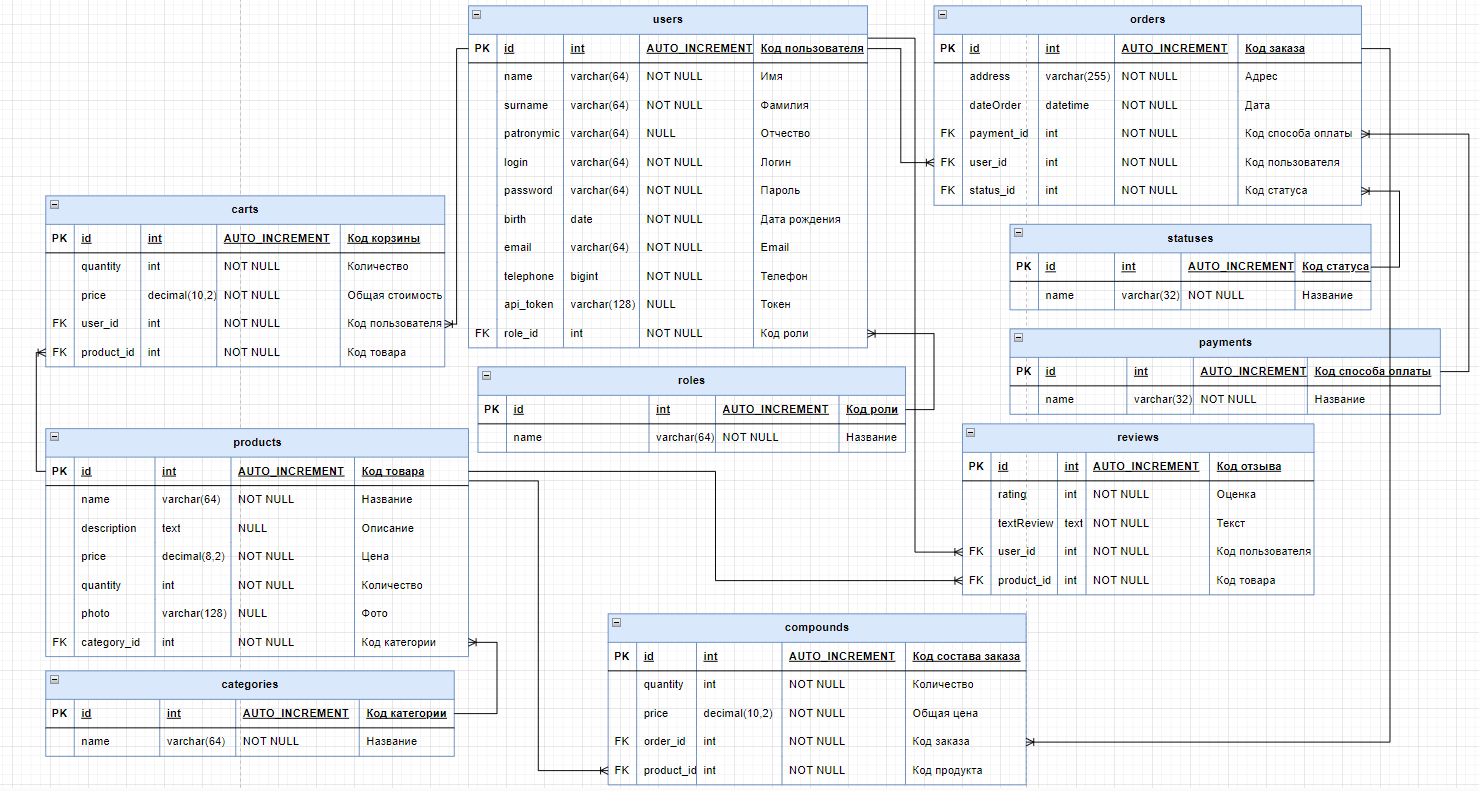


Рисунок 3 - Физическая модель данных

## Словарь данных

Таблица 1 – Словарь данных для БД «Информационная система аниме интернет-магазина»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **roles** | | | | | | |
| **Поле** | **Формат** | | **Обязательное** | | **Ключ** | **Примечание** |
| id | int | | Да | | Первичный | ID роли |
| name | varchar(64) | | Да | |  | Название |
| **users** | | | | | | |
| **Поле** | **Формат** | | **Обязательное** | | **Ключ** | **Примечание** |
| id | int | | Да | | Первичный | ID пользователя |
| name | varchar(64) | | Да | |  | Имя |
| surname | varchar(64) | | Да | |  | Фамилия |
| patronymic | varchar(64) | | Нет | |  | Отчество |
| login | varchar(64) | | Да | |  | Логин |
| password | varchar(64) | | Да | |  | Пароль |
| birth | date | | Да | |  | Дата рождения |
| email | email | | Да | |  | Email |
| telephone | bigint | | Да | |  | Телефон |
| api\_token | int | | Да | |  | Токен |
| role\_id | int | | Да | | Внешний | ID роли |
| **orders** | | | | | | |
| **Поле** | **Формат** | **Обязательное** | | **Ключ** | | **Примечание** |
| id | int | Да | | Первичный | | ID заказа |
| address | varchar(255) | Да | |  | | Адрес |
| dateOrder | date | Да | |  | | Дата |
| discount\_id | int | Да | | Внешний | | ID скидки |
| payment\_id | int | Да | | Внешний | | ID способа оплаты |
| user\_id | int | Да | | Внешний | | ID пользователя |
| status\_id | int | Да | | Внешний | | ID статуса |
| **reviews** | | | | | | |
| **Поле** | **Формат** | | **Обязательное** | | **Ключ** | **Примечание** |
| id | int | | Да | | Первичный | ID отзыва |
| textReview | text | | Да | |  | Текст |
| user\_id | int | | Да | | Внешний | ID пользователя |
| product\_id | int | | Да | | Внешний | ID товара |
| **payments** | | | | | | |
| **Поле** | **Формат** | | **Обязательное** | | **Ключ** | **Примечание** |
| id | int | | Да | | Первичный | ID способа оплаты |
| name | varchar(32) | | Да | |  | Название |
| **compounds** | | | | | | |
| **Поле** | **Формат** | | **Обязательное** | | **Ключ** | **Примечание** |
| id | int | | Да | | Первичный | ID состава заказа |
| quantity | int | | Да | |  | ID количества |
| price | decimal(10, 2) | | Да | |  | ID общей цены |
| order\_id | int | | Да | | Внешний | ID заказа |
| product\_id | int | | Да | | Внешний | ID товара |
| **products** | | | | | | |
| **Поле** | **Формат** | | **Обязательное** | | **Ключ** | **Примечание** |
| id | int | | Да | | Первичный | ID товара |
| name | varchar(64) | | Да | |  | Название |
| description | text | | Нет | |  | Описание |
| price | decimal(8,2) | | Да | |  | Цена |
| quantity | int | | Да | |  | Количество |
| photo | image | | Нет | |  | Фото |
| category\_id | int | | Да | | Внешний | ID категории |
| **categories** | | | | | | |
| **Поле** | **Формат** | | **Обязательное** | | **Ключ** | **Примечание** |
| id | int | | Да | | Первичный | ID категории |
| name | varchar(164) | | Да | |  | Название |
| **carts** | | | | | | |
| **Поле** | **Формат** | | **Обязательное** | | **Ключ** | **Примечание** |
| id | int | | Да | | Первичный | ID корзины |
| quantity | int | | Да | |  | Количество |
| price | decimal(10,2) | | Да | |  | Общая стоимость |
| user\_id | int | | Да | | Внешний | ID пользователя |
| product\_id | int | | Да | | Внешний | ID товара |
| **statuses** | | | | | | |
| **Поле** | **Формат** | | **Обязательное** | | **Ключ** | **Примечание** |
| id | int | | Да | | Первичный | ID статуса |
| name | varchar(32) | | Да | |  | Название |

## Диаграмма базы данных

На рисунке 4 представлена итоговая диаграмма базы данных для интернет-магазина «Anibloom».

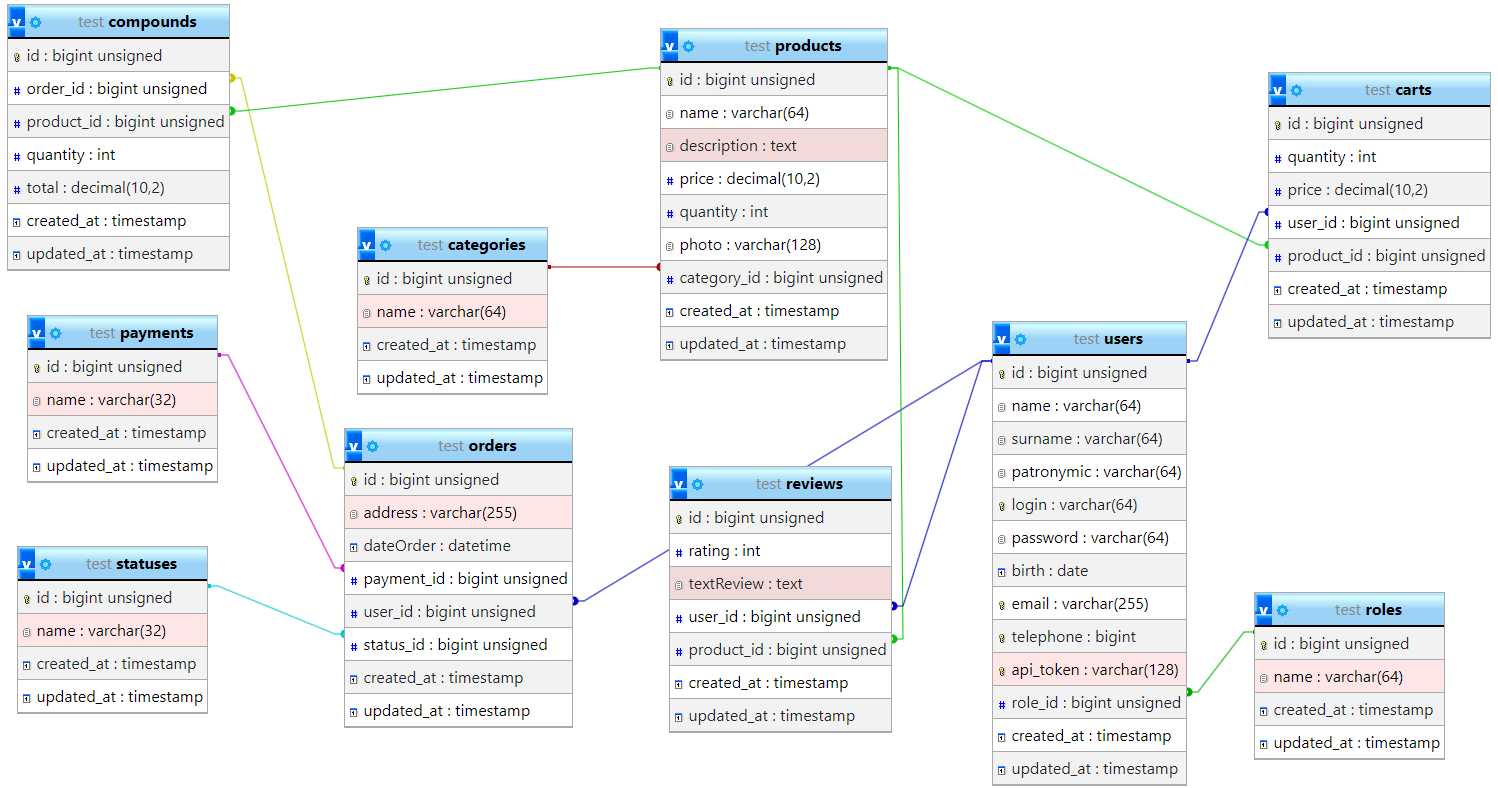


Рисунок 4 - Диаграмма базы данных

# ГЛАВА 3. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ФУНКЦИОНАЛА ИНФОРМАЦИОННОГО ПРОДУКТА

## Установка и настройка фреймворка Laravel

Для установки фреймворка в консоли сервера необходимо перейти в рабочую директорию сайта (используя команду cd) и выполнить команду обновления Composer’а, далее выполнить команду установку фреймворка Laravel.

Так как в данный момент в OsPanel используется версия PHP 8.1, то мы установим Laravel v10.0

|  |
| --- |
| composer self-update  composer create-project laravel/laravel anibloom.com 10.0.\* |

В корневой директории сайта создаем файл .htaccess и прописываем правила перенаправления запросов в точку входа в приложение (в папку public).

RewriteEngine on  
RewriteRule (.\*)? /public/$1

Редактируем файл конфигурации .env, указав данные для подключения к БД MySQL.

DB\_CONNECTION=mysql  
DB\_HOST=127.0.0.1  
DB\_PORT=3306  
DB\_DATABASE=anibloom  
DB\_USERNAME=root  
DB\_PASSWORD=

Редактируем файл конфигурации config\auth.php указав драйвер авторизации по API через систему токенов в разделе Authentication Guards.

'guards' => [  
 ...  
 'api' => [  
 'driver' => 'token',  
 'provider' => 'users',  
],  
],

Удаляем все стандартные миграции в папке database\migrations.

## Создание миграций

Миграции похожи на контроль версий для вашей базы данных, позволяют вашей команде определять схемы базы данных приложения и совместно использовать их определение.

Фасад Schema обеспечивает независимую от базы данных поддержку для создания и управления таблицами во всех поддерживаемых Laravel системах баз данных. В обычной ситуации, этот фасад используется для создания и изменения таблиц / столбцов базы данных во время миграции.

Создаем миграции для таблиц roles, users, carts, categories, products, payments, statuses, orders, compounds, reviews, для этого в корневой директории проекта выполняем следующие команды:

|  |
| --- |
| php artisan make:migration create\_roles\_table  php artisan make:migration create\_users\_table  php artisan make:migration create\_categories\_table  php artisan make:migration create\_products\_table  php artisan make:migration create\_carts\_table  php artisan make:migration create\_payments\_table  php artisan make:migration create\_statuses\_table  php artisan make:migration create\_orders\_table  php artisan make:migration create\_compounds\_table  php artisan make:migration create\_reviews\_table |

Редактируем схему миграции таблицы roles.

|  |
| --- |
| Schema::create('roles', function (Blueprint $table) {  $table->id();  $table->string('name', 64);  $table->timestamps(); }); |

Редактируем схему миграции таблицы users.

|  |
| --- |
| Schema::create('users', function (Blueprint $table) {  $table->id();  $table->string('name', 64);  $table->string('surname', 64);  $table->string('patronymic', 64)->nullable();  $table->string('login', 64)->unique();  $table->string('password', 64);  $table->date('birth');  $table->string('email')->unique();  $table->bigInteger('telephone')->unique();  $table->string('api\_token', 128)->nullable()->unique();  $table->foreignId('role\_id')->constrained('roles', 'id')->onUpdate('cascade');  $table->timestamps(); }); |

Редактируем схему миграции таблицы categories.

|  |
| --- |
| Schema::create('categories', function (Blueprint $table) {  $table->id();  $table->string('name', 64);  $table->timestamps(); }); |

Редактируем схему миграции таблицы products.

|  |
| --- |
| Schema::create('products', function (Blueprint $table) {  $table->id();  $table->string('name', 64);  $table->text('description')->nullable();  $table->decimal('price', 10, 2);  $table->integer('quantity');  $table->string('photo', 128)->nullable();  $table->foreignId('category\_id')->constrained('categories', 'id');  $table->timestamps(); }); |

Редактируем схему миграции таблицы carts.

|  |
| --- |
| Schema::create('carts', function (Blueprint $table) {  $table->id();  $table->integer('quantity');  $table->decimal('price',10, 2);  $table->foreignId('user\_id')->constrained('users', 'id')->onUpdate('cascade');  $table->foreignId('product\_id')->constrained('products', 'id')->onUpdate('cascade');  $table->timestamps(); }); |

Редактируем схему миграции таблицы payments.

|  |
| --- |
| Schema::create('payments', function (Blueprint $table) {  $table->id();  $table->string('name', 32);  $table->timestamps(); }); |

Редактируем схему миграции таблицы statuses.

|  |
| --- |
| Schema::create('statuses', function (Blueprint $table) {  $table->id();  $table->string('name', 32);  $table->timestamps(); }); |

Редактируем схему миграции таблицы orders.

|  |
| --- |
| Schema::create('orders', function (Blueprint $table) {  $table->id();  $table->string('address', 255);  $table->dateTime('dateOrder');  $table->foreignId('payment\_id')->constrained('payments', 'id')->onUpdate('cascade');  $table->foreignId('user\_id')->constrained('users', 'id')->onUpdate('cascade');  $table->foreignId('status\_id')->constrained('statuses', 'id')->onUpdate('cascade');  $table->timestamps(); }); |

Редактируем схему миграции таблицы compounds.

|  |
| --- |
| Schema::create('compounds', function (Blueprint $table) {  $table->id();  $table->foreignId('order\_id')->constrained('orders', 'id')->onUpdate('cascade');  $table->foreignId('product\_id')->constrained('products', 'id')->onUpdate('cascade');  $table->integer('quantity');  $table->decimal('total', 10, 2);  $table->timestamps(); }); |

Редактируем схему миграции таблицы reviews.

|  |
| --- |
| Schema::create('reviews', function (Blueprint $table) {  $table->id();  $table->integer('rating');  $table->text('textReview');  $table->foreignId('user\_id')->constrained('users', 'id')->onUpdate('cascade');  $table->foreignId('product\_id')->constrained('products', 'id')->onUpdate('cascade');  $table->timestamps(); }); |

Выполняем миграцию базы данных с помощью команды:

|  |
| --- |
| php artisan migrate |

Далее заполняем базу данных первоначальными данными.

## Создание класса исключений

Во время выбрасывания исключений Laravel проверяет, есть ли в классе исключения метод render(), если да, то он использует метод этого исключения для отображения результата. Если вы не хотите полагаться на глобальную систему отлова исключений Laravel, то можете вернуть ответ в JSON напрямую из контроллера.

Laravel пытается преобразовать исключения в читаемый формат в зависимости от ожидаемого от клиента формата ответа, будь то HTML или JSON, сначала он преобразует различные форматы исключений в простое исключение типа HttpException. И только затем он обрабатывает некоторые исключения особым образом.

Illuminate\Http\Exceptions\HttpResponseException - это особое исключение, встроенное в Laravel. Особенность этого исключения заключается в том, что оно уже содержит шаблон ответа для клиента, поэтому Laravel просто возвращает ответ из этого исключения.

Создадим особое исключение app\Exceptions\ApiException.php, которое будет определять шаблон выдаваемых пользователю ошибок (будь то ошибка авторизации или ошибка валидации входных данных, либо ошибка доступа к определенному функционалу приложения).

Для этого выполняем следующую команду

|  |
| --- |
| php artisan make:exception ApiException |

Содержимое ApiException

|  |
| --- |
| class ApiException extends Exception {  protected $statusCode;  protected $data;  public function \_\_construct($statusCode, $message = null, $errors = [])  {  $this->statusCode = $statusCode;  $this->data = [  'success' => false,  'code' => $statusCode,  ];  if ($message !== null) {  $this->data['message'] = $message;  }   if (!empty($errors)) {  $this->data['errors'] = $errors;  }  parent::*\_\_construct*();  }  public function render($request)  {  return response()->json($this->data, $this->statusCode);  } } |

## Создание класса посредника

Посредник обеспечивает удобный механизм для проверки и фильтрации HTTP-запросов, поступающих в ваше приложение. Например, в Laravel уже содержится посредник, проверяющий аутентификацию пользователя вашего приложения. Если пользователь не аутентифицирован, то посредник перенаправит пользователя на экран входа в ваше приложение. Однако, если пользователь аутентифицирован, то посредник позволит запросу продолжить работу в приложении.

Посредник может быть написан для выполнения различных задач помимо аутентификации. Например, посредник для ведения журнала может регистрировать все входящие запросы вашего приложения. В состав фреймворка Laravel уже входят несколько посредников, включая посредник для аутентификации и посредник для защиты от CSRF. Все эти посредники находится в каталоге app/Http/Middleware.

Создадим посредник app/Http/Middleware/CheckRole.php, который будет проверять роли пользователей нашего приложения, выполнив следующую команду

|  |
| --- |
| php artisan make:middleware CheckRole |

Содержимое CheckRole

|  |
| --- |
| class CheckRole {  public function handle(Request $request, Closure $next, ...$roles)  {  if (Auth::check() && Auth::user()->hasRole($roles)) {  return $next($request);  }  throw new ApiException(403, 'Forbidden for you');  } } |

Для регистрации посредника в системе его необходимо зарегистрировать в файле app/Http/Kernel.php в секции routeMiddleware.

|  |
| --- |
| protected $routeMiddleware = [  ...  'role' => \App\Http\Middleware\CheckRole::class, ]; |

## Генерация классов моделей

Laravel содержит ORM-библиотеку Eloquent, предоставляющую способ работы с базой данных, который часто удобнее обычного построителя запросов. При использовании Eloquent каждая таблица БД имеет соответствующую «Модель», которая используется для взаимодействия с этой таблицей. Помимо получения записей из таблицы БД, модели Eloquent также позволяют вставлять, обновлять и удалять записи из таблицы.

Модели расширяют класс Illuminate\Database\Eloquent\Model. Чтобы сгенерировать новую модель Eloquent, необходимо использовать Artisan-команду make:model. Эта команда поместит новый класс модели в каталог app/Models вашего приложения.

Создадим модели ко всем таблицам нашей базы данных, кроме User, т. к. автоматически создана, для этого поочередно выполняем следующие команды:

|  |
| --- |
| php artisan make:model Role  php artisan make:model Category  php artisan make:model Product  php artisan make:model Cart  php artisan make:model Payment  php artisan make:model Status  php artisan make:model Order  php artisan make:model Compound  php artisan make:model Review |

Далее отредактируем наши модели указав:

* Массово назначаемые атрибуты (переменная $fillable).
* Отношения.
* Дополнительные методы для получения и/или изменения данных в зависимости от технического задания.

В конечном итоге наши модели будут имеет следующий вид:

* Модель Role

|  |
| --- |
| class Role extends Model {  use HasFactory;  protected $fillable = ['name'];  *//Отношение с таблицей users* public function users()  {  return $this->hasMany(User::class);  } } |

* Модель User

|  |
| --- |
| class User extends *Authenticatable* {  use HasApiTokens, HasFactory, Notifiable;   */\*\*  \* The attributes that are mass assignable.  \*  \* @var array<int, string>  \*/* protected $fillable = [  'name', 'surname', 'patronymic', 'login', 'password', 'birth', 'email', 'telephone', 'api\_token', 'role\_id'  ];  *//Отношение с таблицей roles* public function role()  {  return $this->belongsTo(Role::class);  }  *//Отношение с таблицей orders* public function order()  {  return $this->hasMany(Order::class);  }  *//Отношение с таблицей carts* public function cart()  {  return $this->hasMany(Cart::class);  }  *//Метод проверки ролей из списка ролей* public function hasRole(array $role) {  return in\_array($this->role->id, $role);  } } |

* Модель Category

|  |
| --- |
| class Category extends Model {  use HasFactory;  protected $fillable = ['name'];  *//Отношение с таблицей products* public function products()  {  return $this->hasMany(Product::class);  } } |

* Модель Product

|  |
| --- |
| class Product extends Model {  use HasFactory;  protected $fillable = ['name', 'description', 'price', 'quantity', 'photo', 'category\_id'];  *//Отношение с таблицей categories* public function category()  {  return $this->belongsTo(Category::class);  }  *//Отношение с таблицей carts* public function cart()  {  return $this->hasMany(Cart::class);  }  *//Отношение с таблицей reviews* public function review()  {  return $this->hasMany(Review::class);  }  *//Отношение с таблицей orders* public function order()  {  return $this->belongsToMany(Order::class, 'compound');  } } |

* Модель Cart

|  |
| --- |
| class Cart extends Model {  use HasFactory;  protected $fillable = ['quantity', 'price','user\_id', 'product\_id'];  *//Отношение с таблицей users* public function user()  {  return $this->belongsTo(User::class);  }  *//Отношение с таблицей products* public function product()  {  return $this->belongsTo(Product::class);  } } |

* Модель Payment

|  |
| --- |
| class Payment extends Model {  use HasFactory;  protected $fillable = ['name'];  *//Отношение с таблицей orders* public function orders()  {  return $this->hasMany(Order::class);  } } |

* Модель Status

|  |
| --- |
| class Status extends Model {  use HasFactory;  protected $fillable = ['name'];  *//Отношение с таблицей orders* public function orders()  {  return $this->hasMany(Order::class);  } } |

* Модель Order

|  |
| --- |
| class Order extends Model {  use HasFactory;  protected $fillable = ['address', 'dateOrder', 'payment\_id', 'user\_id', 'status\_id'];  *//Отношение с таблицей users* public function user()  {  return $this->belongsTo(User::class);  }  *//Отношение с таблицей payments* public function payment()  {  return $this->belongsTo(Payment::class);  }  *//Отношение с таблицей statuses* public function status()  {  return $this->belongsTo(Status::class);  }  *//Отношение с таблицей products* public function products()  {  return $this->belongsToMany(Product::class)->withPivot('quantity');  }  *//Отношение с таблицей compounds* public function compound()  {  return $this->hasMany(Compound::class);  } } |

* Модель Compound

|  |
| --- |
| class Compound extends Model {  use HasFactory;  protected $fillable = ['order\_id', 'product\_id', 'quantity', 'total'];  *//Отношение с таблицей compounds* public function order()  {  return $this->belongsTo(Order::class);  }  *//Отношение с таблицей products* public function product()  {  return $this->belongsTo(Product::class);  } } |

* Модель Review

|  |
| --- |
| class Review extends Model {  use HasFactory;  protected $fillable = ['rating', 'textReview', 'user\_id', 'product\_id'];  *//Отношение с таблицей users* public function user()  {  return $this->belongsTo(User::class);  }  *//Отношение с таблицей products* public function product()  {  return $this->belongsTo(Product::class);  } } |

## Валидация запроса формы

Form Request — это класс, где мы описываем правила валидации для входящих данных. Обычно класс содержит набор правил под запрос из клиента. Мы можем его декларировать в контроллере, и через контейнер в Laravel он автоматически проверит данные на соответствии нашим правилам и через внутренние механизмы фреймворка выдаст ответ клиенту.

Для более сложных сценариев валидации мы создадим «запрос формы». Запрос формы – это класс запроса, который инкапсулирует свою собственную логику валидации и авторизации. Сгенерируем родительский запрос формы ApiRequest c методами непрошедшей авторизации и непрошедшей валидации.

|  |
| --- |
| php artisan make:request ApiRequest |

Отредактируем ApiRequest

|  |
| --- |
| class ApiRequest extends FormRequest {  public function failedValidation(Validator $validator)  {  throw new ApiException(422, '', $validator->errors());  }  public function failedAuthorization()  {  return throw new ApiException(403, 'Login failed');  } } |

Далее необходимо на основе родительского класса ApiRequest сгенерировать запрос формы для каждого запроса, создадим сразу все необходимые:

|  |
| --- |
| php artisan make:request CategoryCreateRequest  php artisan make:request CategoryUpdateRequest  php artisan make:request LoginRequest  php artisan make:request OrderCreateRequest  php artisan make:request ProductCreateRequest  php artisan make:request ProductUpdateRequest  php artisan make:request ReviewCreateRequest  php artisan make:request UserCreateRequest  php artisan make:request UserUpdateRequest |

* Класс валидации запроса формы CategoryCreateRequest – создание категории.

|  |
| --- |
| class CategoryCreateRequest extends ApiRequest {  */\*\*  \* Determine if the user is authorized to make this request.  \*  \* @return bool  \*/* public function authorize()  {  return true;  }   */\*\*  \* Get the validation rules that apply to the request.  \*  \* @return array<string, mixed>  \*/* public function rules()  {  return [  'name' => 'required|string|max:64'  ];  }  public function messages()  {  return [  'name.required' => 'Поле не может быть пустым.',  'name.max' => 'Поле не может содержать более :max символов.',  ];  } } |

* Класс валидации запроса формы CategoryUpdateRequest – обновление категории.

|  |
| --- |
| class CategoryUpdateRequest extends ApiRequest {  */\*\*  \* Determine if the user is authorized to make this request.  \*  \* @return bool  \*/* public function authorize()  {  return true;  }   */\*\*  \* Get the validation rules that apply to the request.  \*  \* @return array<string, mixed>  \*/* public function rules()  {  return [  'name' => 'string|max:64'  ];  }  public function messages()  {  return [  'name.max' => 'Поле не может содержать более :max символов.',  ];  } } |

* Класс валидации запроса формы LoginRequest – авторизация.

|  |
| --- |
| class LoginRequest extends ApiRequest {  */\*\*  \* Determine if the user is authorized to make this request.  \*  \* @return bool  \*/* public function authorize()  {  return true;  }   */\*\*  \* Get the validation rules that apply to the request.  \*  \* @return array<string, mixed>  \*/* public function rules(): array  {  return [  'login' => 'required|string|max:64',  'password' => 'required|string|max:64',  ];  }  public function messages()  {  return [  'login.required' => 'Поле не может быть пустым.',  'login.max' => 'Поле не может содержать более :max символов.',  'password.required' => 'Поле не может быть пустым.',  'password.max' => 'Поле не может содержать более :max символов.',  ];  } } |

* Класс валидации запроса формы OrderCreateRequest – создание заказа.

|  |
| --- |
| class OrderCreateRequest extends ApiRequest {  */\*\*  \* Determine if the user is authorized to make this request.  \*  \* @return bool  \*/* public function authorize()  {  return true;  }   */\*\*  \* Get the validation rules that apply to the request.  \*  \* @return array<string, mixed>  \*/* public function rules()  {  return [  'address' => 'required|string|max:255',  'payment\_id' => 'required|integer|min:1|max:20'  ];  }  public function messages()  {  return [  'address.required' => 'Поле обязательно для заполнения.',  'address.max' => 'Поле должно содержать не более :max символов.',  'payment\_id.required' => 'Поле обязательно для заполнения.',  'payment\_id.integer' => 'Поле должно быть целым числом.',  'payment\_id.min' => 'Поле должно быть не менее :min.',  'payment\_id.max' => 'Поле должно быть не более :max.',  ];  } } |

* Класс валидации запроса формы ProductCreateRequest – создание товара.

|  |
| --- |
| class ProductCreateRequest extends ApiRequest {  */\*\*  \* Determine if the user is authorized to make this request.  \*  \* @return bool  \*/* public function authorize()  {  return true;  }   */\*\*  \* Get the validation rules that apply to the request.  \*  \* @return array<string, mixed>  \*/* public function rules()  {  return [  'name' => 'required|string|max:64',  'description' => 'nullable|string',  'price' => ['required', 'numeric', 'min:0', 'regex:/^\d{1,8}(\.\d{1,2})?$/'], *// Формат decimal(10,2)* 'quantity' => 'required|integer|min:1',  'photo' => 'nullable|file|mimes:jpeg,jpg,png,webp|max:4096', *//до 4мб* 'category\_id' => 'required|integer|min:1',  ];  }  public function messages()  {  return [  'name.required' => 'Поле обязательно для заполнения.',  'name.max' => 'Поле должно содержать не более :max символов.',  'price.required' => 'Поле обязательно для заполнения.',  'price.numeric' => 'Поле должно быть числом.',  'price.min' => 'Поле должно быть не менее :min.',  'price.regex' => 'Недопустимый формат поля.',  'quantity.required' => 'Поле обязательно для заполнения.',  'quantity.integer' => 'Поле должно быть целым числом.',  'quantity.min' => 'Поле должно быть не менее :min.',  'photo.file' => 'Поле должно быть файлом.',  'photo.mimes' => 'Поле должно быть файлом типа: jpeg, jpg, png, webp.',  'photo.max' => 'Файл в поле должен быть не больше :max килобайт.',  'category\_id.required'=> 'Поле обязательно для заполнения.',  'category\_id.integer' => 'Поле должно быть целым числом.',  'category\_id.min' => 'Поле должно быть не менее :min.',  ];  } } |

* Класс валидации запроса формы ProductUpdateRequest – обновление товара.

|  |
| --- |
| class ProductUpdateRequest extends ApiRequest {  */\*\*  \* Determine if the user is authorized to make this request.  \*  \* @return bool  \*/* public function authorize()  {  return true;  }   */\*\*  \* Get the validation rules that apply to the request.  \*  \* @return array<string, mixed>  \*/* public function rules()  {  return [  'name' => 'string|max:64',  'description' => 'nullable|string',  'price' => ['numeric', 'min:0', 'regex:/^\d{1,8}(\.\d{1,2})?$/'], *// Формат decimal(10,2)* 'quantity' => 'integer|min:1',  'photo' => 'nullable|file|mimes:jpeg,jpg,png,webp|max:4096', *//до 4мб* 'category\_id' => 'integer|min:1',  ];  }  public function messages()  {  return [  'name.max' => 'Поле должно содержать не более :max символов.',  'price.numeric' => 'Поле должно быть числом.',  'price.min' => 'Поле должно быть не менее :min.',  'price.regex' => 'Недопустимый формат поля.',  'quantity.integer' => 'Поле должно быть целым числом.',  'quantity.min' => 'Поле должно быть не менее :min.',  'photo.file' => 'Поле должно быть файлом.',  'photo.mimes' => 'Поле должно быть файлом типа: jpeg, jpg, png, webp.',  'photo.max' => 'Файл в поле должен быть не больше :max килобайт.',  'category\_id.integer' => 'Поле должно быть целым числом.',  'category\_id.min' => 'Поле должно быть не менее :min.',  ];  } } |

* Класс валидации запроса формы ReviewCreateRequest – создание отзыва.

|  |
| --- |
| class ReviewCreateRequest extends ApiRequest {  */\*\*  \* Determine if the user is authorized to make this request.  \*  \* @return bool  \*/* public function authorize()  {  return true;  }   */\*\*  \* Get the validation rules that apply to the request.  \*  \* @return array<string, mixed>  \*/* public function rules()  {  return [  'rating' => 'required|integer|min:1|max:5',  'textReview' => 'required|string',  ];  }  public function messages()  {  return [  'rating.required' => 'Поле обязательно для заполнения.',  'rating.integer' => 'Поле должно быть целым числом.',  'rating.min' => 'Поле должно быть не менее :min.',  'rating.max' => 'Поле должно быть не более :max.',  'textReview.required' => 'Поле обязательно для заполнения.',  ];  } } |

* Класс валидации запроса формы UserCreateRequest – регистрация.

|  |
| --- |
| class UserCreateRequest extends ApiRequest {  */\*\*  \* Determine if the user is authorized to make this request.  \*/* public function authorize(): bool  {  return true;  }   */\*\*  \* Get the validation rules that apply to the request.  \*  \* @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>  \*/* public function rules(): array  {  return [  'name' => 'required|string|max:64',  'surname' => 'required|string|max:64',  'patronymic' => 'nullable|string|max:64',  'login' => 'required|string|min:5|max:64|unique:users',  'password' => 'required|string|min:8|max:64|regex:/^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]+$/',  'birth' => 'required|date|date\_format:Y-m-d|before\_or\_equal:2010-01-01|after\_or\_equal:1930-01-01',  'email' => 'required|email|max:64|unique:users',  'telephone' => 'required|integer|digits\_between:1,20|unique:users',  ];  }  public function messages()  {  return [  'name.required' => 'Поле обязательно для заполнения.',  'name.max' => 'Поле должно содержать не более :max символов.',   'surname.required' => 'Поле обязательно для заполнения.',  'surname.max' => 'Поле должно содержать не более :max символов.',   'patronymic.max' => 'Поле должно содержать не более :max символов.',   'login.required' => 'Полобязательно для заполнения.',  'login.min' => 'Поле должно содержать не менее :min символов.',  'login.max' => 'Поле должно содержать не более :max символов.',  'login.unique' => 'Такое уже существует.',   'password.required' => 'Поле обязательно для заполнения.',  'password.min' => 'Поле должно содержать не менее :min символов.',  'password.max' => 'Поле должно содержать не более :max символов.',  'password.regex' => 'Пароль должен содержать как минимум одну цифру, одну заглавную букву, одну малую букву и один специальный символ.',   'birth.required' => 'Поле обязательно для заполнения.',  'birth.date' => 'Поле должно быть датой.',  'birth.date\_format' => 'Неверный формат поля. Используйте формат ГГГГ-ММ-ДД.',  'birth.before\_or\_equal' => 'Дата рождения должна быть до или равна :date.',   'email.required' => 'Поле обязательно для заполнения.',  'email.email' => 'Поле должно быть действительным адресом электронной почты.',  'email.max' => 'Поле должно содержать не более :max символов.',  'email.unique' => 'Такое уже существует.',   'telephone.required' => 'Поле обязательно для заполнения.',  'telephone.integer' => 'Поле должно быть целым числом.',  'telephone.digits\_between' => 'Поле должно содержать от :min до :max цифр.',  'telephone.unique' => 'Такое уже существует.',  ];  }  } |

* Класс валидации запроса формы UserUpdateRequest – редактирование профиля.

|  |
| --- |
| class UserUpdateRequest extends ApiRequest {  */\*\*  \* Determine if the user is authorized to make this request.  \*  \* @return bool  \*/* public function authorize()  {  return true;  }   */\*\*  \* Get the validation rules that apply to the request.  \*  \* @return array<string, mixed>  \*/* public function rules()  {  $user = auth()->user();   return [  'name' => 'string|max:64',  'surname' => 'string|max:64',  'patronymic' => 'nullable|string|max:64',  'login' => 'string|min:5|max:64|unique:users,login,'.$user->id,  'password' => 'string|min:8|max:64|regex:/^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]+$/',  'birth' => 'date|date\_format:Y-m-d|before\_or\_equal:2010-01-01',  'email' => 'email|max:64|unique:users,email,'.$user->id,  'telephone' => 'integer|digits\_between:1,20|unique:users,telephone,'.$user->id,  ];  }  public function messages()  {  return [  'name.max' => 'Поле должно содержать не более :max символов.',   'surname.max' => 'Поле должно содержать не более :max символов.',   'patronymic.max' => 'Поле должно содержать не более :max символов.',   'login.min' => 'Поле должно содержать не менее :min символов.',  'login.max' => 'Поле должно содержать не более :max символов.',  'login.unique' => 'Такое уже существует.',   'password.min' => 'Поле должно содержать не менее :min символов.',  'password.max' => 'Поле должно содержать не более :max символов.',  'password.regex' => 'Пароль должен содержать как минимум одну цифру, одну заглавную букву, одну малую букву и один специальный символ.',   'birth.date' => 'Поле должно быть датой.',  'birth.date\_format' => 'Неверный формат поля. Используйте формат ГГГГ-ММ-ДД.',  'birth.before\_or\_equal' => 'Дата рождения должна быть до или равна :date.',   'email.email' => 'Поле должно быть действительным адресом электронной почты.',  'email.max' => 'Поле должно содержать не более :max символов.',  'email.unique' => 'Такае уже существует.',   'telephone.integer' => 'Поле должно быть целым числом.',  'telephone.digits\_between' => 'Поле должно содержать от :min до :max цифр.',  'telephone.unique' => 'Такое уже существует.',  ];  }  } |

## Генерация классов контроллера

Вместо того чтобы определять всю логику обработки запросов как замыкания в файлах маршрутов, мы организуем это поведение с помощью классов «контроллеров». Контроллеры могут сгруппировать связанную логику обработки запросов в один класс.

Выполним генерацию восьми контроллеров, необходимых для решения технического задания, для этого выполним следующие команды:

|  |
| --- |
| php artisan make:controller AuthController  php artisan make:controller AdminController  php artisan make:controller CartController  php artisan make:controller CategoryController  php artisan make:controller OrderController  php artisan make:controller ProductController  php artisan make:controller ReviewController  php artisan make:controller UserController |

Далее отредактируем наши котроллеры написав необходимые методы:

* Контроллер AuthController

|  |
| --- |
| class AuthController extends Controller {  *//Метод авторизации* public function login(LoginRequest $request) {  $user = User  ::*where*('login', $request->login)  ->where('password', $request->password)  ->first();   if (!$user) throw new ApiException(401, 'Ошибка аутентификации');   $newToken = Hash::make(microtime(true) \* 1000);   $user->api\_token = $newToken;  $user->save();   return response()->json([  'data' => [  'api\_token' => $user->api\_token,  ],  ]);  }  *//Метод выхода из аккаунта* public function logout(Request $request) {  $user = $request->user();  if (!$user) throw new ApiException(401, 'Ошибка аутентификации');  $user->api\_token = null;  $user->save();  return response([  'data' => [  'message' => 'Вы успешно вышли из системы',  ],  ]);  } } |

* Контроллер AdminController

|  |
| --- |
| class AdminController extends Controller {  *//Метод создания категории* public function createCategory(CategoryCreateRequest $request)  {  *// Проверяем, есть ли категория с таким именем уже в базе данных* $existingCategory = Category::*where*('name', $request->input('name'))->first();  if ($existingCategory) {  throw new ApiException(422, 'Категория с таким именем уже существует');  }  *// Создаем новую категорию* $category = new Category([  'name' => $request->input('name'),  ]);  *// Сохраняем категорию в базе данных* $category->save();  *// Возвращаем успешный ответ* return response()->json(['message' => 'Категория успешно создана'], 201);  }  *//Метод создания товара* public function createProduct(ProductCreateRequest $request)  {  *// Проверяем, есть ли продукт с таким именем уже в базе данных* $existingProduct = Product::*where*('name', $request->input('name'))->first();  if ($existingProduct) {  throw new ApiException(422, 'Продукт с таким именем уже существует');  }  *// Создаем новый продукт* $product = new Product($request->all());  *// Сохраняем продукт в базе данных* $product->save();  *// Получаем ID только что созданного продукта* $productId = $product->id;  *// Проверяем, загружен ли файл* if ($request->hasFile('photo')) {  *// Получаем файл из запроса* $file = $request->file('photo');  *// Определяем путь для сохранения файла* $filePath = 'uploads/' . $request->input('category\_id');  *// Переименовываем файл* $fileName = $productId . '.' . $file->getClientOriginalExtension(); *// Получение расширения оригинального файла  // Сохраняем файл на сервере* $filePathToPlace = $file->storeAs($filePath, $fileName);  *// Проверяем успешность сохранения файла* if ($fileName) {  *// Файл успешно сохранен, продолжаем сохранение продукта с указанием имени файла* $product->photo = $filePathToPlace; *// Сохраняем путь до файла* Storage::putFileAs('public/' . $filePath, $file, $fileName);  $product->save();  return response()->json(['message' => 'Продукт успешно создан'], 201);  } else {  *// Если возникла ошибка при сохранении файла* throw new ApiException(500, 'Ошибка сохранить файл');  }  } else {  *// Если файл не был загружен, просто возвращаем ответ об успешном создании продукта* return response()->json(['message' => 'Продукт успешно создан'], 201);  }  }  *//Метод обновления категории* public function updateCategory(CategoryUpdateRequest $request, $id)  {  *//Проверка существования* $category = Category::*find*($id);  if (!$category) {  throw new ApiException(404, 'Категория не найдена');  }  *// Проверяем, есть ли категория с таким именем уже в базе данных* $existingCategory = Category::*where*('name', $request->input('name'))->first();  if ($existingCategory) {  throw new ApiException(422, 'Категория с таким именем уже существует');  }  $category->name = $request->input('name');  $category->save();  return response()->json(['message' => 'Категория успешно обновлена'], 200);  }  *//Метод обновления товара* public function updateProduct(ProductUpdateRequest $request, $id)  {  *//Проверка существования* $product = Product::*find*($id);  if (!$product) {  throw new ApiException(404, 'Товар не найден');  }  *// Проверяем, есть ли продукт с таким именем уже в базе данных* $existingProduct = Product::*where*('name', $request->input('name'))->first();  if ($existingProduct) {  throw new ApiException(422, 'Продукт с таким именем уже существует');  }  *// Обновление изображения товара, если новое изображение предоставлено  // Проверяем, загружен ли файл* if ($request->hasFile('photo')) {  *// Получаем файл из запроса* $file = $request->file('photo');  *// Получаем текущую категорию или новую* $category\_id = $request->input('category\_id') ? $request->input('category\_id') : $product->category\_id;  *// Определяем путь для сохранения файла* $filePath = 'uploads/' . $category\_id;  *//переименовываем файл* $fileName = $product->id . '.' . $file->getClientOriginalExtension(); *// Получение расширения оригинального файла  //удаление файла на сервере* if($product->photo != NULL)Storage::delete($product->photo);  *// Сохраняем файл на сервере* Storage::putFileAs('public/' . $filePath, $file, $fileName);  $filePathToPlace = $file->storeAs($filePath, $fileName);  $product->photo = $filePathToPlace; *// Сохраняем путь до файла* }  *// Сохранение остальных данных товара* $product->fill($request->except('photo')); *// Обновляем все остальные поля товара, кроме изображения  // Сохранение изменений* $product->save();  return response()->json(['message' => 'Товар успешно обновлен'], 200);  }  *//Метод удаления категории* public function deleteCategory($id)  {  $category = Category::*find*($id);   if (!$category) {  throw new ApiException(404, 'Категория не найдена');  }   $category->delete();   return response()->json(['message' => 'Категория успешно удалена'], 200);  }  *//Метод удаления товара* public function deleteProduct($id)  {  $product = Product::*find*($id);   if (!$product) {  throw new ApiException(404, 'Продукт не найден');   }  $product->delete();  return response()->json(['message' => 'Продукт успешно удален'], 200);  } } |

* Контроллер CartController

|  |
| --- |
| class CartController extends Controller {  *//Метод просмотра корзины текущего пользователя* public function index()  {  *// Получение текущего пользователя* $user = auth()->user();  *// Получаем корзину текущего пользователя* $cartItems = $user->cart;  *// Вычисляем общую стоимость всех товаров в корзине* $totalPrice = 0;  foreach ($cartItems as $item) {  $totalPrice += $item->product->price \* $item->quantity;  }  *// Форматирование общей цены* $formattedTotalPrice = number\_format($totalPrice, 2, '.', '');  *// Возвращаем JSON-ответ с содержимым корзины и общей стоимостью* return response()->json([  'cart\_items' => $cartItems,  'total' => $formattedTotalPrice  ]);  }  *//Метод редактирования корзины* public function update(Request $request)  {  *// Получаем текущего пользователя* $user = auth()->user();  *// Получаем product\_id и новое количество товара из запроса* $productId = $request->input('product\_id');  $newQuantity = $request->input('quantity');  *// Находим товар в корзине текущего пользователя по product\_id* $cartItem = Cart::*where*('user\_id', $user->id)  ->where('product\_id', $productId)  ->first();  *// Проверка на существование товара* $product = Product::*find*($productId);   *// Проверяем, найден ли товар в корзине* if (!$cartItem) {  throw new ApiException(404, 'Товар не найден в вашей корзине');  }  *// Получение доступного количества товара из базы данных* $availableQuantity = $product->quantity;  *// Проверяем, существует ли уже запись для этого товара в корзине пользователя* $existingCartItem = $user->cart()->where('product\_id', $product->id)->first();  *// Проверяем, чтобы новое количество было положительным числом и не было больше того, что в хранится в БД* if ($newQuantity > 0 && $newQuantity <= $availableQuantity) {  *// Обновляем количество товара* $existingCartItem->quantity = $newQuantity;  $existingCartItem->price = $product->price \* $newQuantity;  $existingCartItem->save();  *// Возвращаем ответ с сообщением об успешном обновлении корзины* return response()->json(['message' => 'Количество товара в корзине успешно обновлено'], 200);  } else {  throw new ApiException(400, 'Количество товара должно быть положительным числом');  }  }  *//Метод добавления товара в корзину* public function addToCart(Request $request, $id) {  *// Получение текущего пользователя* $user = auth()->user();  *// Проверка существует ли пользователь* if (!$user) {  return response()->json(['error' => 'Пользователь не авторизирован'], 401);  }  $product = Product::*find*($id);  *// Проверка на существование товара* if(!$product) {  throw new ApiException(404, 'Товар не найден');  }  *// Получение доступного количества товара из базы данных* $availableQuantity = $product->quantity;  *// Получение количества товара из запроса (добавляем значение по умолчанию = 1)* $quantity = $request->input('quantity', 1);  *// Проверка, что количество товара больше 0* if ($quantity <= 0) {  throw new ApiException(400, 'Количество товара должно быть больше 0');  }  *// Проверка, что запрошенное количество товара не превышает доступное количество* if ($quantity > $availableQuantity) {  throw new ApiException(400, 'Недостаточное количество товара в наличии');   }  *// Проверяем, существует ли уже запись для этого товара в корзине пользователя* $existingCartItem = $user->cart()->where('product\_id', $product->id)->first();  if ($existingCartItem) {  *// Если запись уже существует, обновляем количество товара* $existingCartItem->quantity = $quantity;  $existingCartItem->price = $product->price \* $quantity;  $existingCartItem->save();  } else {  *// Создание нового элемента корзины и связывание с пользователем* $cartItem = new Cart([  'quantity' => $quantity,  'price' => $product->price \* $quantity,  'product\_id' => $product->id,  ]);  $user->cart()->save($cartItem);  }  return response()->json(['message' => 'Продукт добавлен в корзину']);  }  *//Метод удаления товара из корзины* public function delete(Request $request, int $id)  {  *// Получаем текущего пользователя* $user = auth()->user();  *// Находим товар в корзине текущего пользователя по id* $cartItem = Cart::*where*('user\_id', $user->id)  ->where('product\_id', $id)  ->first();  *// Проверяем, найден ли товар в корзине* if (!$cartItem) {  throw new ApiException(404, 'Товар не найден в вашей корзине');  }  *// Удаляем товар из корзины* $cartItem->delete();  *// Возвращаем ответ с сообщением об успешном удалении товара из корзины* return response()->json(['message' => 'Товар успешно удален из корзины'], 200);  } } |

* Контроллер CategoryController

|  |
| --- |
| class CategoryController extends Controller {  *//Метод просмотра категорий* public function index() {  $categories = Category::*all*();  if(!$categories) {  throw new ApiException(404, 'Категории не найдены');  } else {  return response([  'data' => $categories,  ]);  }  } } |

* Контроллер OrderController

|  |
| --- |
| class OrderController extends Controller {  *//Метод оформления заказа* public function checkout(OrderCreateRequest $request) {  *// Получаем текущее время* $currentDateTime = Carbon::*now*()->format('Y-m-d H:i:s');  *// Получаем данные пользователя* $user = Auth::user();  $address = $request->input('address');  $paymentId = $request->input('payment\_id');  *// Создаем заказ* $order = new Order([  'address' => $address,  'dateOrder' => $currentDateTime,  'payment\_id' => $paymentId,  'user\_id' => $user->id,  'status\_id' => 1, *//Ставим статус "В ожидании"* ]);  $order->save();  *// Получаем товары из корзины пользователя* $cartItems = Cart::*where*('user\_id', $user->id)->get();  *//Если товаров нет, то выводим сообщение об ошибке* if($cartItems->isEmpty()) {  throw new ApiException(404, 'Не удалось оформить заказ, возможно, ваша корзина пуста');  }  foreach ($cartItems as $cartItem) {  *// Находим товар по его ID* $product = Product::*find*($cartItem->product\_id);  if (!$product) {  *// Если товар не найден, пропускаем его* continue;  }  *// Проверяем, достаточно ли товара для оформления заказа* if ($product->quantity < $cartItem->quantity) {  return response()->json(['error' => 'Недостаточное количество товара в наличии'], 400);  }  *// Уменьшаем количество товара в таблице Product* $product->quantity -= $cartItem->quantity;  $product->save();  *// Создаем состав заказа (Compound)* $compound = new Compound([  'order\_id' => $order->id,  'product\_id' => $cartItem->product\_id,  'quantity' => $cartItem->quantity,  'total' => $cartItem->price  ]);  $compound->save();  *// Удаляем товар из корзины пользователя* $cartItem->delete();  }  *// Возвращаем ответ с сообщением об успешном оформлении заказа* return response()->json(['message' => 'Заказ успешно оформлен'], 200);  }  *//Метод получения способов оплаты* public function payment()  {  $payment = Payment::*all*();  return response()->json([  'data' => $payment  ]);  }  *//Метод для просмотра всех заказов текущего пользователя* public function index()  {  *// Получаем данные пользователя* $user = Auth::user();  $orders = Order::*where*('user\_id', $user->id)->get();  *// Добавляем связанные модели для payment и status* $orders->each(function ($order) {  $order->payment\_name = $order->payment->name;  $order->status\_name = $order->status->name;  });  return response()->json(['data' => $orders]);  } } |

* Контроллер ProductController

|  |
| --- |
| class ProductController extends Controller {  *//Метод просмотра товаров конкретной категории* public function showMany(int $id) {  $products = Product::*where*('category\_id', $id)->get();  if($products->isEmpty()) {  throw new ApiException(404, 'Товары не найдены');  }  return response([  'data' => $products  ]);  }  *//Метод просмотра всех товаров* public function index() {  $products = Product::*all*();  if($products->isEmpty()) {  throw new ApiException(404, 'Товары не найдены');  } else {  return response([  'data' => $products,  ]);  }  }  *//Метод просмотра конкретного товара* public function show(int $id) {  $product = Product::*where*('id', $id)->first();  if(!$product) throw new ApiException(404, 'Товар не найден');  return response([  'data' => $product  ]);  } } |

* Контроллер ReviewController

|  |
| --- |
| class ReviewController extends Controller {  *//Метод просмотра отзывов конкретного товара* public function index($productId)  {  *// Получаем отзывы для указанного товара с данными пользователя* $reviews = Review::*where*('product\_id', $productId)  ->with('user')  ->get();  *// Проверяем, найдены ли отзывы* if ($reviews->isEmpty()) {  *// Если отзывы не найдены, возвращаем сообщение об ошибке* throw new ApiException(404, 'Отзывы не найдены для указанного товара');  }  *// Возвращаем отзывы в формате JSON* return response()->json($reviews);  }  *//Метод создания нового отзыва* public function store(ReviewCreateRequest $request, $productId)  {  *// Получение текущего пользователя* $user = auth()->user();  *// Проверка, есть ли у пользователя заказы, в которых есть указанный товар* $order = Order::*where*('user\_id', $user->id)  ->whereHas('compound', function ($query) use ($productId) {  $query->where('product\_id', $productId);  })  ->first();  *// Если заказ не найден, возвращаем сообщение об ошибке* if (!$order) {  throw new ApiException(404, 'Вы не можете оставить отзыв на товар, который вы не покупали');  }  *// Проверяем, оставлял ли пользователь уже отзыв на данный товар* $existingReview = Review::*where*('user\_id', $user->id)  ->where('product\_id', $productId)  ->exists();  *// Если пользователь уже оставлял отзыв, возвращаем сообщение об ошибке* if ($existingReview) {  throw new ApiException(409, 'Вы уже оставили отзыв на этот товар');   }  *// Сохранение нового отзыва* $review = new Review([  'rating' => $request->input('rating'),  'textReview' => $request->input('textReview'),  'user\_id' => $user->id,  'product\_id' => $productId,  ]);  $review->save();  *// Возвращаем сообщение об успешном сохранении отзыва* return response()->json(['message' => 'Отзыв успешно сохранен'], 200);  } } |

* Контроллер UserController

|  |
| --- |
| class UserController extends Controller {  *//Метод регистрации* public function create(UserCreateRequest $request) {  $user = new User($request->all());  $user->role\_id = 1; *// Устанавливаем значение role\_id(User)* $user->save();  return response([  'message' => 'Регистрация прошла успешно',  ], 201);  }  *//Метод просмотра текущего пользователя* public function this() {  $user = auth()->user();   return response()->json([  'data' => $user  ]);  }  *//Метод обновления профиля* public function updateProfile(UserUpdateRequest $request) {  *// Получаем текущего аутентифицированного пользователя* $user = auth()->user();  *//Обновляем данные* $user->fill($request->all());  *// Сохраняем обновленные данные профиля пользователя* $user->save();  return response()->json(['message' => 'Профиль успешно обновлен'], 200);  }  *//Метод для просмотра всех купленных товаров и другой соответствующей информации* public function compound()  {  $user = auth()->user();  $orders = Order::*where*('user\_id', $user->id)->get();  $result = [];  foreach ($orders as $order) {  $compound = Compound::*where*('order\_id', $order->id)->get();  $products = [];  foreach ($compound as $item) {  $product = Product::*find*($item->product\_id);  if ($product) {  $products[] = $product;  }  }  *// Получаем соответствующий платеж* $payment = Payment::*find*($order->payment\_id);  *// Получаем соответствующий статус* $status = Status::*find*($order->status\_id);  $result[] = [  'order' => $order,  'compound' => $compound,  'products' => $products,  'paymentName' => $payment ? $payment->name : 'Неизвестный способ оплаты',  'statusName' => $status ? $status->name : 'Неизвестный статус',  ];  }  return response($result);  } } |

## Маршрутизация

Все маршруты Laravel должны быть определены в файлах маршрутов, находящихся в нашем каталоге routes. Эти файлы автоматически загружаются нашим поставщиком App\Providers\RouteServiceProvider приложения. Файл routes/web.php определяет маршруты веб-интерфейса. Этим маршрутам назначается группа посредников web, которая обеспечивает такие функции, как состояние сессии и защита от CSRF (Межсайтовая подделка запроса Cross-site request forgery). Маршруты в routes/api.php не сохраняют состояния и им назначается группа посредников API.

Маршруты, определенные в файле routes/api.php, вложены в группу маршрутов в RouteServiceProvider. Внутри этой группы автоматически применяется префикс URI /api, поэтому вам не нужно вручную добавлять его к каждому маршруту в файле маршрутов.

Исходя из технического задания были разработаны следующие маршруты, которые записаны в файле routes\api.php:

|  |
| --- |
| *//Регистрация* Route::post('/register' , [UserController::class, 'create' ]);  *//Авторизация* Route::post('/login' , [AuthController::class, 'login' ]);  *//Просмотр категорий товаров* Route::get('/categories' , [CategoryController::class, 'index']);  *//Просмотр товаров определенной категории* Route::get('/category/{id}' , [ProductController::class, 'showMany']);  *//Просмотр конкретного товара* Route::get('/product/{id}' , [ProductController::class, 'show']);  *//Просмотр всех товаров* Route::get('/products', [ProductController::class, 'index']);  *//Просмотр способов оплаты* Route::get('/payment', [OrderController::class, 'payment']);  *//Просмотр отзывов у товара* Route::get('/product/{id}/review', [ReviewController::class, 'index']);  *//Функционал авторизированного пользователя* Route::middleware('auth:api')->group(function () {  *//Выход* Route::get('/logout', [AuthController::class, 'logout']);  *//Просмотр профиля текущего пользователя* Route::get('/profile', [UserController::class, 'this']);  *//Добавления товара в корзину* Route::post('/product/{id}', [CartController::class, 'addToCart']);  *//Просмотр корзины текущего пользователя* Route::get('/cart', [CartController::class, 'index']);  *//Оформление заказа* Route::post('/checkout', [OrderController::class, 'checkout']);  *//Оставление отзыва для товара* Route::post('/product/{id}/review', [ReviewController::class, 'store']);  *//Редактирование корзины текущего пользователя* Route::patch('/cart', [CartController::class, 'update']);  *//Редактирование профиля текущего пользователя* Route::patch('/profile', [UserController::class, 'updateProfile']);  *//Удалеления товара из корзины* Route::delete('/cart/product/{id}', [CartController::class, 'delete']);  *//Просмотр всех заказов текущего пользователя* Route::get('/orders', [OrderController::class, 'index']);  *//Просм* Route::get('/compound', [UserController::class, 'compound']); }); *//Функционал администратора* Route::middleware('auth:api', 'role:2')->group(function () {  *//Создание категории* Route::post('/admin/category/create', [AdminController::class, 'createCategory']);  *//Создание товара* Route::post('/admin/product/create', [AdminController::class, 'createProduct']);  *//Редактирование категории* Route::patch('/admin/category/{id}/edit', [AdminController::class, 'updateCategory']);  *//Редактирование товара* Route::post('/admin/product/{id}/edit', [AdminController::class, 'updateProduct']);  *//Удаление категории* Route::delete('/admin/category/{id}/delete', [AdminController::class, 'deleteCategory']);  *//Удаление товара* Route::delete('/admin/product/{id}/delete', [AdminController::class, 'deleteProduct']); }); |

# ГЛАВА 4. АВТОТЕСТЫ

**Название теста:** Status code is 401, Unauthorized

**Описание:** Этот тест проверяет аутентификацию пользователя, он отправляет запрос без авторизации на ресурс, ожидая получить ответ с кодом состояния 401 и сообщением "Unauthorized".

**Ресурсы:**

|  |  |
| --- | --- |
| Method | Path |
| GET | {{host}}/cart |
| PATCH | {{host}}/cart |
| POST | {{host}}/checkout |
| POST | {{host}}/product/{product\_id}/review |
| PATCH | {{host}}/profile |
| GET | {{host}}/profile |
| DEL | {{host}}/cart/product/{product\_id} |
| GET | {{host}}/logout |
| POST | {{host}}/admin/category/create |
| POST | {{host}}/admin/product/create |
| POST | {{host}}/admin/product/{product\_id}/edit |
| PATCH | {{host}}/admin/category/{category\_id}/edit |
| DEL | {{host}}/admin/category/{category\_id}/delete |
| DEL | {{host}}/admin/product/{product\_id}/delete |

**Код теста:**

|  |
| --- |
| pm.**test**("Status code is 401", **function** () {      pm.response.to.have.status(401);  });  pm.**test**("Unauthorized", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Unauthorized");  }); |

**Название теста:** Status code is 403, Forbidden for you

**Описание:** Этот тест проверяет ошибку доступа для пользователя с недостаточными правами. Он отправляет запрос на ресурс с токеном аутентификации пользователя, ожидая получить ответ с кодом состояния 403 и сообщением "Forbidden for you".

**Ресурсы:**

|  |  |
| --- | --- |
| Method | Path |
| POST | {{host}}/admin/category/create |
| POST | {{host}}/admin/product/create |
| POST | {{host}}/admin/product/{product\_id}/edit |
| PATCH | {{host}}/admin/category/{category\_id}/edit |
| DEL | {{host}}/admin/category/{category\_id}/delete |
| DEL | {{host}}/admin/product/{product\_id}/delete |

**Код теста:**

|  |
| --- |
| pm.**test**("Status code is 403", **function** () {      pm.response.to.have.status(403);  });  pm.**test**("Forbidden for you", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Forbidden for you");  }); |

**Название теста:** Status code is 422, response

**Описание:** Этот тест проверяет ошибку валидации данных. Он отправляет запрос на ресурс с неверными полями данных, ожидая получить ответ с кодом состояния 422 и наличием в ответе поля "errors".

**Ресурсы:**

|  |  |
| --- | --- |
| Method | Path |
| POST | {{host}}/register |
| POST | {{host}}/login |
| POST | {{host}}/checkout |
| POST | {{host}}/product/{product\_id}/review |
| PATCH | {{host}}/profile |
| POST | {{host}}/admin/category/create |
| POST | {{host}}/admin/product/create |
| POST | {{host}}/admin/product/{product\_id}/edit |
| PATCH | {{host}}/admin/category/{categpry\_id}/edit |
|  | {{host}} |
|  | {{host}} |
|  | {{host}} |

**Код теста:**

|  |
| --- |
| pm.**test**("Status code is 422", **function** () {      pm.response.to.have.status(422);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("errors");  }); |

**Название теста: Register201**

**Описание:** Этот тест проверяет успешную регистрацию пользователя через API. Он отправляет POST запрос на регистрацию пользователя с заполненными данными, ожидая получить ответ с кодом состояния 201 и сообщением "Регистрация прошла успешно".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 201", **function** () {      pm.response.to.have.status(201);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Регистрация прошла успешно");  }); |

**Название теста: Login200**

**Описание:** Этот тест проверяет успешный вход пользователя в систему. Он отправляет POST запрос на вход пользователя с корректными учетными данными, ожидая получить ответ с кодом состояния 200 и наличием api\_token.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("Проверка наличия api\_token", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("data");      pm.expect(jsonData.data).to.have.property("api\_token");      pm.expect(jsonData.data.api\_token).to.not.be.empty;  }); |

**Название теста: Categories200**

**Описание:** Этот тест проверяет успешное получение списка категорий. Он отправляет GET запрос на эндпоинт /categories, ожидая получить ответ с кодом состояния 200 и непустым списком категорий.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("Проверка наличия data", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("data");      pm.expect(jsonData.data).to.not.be.empty;  }); |

**Название теста: ProductsInCategory200**

**Описание:** Этот тест проверяет успешное получение списка продуктов в определенной категории. Он отправляет GET запрос на эндпоинт /category/{category\_id}, ожидая получить ответ с кодом состояния 200 и товарами, принадлежащими указанной категории.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("Проверка значения category\_id", **function** () {      var jsonData **=** pm.response.json();      var data **=** jsonData.data;  **for** (var i **=** 0; i **<** data.length; i**++**) {          pm.expect(data[i].category\_id).to.equal(1);      }  }); |

**Название теста: Product200**

**Описание:** Этот тест проверяет успешное получение информации о продукте. Он отправляет GET запрос на эндпоинт /product/{product\_id}, ожидая получить ответ с кодом состояния 200 и информацией о запрошенном продукте.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("Проверка значения id", **function** () {      var jsonData **=** pm.response.json();      var product **=** jsonData.data;      pm.expect(product.id).to.equal(1);  }); |

**Название теста: Product404**

**Описание:** Этот тест проверяет обработку ошибки при запросе несуществующего продукта. Он отправляет GET запрос на эндпоинт /product/{nonexistent\_product\_id}, ожидая получить ответ с кодом состояния 404 и сообщением о том, что продукт не найден.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("Проверка формата ответа", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("success", **false**);      pm.expect(jsonData).to.have.property("code", 404);      pm.expect(jsonData).to.have.property("message", "Товар не найден");  }); |

**Название теста: Category404**

**Описание:** Этот тест проверяет обработку ошибки при запросе несуществующей категории. Он отправляет GET запрос на эндпоинт /category/{nonexistent\_category\_id}, ожидая получить ответ с кодом состояния 404 и сообщением о том, что товары в данной категории не найдены.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("Проверка формата ответа", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("success", **false**);      pm.expect(jsonData).to.have.property("code", 404);      pm.expect(jsonData).to.have.property("message", "Товары не найдены");  }); |

**Название теста: Cart200**

**Описание:** Этот тест проверяет успешное получение информации о корзине авторизированного пользователя. Он отправляет GET запрос на эндпоинт /cart, ожидая получить ответ с кодом состояния 200 и информацией о товарах в корзине.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("cart\_items");      pm.expect(jsonData).to.have.property("total");  }); |

**Название теста: AddToCart200**

**Описание:** Этот тест проверяет успешное добавление товара в корзину. Он отправляет POST запрос на эндпоинт /product/{product\_id} с информацией о добавляемом товаре, ожидая получить ответ с кодом состояния 200 и сообщением "Продукт добавлен в корзину".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Продукт добавлен в корзину");  }); |

**Название теста: AddToCart404**

**Описание:** Этот тест проверяет обработку ошибки при добавлении несуществующего товара в корзину. Он отправляет POST запрос на эндпоинт /product/{nonexistent\_product\_id}, ожидая получить ответ с кодом состояния 404 и сообщением о том, что товар не найден.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Товар не найден");  }); |

**Название теста: AddToCart400**

**Описание:** Этот тест проверяет обработку ошибки при добавлении товара в корзину с недопустимым количеством. Он отправляет POST запрос на эндпоинт /product/{product\_id} с недопустимым количеством товара, ожидая получить ответ с кодом состояния 400 и сообщением о том, что количество товара должно быть положительным числом.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 400", **function** () {      pm.response.to.have.status(400);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Недостаточное количество товара в наличии");  }); |

**Название теста: EditCart200**

**Описание:** Этот тест проверяет успешное изменение количества товара в корзине. Он отправляет PATCH запрос на эндпоинт /cart с информацией о товаре и его новом количестве, ожидая получить ответ с кодом состояния 200 и сообщением "Количество товара в корзине успешно обновлено".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Количество товара в корзине успешно обновлено");  }); |

**Название теста: EditCart400**

**Описание:** Этот тест проверяет обработку ошибки при изменении количества товара в корзине на недопустимое количество. Он отправляет PATCH запрос на эндпоинт /cart с отрицательным количеством товара, ожидая получить ответ с кодом состояния 400 и сообщением о том, что количество товара должно быть положительным числом.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 400", **function** () {      pm.response.to.have.status(400);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Количество товара должно быть положительным числом");  }); |

**Название теста: EditCart404**

**Описание:** Этот тест проверяет обработку ошибки при изменении количества несуществующего товара в корзине. Он отправляет PATCH запрос на эндпоинт /cart с информацией о несуществующем товаре, ожидая получить ответ с кодом состояния 404 и сообщением о том, что товар не найден в корзине.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("Not Found", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Товар не найден в вашей корзине");  }); |

**Название теста: Checkout200**

**Описание:** Этот тест проверяет успешное оформление заказа. Он отправляет POST запрос на эндпоинт /checkout с информацией об адресе доставки и методе оплаты, ожидая получить ответ с кодом состояния 200 и сообщением "Заказ успешно оформлен".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Заказ успешно оформлен");  }); |

**Название теста: Checkout404**

**Описание:** Этот тест проверяет обработку ошибки при оформлении заказа из пустой корзины. Он отправляет POST запрос на эндпоинт /checkout с информацией об адресе доставки и методе оплаты, ожидая получить ответ с кодом состояния 404 и сообщением о том, что заказ не удалось оформить из-за пустой корзины.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Не удалось оформить заказ, возможно, ваша корзина пуста");  }); |

**Название теста: Review200**

**Описание:** Этот тест проверяет успешное сохранение отзыва о товаре. Он отправляет POST запрос на эндпоинт /product/{product\_id}/review с рейтингом и текстом отзыва, ожидая получить ответ с кодом состояния 200 и сообщением "Отзыв успешно сохранен".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Отзыв успешно сохранен");  }); |

**Название теста: Review404**

**Описание:** Этот тест проверяет обработку ошибки при попытке оставить отзыв на несуществующий товар. Он отправляет POST запрос на эндпоинт /product/{nonexistent\_product\_id}/review, ожидая получить ответ с кодом состояния 404 и сообщением о том, что невозможно оставить отзыв на товар, который не был приобретен.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Вы не можете оставить отзыв на товар, который вы не покупали");  }); |

**Название теста: Review409**

**Описание:** Этот тест проверяет обработку ошибки при попытке повторного оставления отзыва на товар. Он отправляет POST запрос на эндпоинт /product/{product\_id}/review, ожидая получить ответ с кодом состояния 409 и сообщением о том, что отзыв уже был оставлен.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 409", **function** () {      pm.response.to.have.status(409);  });  pm.**test**("Error", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Вы уже оставили отзыв на этот товар");  }); |

**Название теста: EditProfile200**

**Описание:** Этот тест проверяет успешное обновление профиля пользователя. Он отправляет PATCH запрос на эндпоинт /profile с новым именем и фамилией пользователя, ожидая получить ответ с кодом состояния 200 и сообщением "Профиль успешно обновлен".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Профиль успешно обновлен");  }); |

**Название теста: Profile200**

**Описание:** Этот тест проверяет успешное получение профиля пользователя. Он отправляет GET запрос на эндпоинт /profile, ожидая получить ответ с кодом состояния 200 и наличием данных профиля в ответе.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("data");  }); |

**Название теста: DeleteFromCart200**

**Описание:** Этот тест проверяет успешное удаление продукта из корзины. Он отправляет DELETE запрос на эндпоинт /cart/product/{product\_id}, ожидая получить ответ с кодом состояния 200 и сообщением "Товар успешно удален из корзины".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Товар успешно удален из корзины");  }); |

**Название теста: DeleteFromCart404**

**Описание:** Этот тест проверяет обработку ошибки при попытке удалить несуществующий продукт из корзины. Он отправляет DELETE запрос на эндпоинт /cart/product/{nonexistent\_product\_id}, ожидая получить ответ с кодом состояния 404 и сообщением о том, что товар не найден в корзине.

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Товар не найден в вашей корзине");  }); |

**Название теста: Logout200**

**Описание:** Этот тест проверяет успешное выход пользователя из системы. Он отправляет GET запрос на эндпоинт /logout с токеном аутентификации, ожидая получить ответ с кодом состояния 200 и сообщением "Вы успешно вышли из системы".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("Проверка формата ответа", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("data");      pm.expect(jsonData.data).to.have.property("message", "Вы успешно вышли из системы");  }); |

**Название теста: CategoryCreate201**

**Описание:** Этот тест проверяет успешное создание новой категории товаров администратором. Он отправляет POST запрос на эндпоинт /admin/category/create с данными о новой категории, ожидая получить ответ с кодом состояния 201 и сообщением "Категория успешно создана".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 201", **function** () {      pm.response.to.have.status(201);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Категория успешно создана");  }); |

**Название теста: ProductCreate201**

**Описание:** Этот тест проверяет успешное создание нового товара администратором. Он отправляет POST запрос на эндпоинт /admin/product/create с данными о новом товаре, ожидая получить ответ с кодом состояния 201 и сообщением "Продукт успешно создан".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 201", **function** () {      pm.response.to.have.status(201);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Продукт успешно создан");  }); |

**Название теста: ProductEdit200**

**Описание:** Этот тест проверяет успешное редактирование товара администратором. Он отправляет POST запрос на эндпоинт /admin/product/1/edit с данными отредактированного товара, ожидая получить ответ с кодом состояния 200 и сообщением "Товар успешно обновлен".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Товар успешно обновлен");  }); |

**Название теста: ProductEdit404**

**Описание:** Этот тест проверяет обработку ошибки при редактировании несуществующего товара. Он отправляет POST запрос на эндпоинт /admin/product/1000/edit с данными отредактированного товара, ожидая получить ответ с кодом состояния 404 и сообщением "Товар не найден".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("Unauthorized", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Товар не найден");  }); |

**Название теста: EditCategory200**

**Описание:** Этот тест проверяет успешное редактирование категории администратором. Он отправляет PATCH запрос на эндпоинт /admin/category/1/edit с данными измененной категории, ожидая получить ответ с кодом состояния 200 и сообщением "Категория успешно обновлена".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Категория успешно обновлена");  }); |

**Название теста: EditCategory404**

**Описание:** Этот тест проверяет обработку ошибки при редактировании несуществующей категории. Он отправляет PATCH запрос на эндпоинт /admin/category/999/edit с данными измененной категории, ожидая получить ответ с кодом состояния 404 и сообщением "Категория не найдена".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("Unauthorized", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Категория не найдена");  }); |

**Название теста: DeleteCategory200**

**Описание:** Этот тест проверяет успешное удаление категории администратором. Он отправляет DELETE запрос на эндпоинт /admin/category/5/delete, ожидая получить ответ с кодом состояния 200 и сообщением "Категория успешно удалена".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Категория успешно удалена");  }); |

**Название теста: DeleteCategory404**

**Описание:** Этот тест проверяет обработку ошибки при попытке удаления несуществующей категории. Он отправляет DELETE запрос на эндпоинт /admin/category/999/delete, ожидая получить ответ с кодом состояния 404 и сообщением "Категория не найдена".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("Unauthorized", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Категория не найдена");  }); |

**Название теста: DeleteProduct200**

**Описание:** Этот тест проверяет успешное удаление продукта администратором. Он отправляет DELETE запрос на эндпоинт /admin/product/28/delete, ожидая получить ответ с кодом состояния 200 и сообщением "Продукт успешно удален".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 200", **function** () {      pm.response.to.have.status(200);  });  pm.**test**("response", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Продукт успешно удален");  }); |

**Название теста: DeleteProduct404**

**Описание:** Этот тест проверяет обработку ошибки при попытке удаления несуществующего продукта. Он отправляет DELETE запрос на эндпоинт /admin/product/999/delete, ожидая получить ответ с кодом состояния 404 и сообщением "Продукт не найден".

**Код автотеста:**

|  |
| --- |
| pm.**test**("Status code is 404", **function** () {      pm.response.to.have.status(404);  });  pm.**test**("Unauthorized", **function** () {      var jsonData **=** pm.response.json();      pm.expect(jsonData).to.have.property("message", "Продукт не найден");  }); |

# ГЛАВА 5. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для работы и тестирования API приложения нам потребуется приложение Postman.

## Функционал пользователя

**Регистрация**. Запрос предназначен для создания аккаунта. Если клиент отправил корректные данные, то возвращается сообщение «Регистрация прошла успешно» (см. рис. 5), в противном случае сообщение об ошибке валидации данных (см. рис. 6).

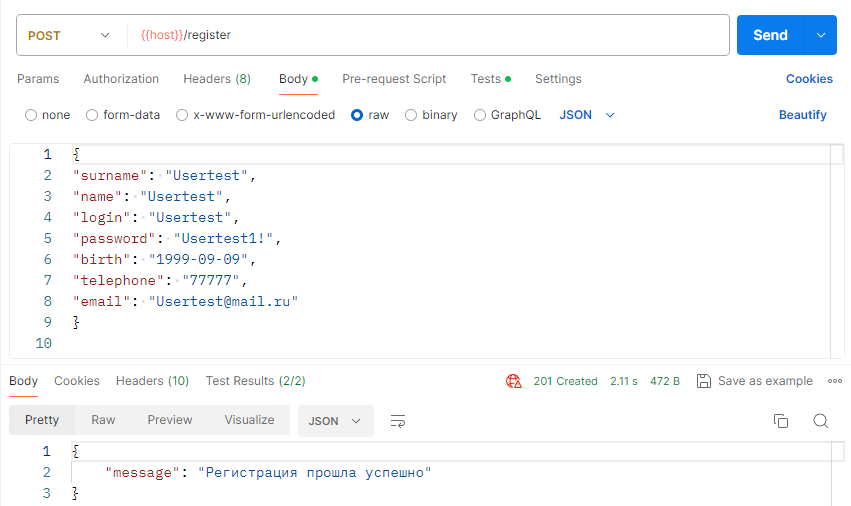


Рисунок 5 – Регистрация прошла успешно

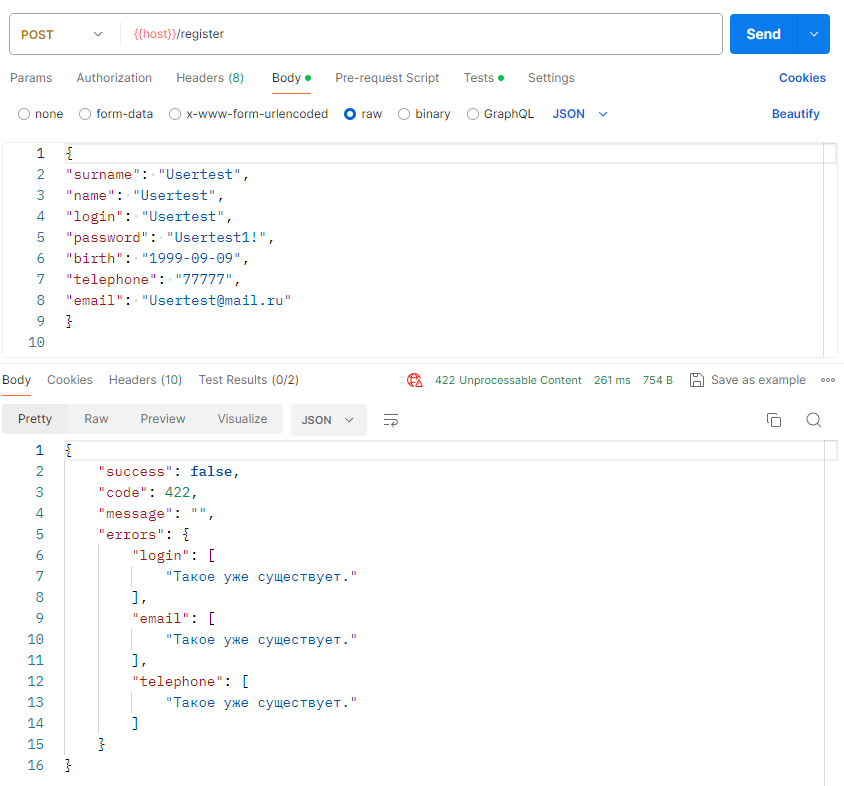


Рисунок 6 – Ошибка валидации при регистрации

**Аутентификация**. Запрос предназначен для входа в аккаунт. При успешном выполнении запроса в качестве ответе возвращается api\_token (см. рис. 7), в противном случае сообщение об ошибке авторизации (см. рис. 8) или валидации данных (см. рис. 9).

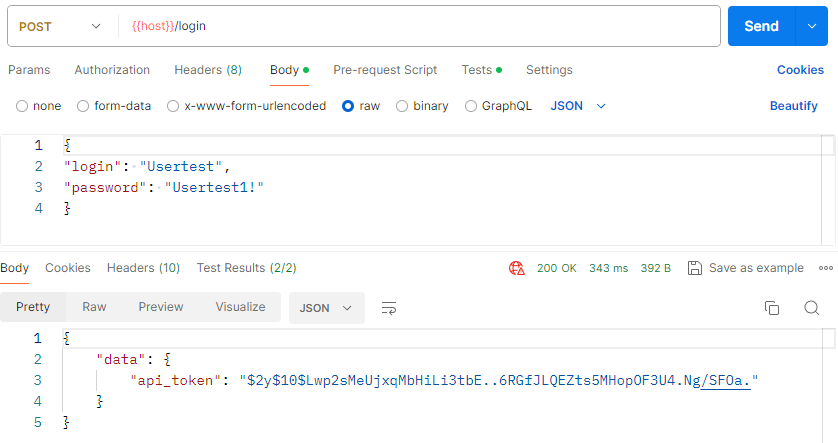


Рисунок 7 – Успешный вход

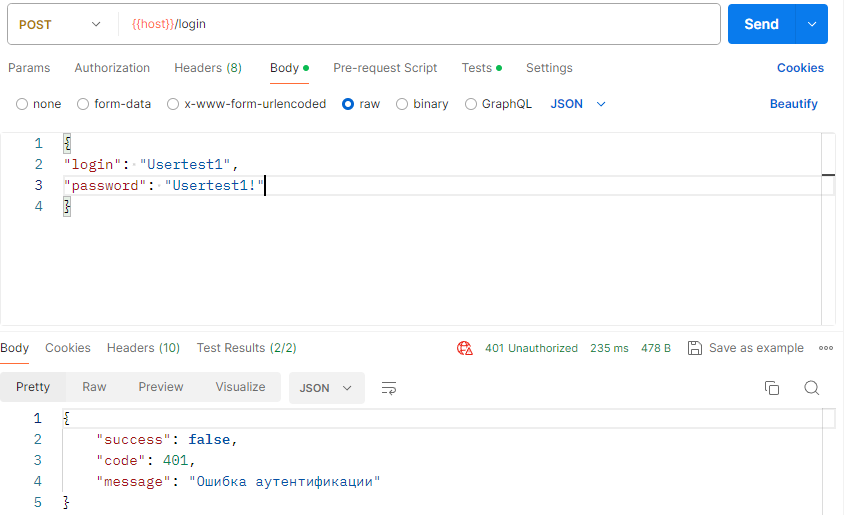


Рисунок 8 – Ошибка аутентификации

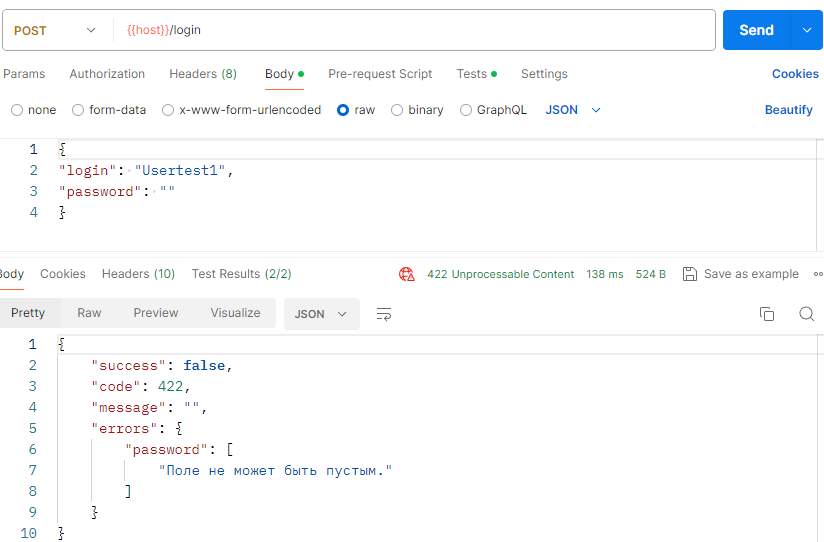


Рисунок 9 – Ошибка валидации данных

**Просмотр категорий.** Запрос предназначен для просмотра всех имеющихся категорий (см. рис. 10).

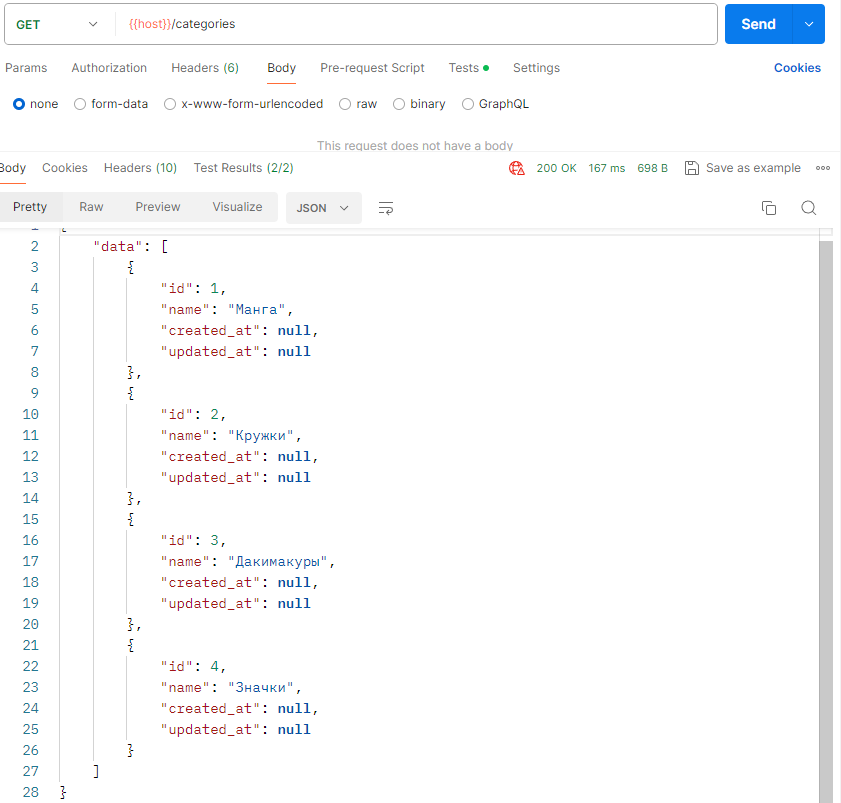


Рисунок 10 – Просмотр категорий

**Просмотр товаров определенной категории.** Запрос предназначен для просмотра всех товаров конкретной категории (см. рис. 11). При попытке просмотра товаров в пустой категории (см. рис. 12).

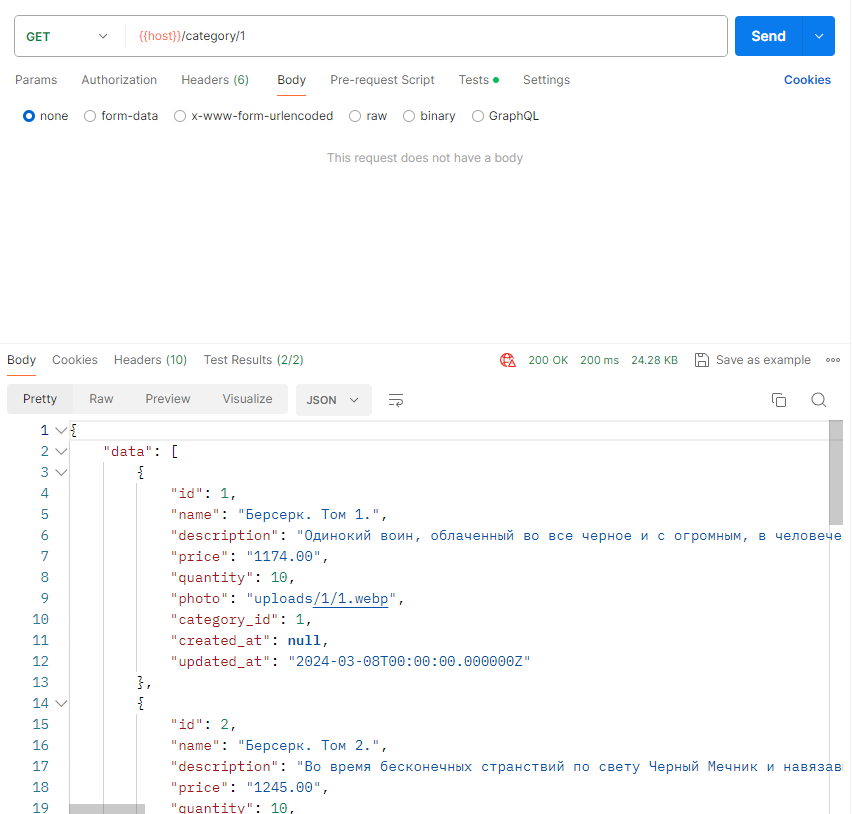


Рисунок 11 – Просмотр товаров конкретной категории

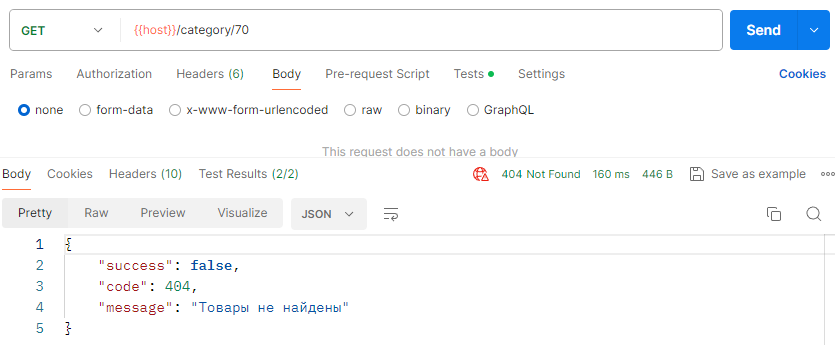


Рисунок 12 – Просмотр товаров пустой категории

**Просмотр конкретного товара.** Запроспредназначен для просмотра определенного товара. При успешном выполнении выводится его полная информация (см. рис. 13), в противном случае ошибка о ненахождении товара (см. рис. 14).

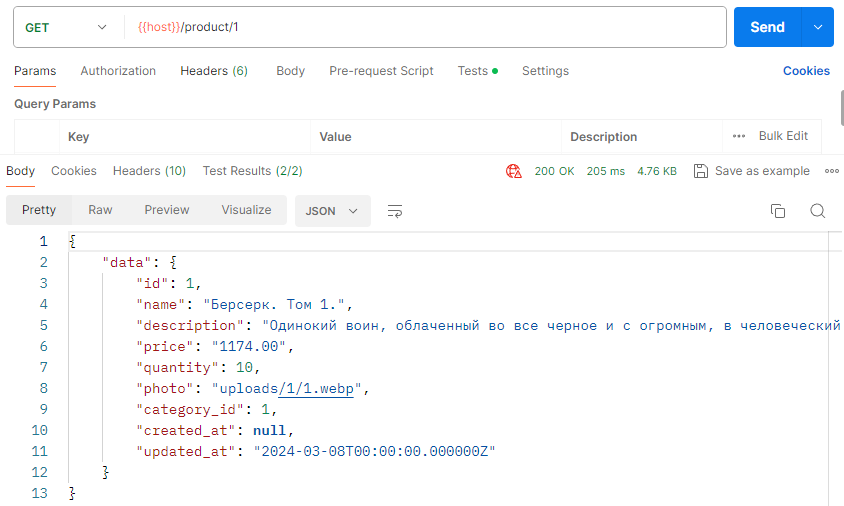


Рисунок 13 – Просмотр конкретного товара

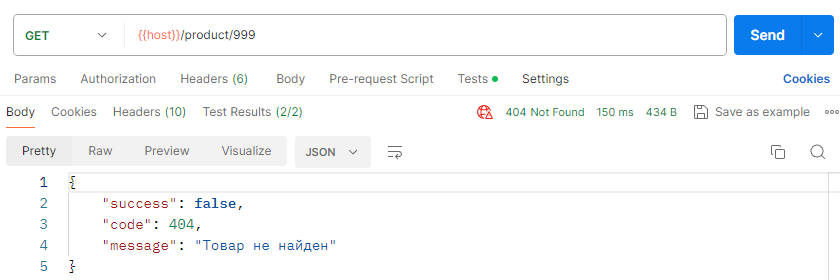


Рисунок 14 – Ошибка о ненахождение товара

## Функционал авторизированного пользователя

При попытке обратиться к каждому последующему запросу, будучи неавторизированным пользователем, будет возникать ошибка (см. рис. 15).

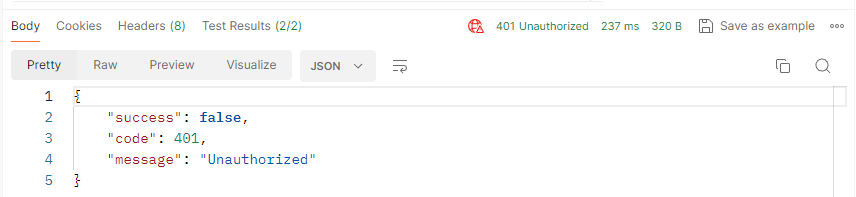


Рисунок 15 – Ошибка не авторизирован

**Просмотр корзины.** Запрос предназначен для просмотра корзины текущего авторизированного пользователя (см. рис. 16).

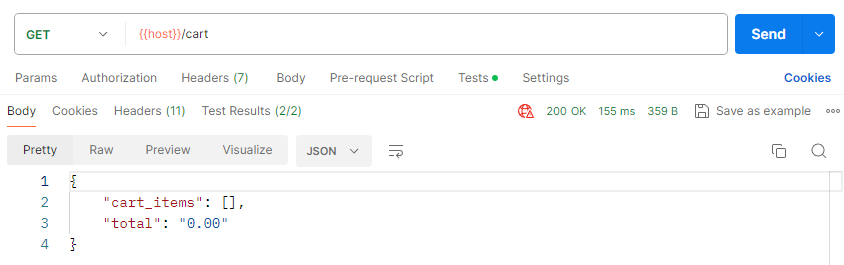


Рисунок 16 – Просмотр корзины

**Добавление товара в корзину.** Запрос предназначен для добавление товара в корзину. При успешном выполнении товар добавляется в корзину (см. рис. 17), при попытке добавить несуществующий товар будет ошибка (см. рис. 18), при попытке добавить большее количество, чем имеется в БД будет ошибка (см. рис. 19).

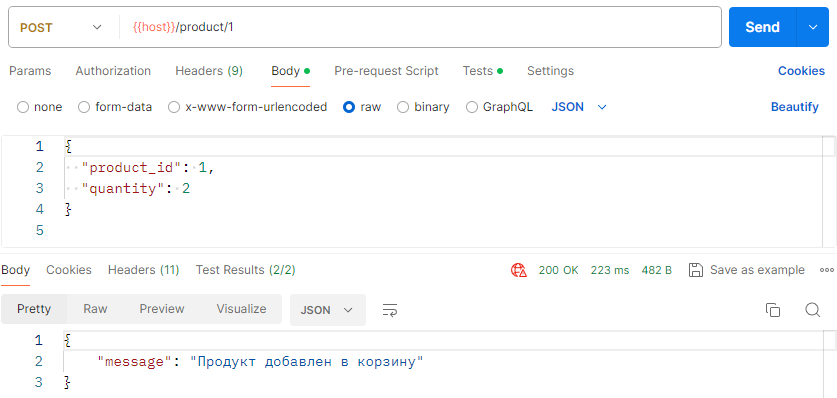


Рисунок 17 – Добавление товара в корзину

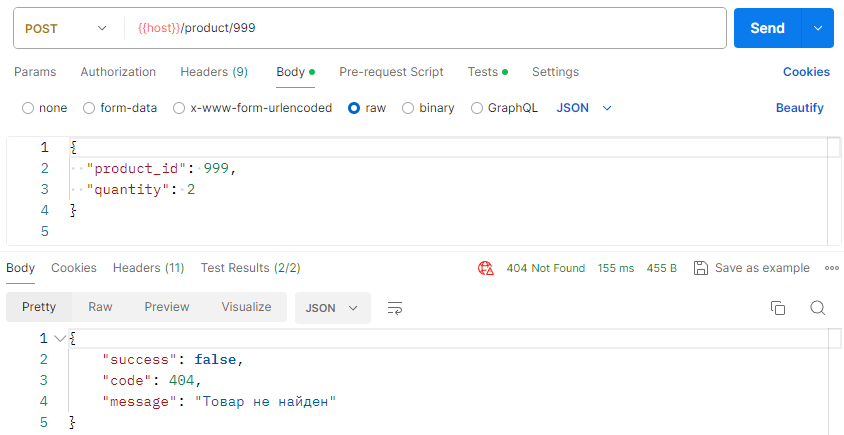


Рисунок 18 – Добавление несуществующего товара в корзину

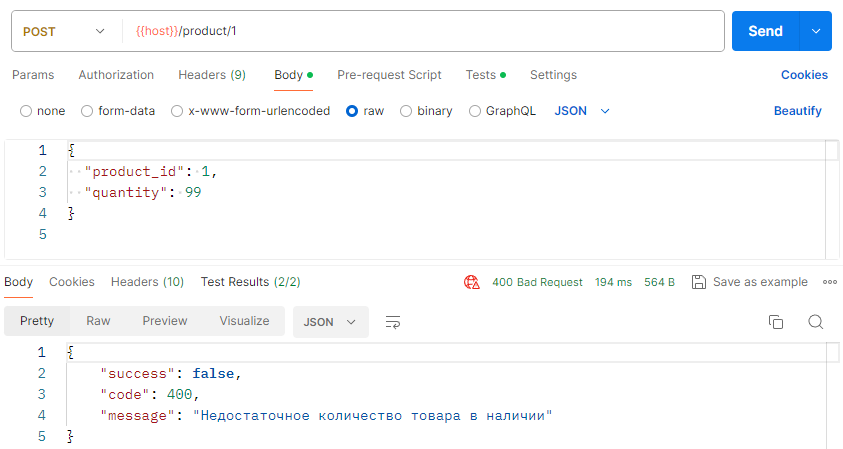


Рисунок 19 – Товар не в наличии

**Редактирование корзины.** Запрос предназначен для редактирование количества товара в корзине. При успешном выполнении выведется сообщение (см. рис. 20). При попытке изменить количество товара, которого нет в корзине (см. рис. 21). При попытке поставить отрицательное значение количеству (см. рис 22).

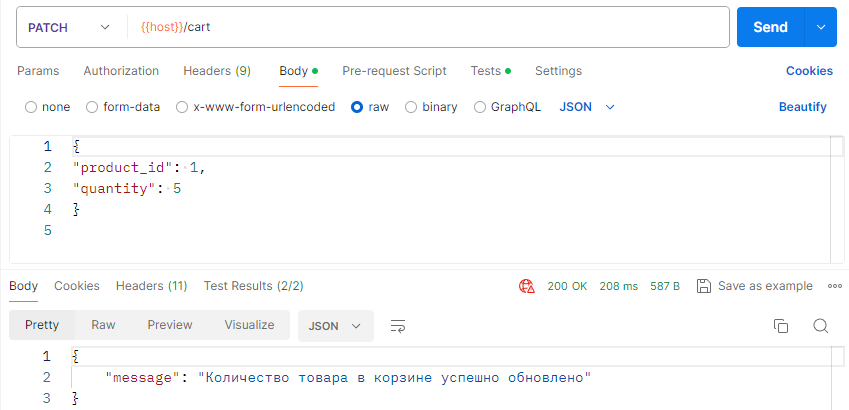


Рисунок 20 – Редактирование корзины

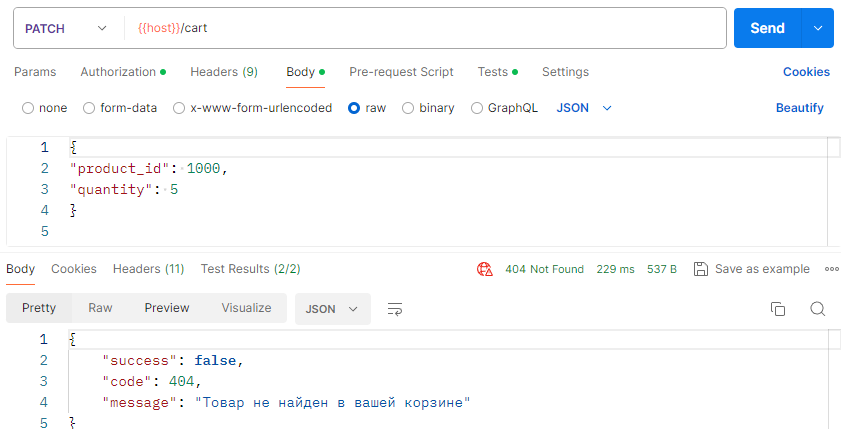


Рисунок 21 – Товар не найден в корзине

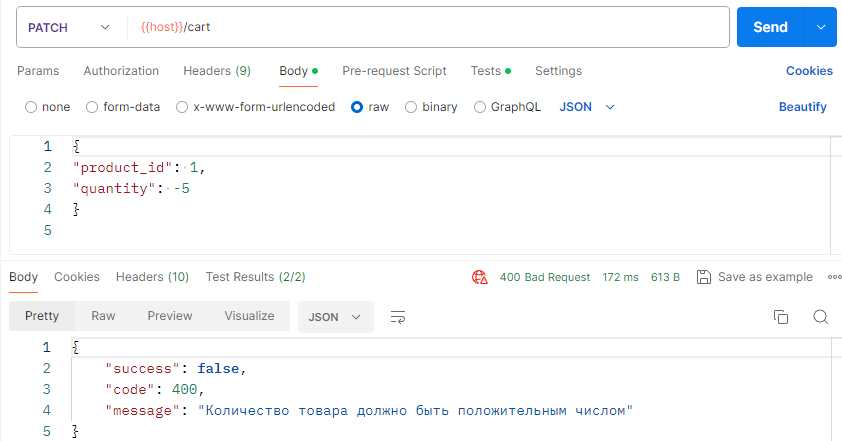


Рисунок 22 – Плохой запрос изменения корзины

**Удаление товара из корзины.** Запрос предназначен для удаление товара из корзины. При успешном выполнении выведется сообщение (см. рис. 23), в противном случае сообщение (см. рис. 24).

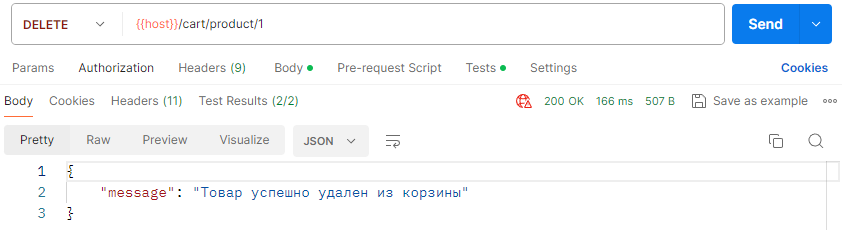
****

Рисунок 23 – Удаление товара из корзины

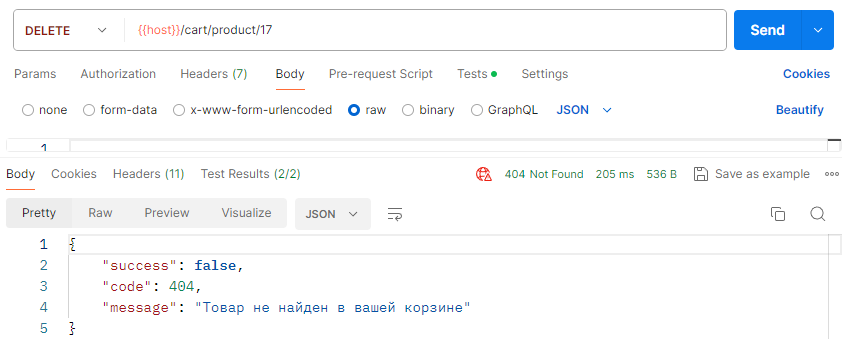
****

Рисунок 24 – Ошибка удаления товара из корзины

**Оформление заказа.** Запрос предназначен для оформления заказа. При успешном выполнении выведется сообщение (см. рис. 25), в противном случае сообщение об ошибке оформления (см. рис. 26) или об ошибке валидации данных (см. рис. 27).

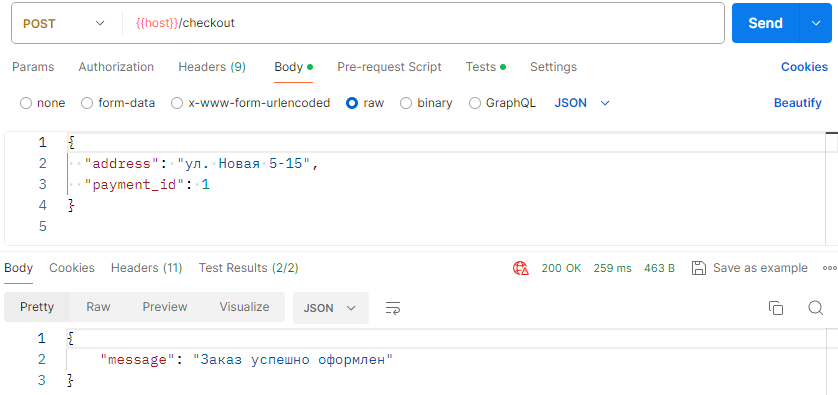
****

Рисунок 25 – Оформление заказа

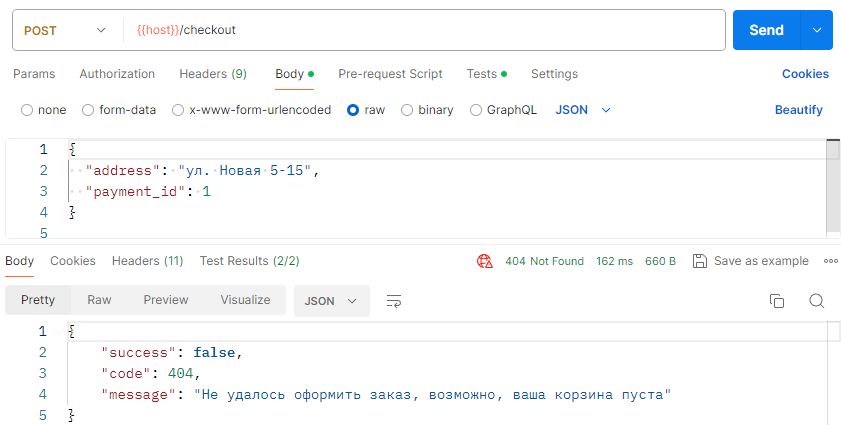
****

Рисунок 26 – Ошибка оформления заказа

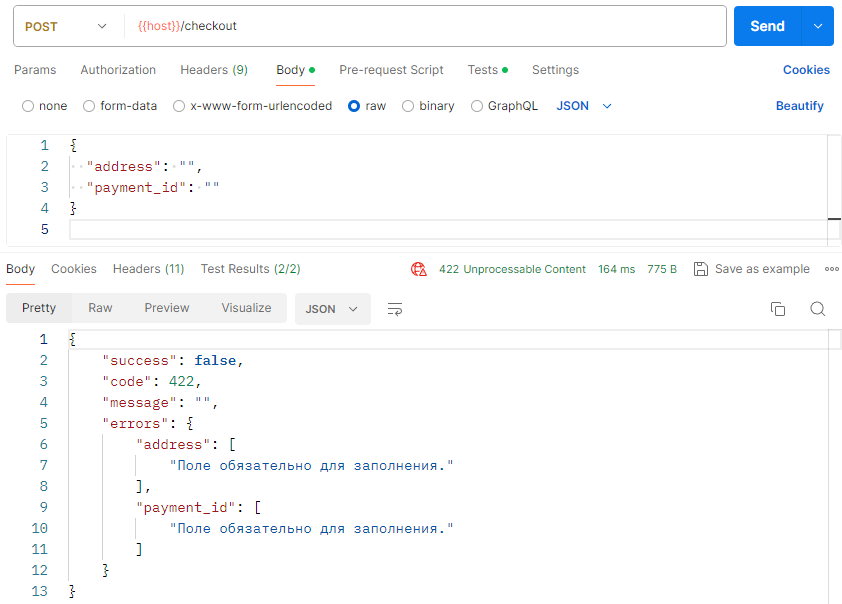
****

Рисунок 27 – Ошибка валидации данных

**Оставление отзыва.** Запрос предназначен для оставления отзыва для купленного товара. В случае успеха выведется сообщение (см. рис. 28), в противном случае об ошибке валидации данных (см. рис. 29). При попытке оставить отзыв на товар, который не был куплен пользователем, будет выводится ошибка (см. рис. 30).

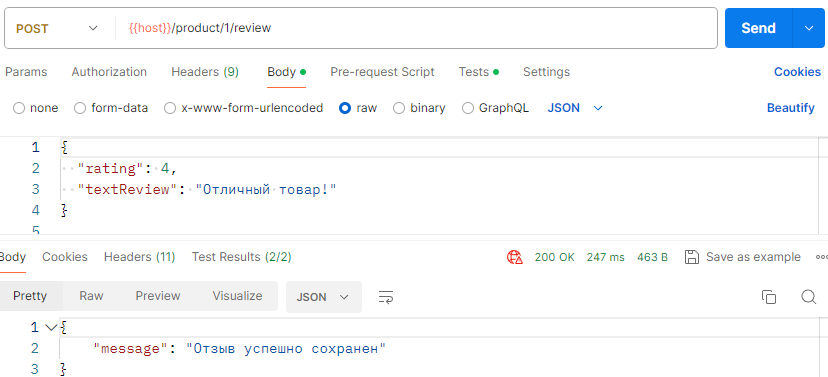


Рисунок 28 – Оставление отзыва

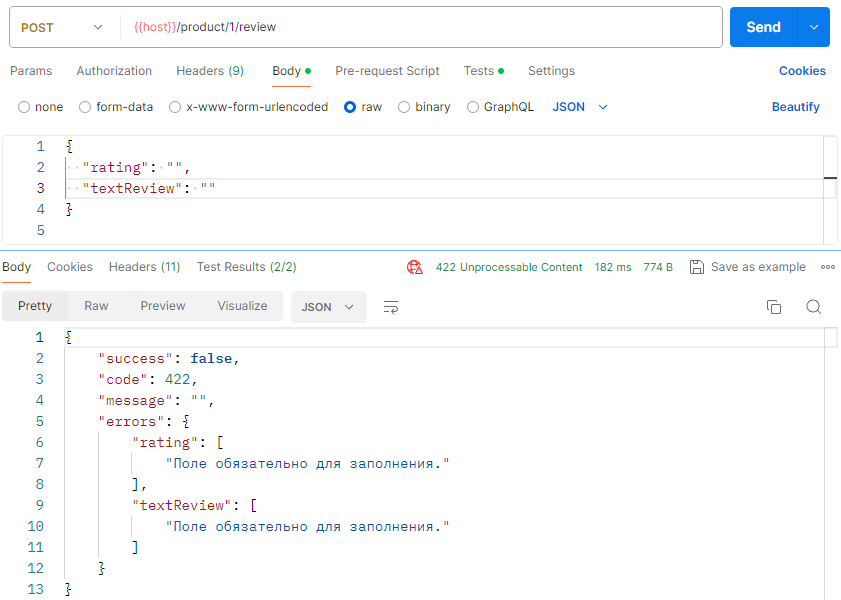


Рисунок 29 – Ошибка валидации данных

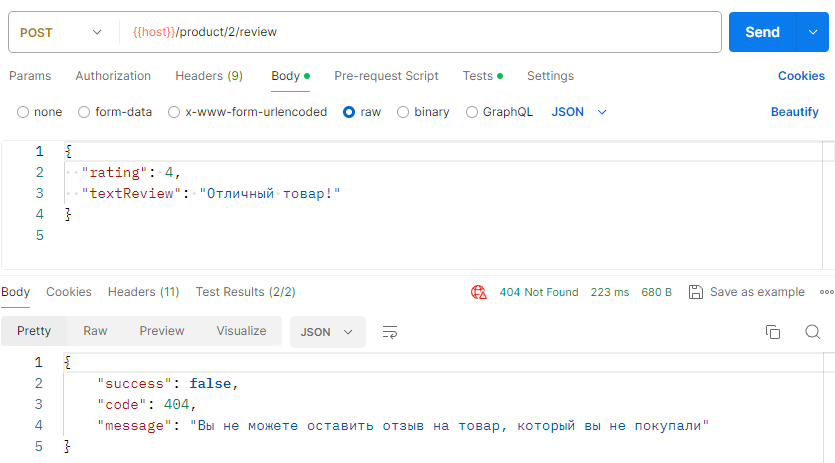


Рисунок 30 – Ошибка добавления отзыва

**Редактирование профиля.** Запрос предназначен для редактирования профиля текущего авторизированного пользователя. В случае успеха выведется сообщение (см. рис. 31), в противном случае сообщение об ошибке валидации данных (см. рис. 32).

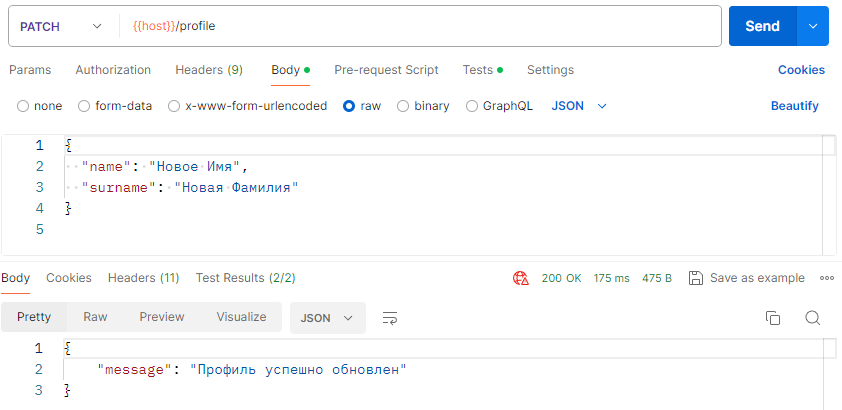
****

Рисунок 41 – Редактирование профиля

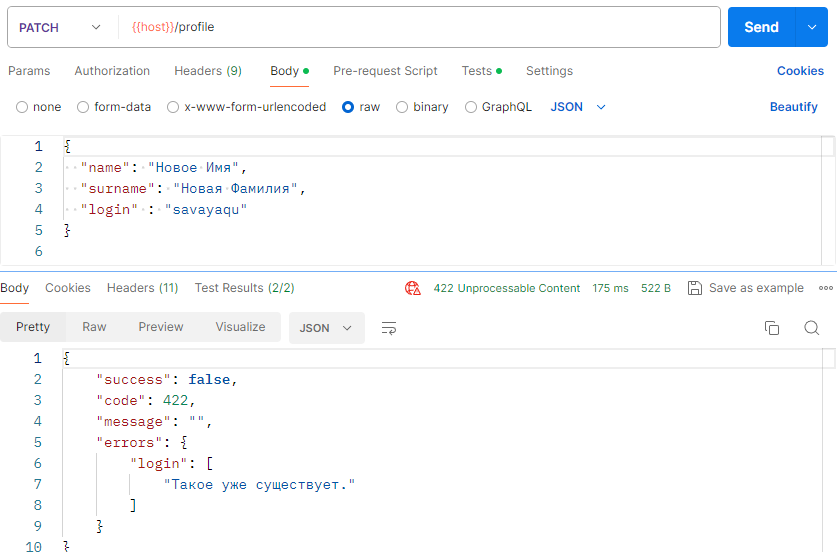
****

Рисунок 32 – Ошибка валидации данных

**Просмотр профиля.** Запрос предназначен для просмотра данных текущего авторизированного пользователя (см. рис. 33).

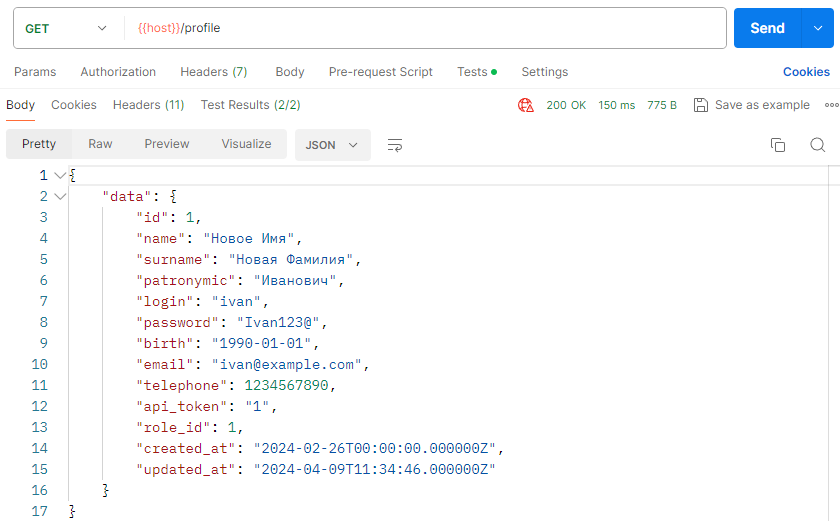
****

Рисунок 53 – Просмотр профиля

**Выход.** Запрос предназначен для очистки значения токена пользователя (см. рис. 34).

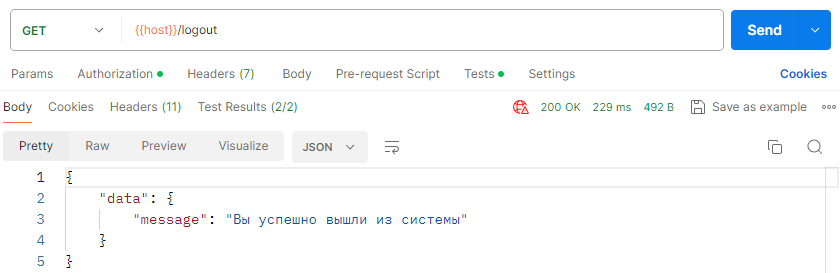
****

Рисунок 34 - Выход

## Функционал администратора

При попытке обратиться к какому-либо запросу, будучи неавторизированным пользователем, будет приходить ошибка (см. рис. 35).

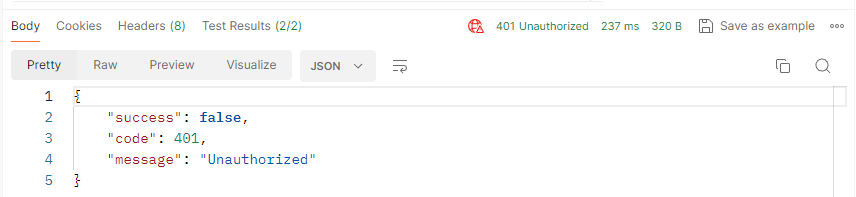


Рисунок 35 – Пользователь не авторизирован

При попытке обратиться к какому-либо запросу, будучи простым пользователем, будет приходить ошибка (см. рис. 36).

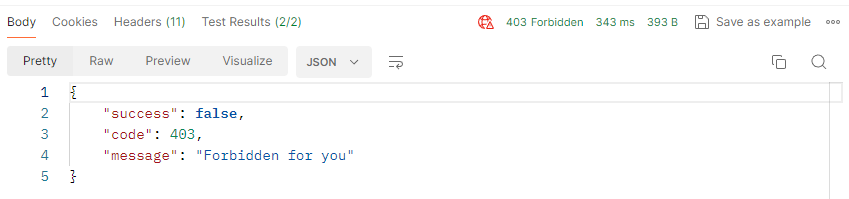


Рисунок 36 – Недостаточно прав

**Создание категории.** Запрос предназначен для создания новой категории. В случае успеха (см. рис. 37), в случае ошибки валидации данных (см. рис. 38).

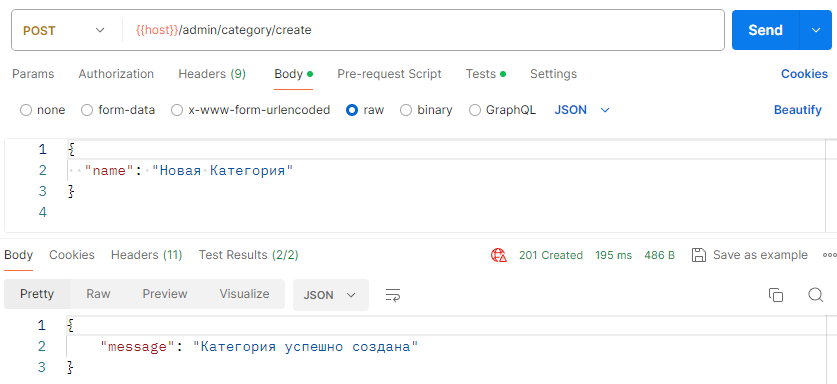


Рисунок 37 – Создание категории

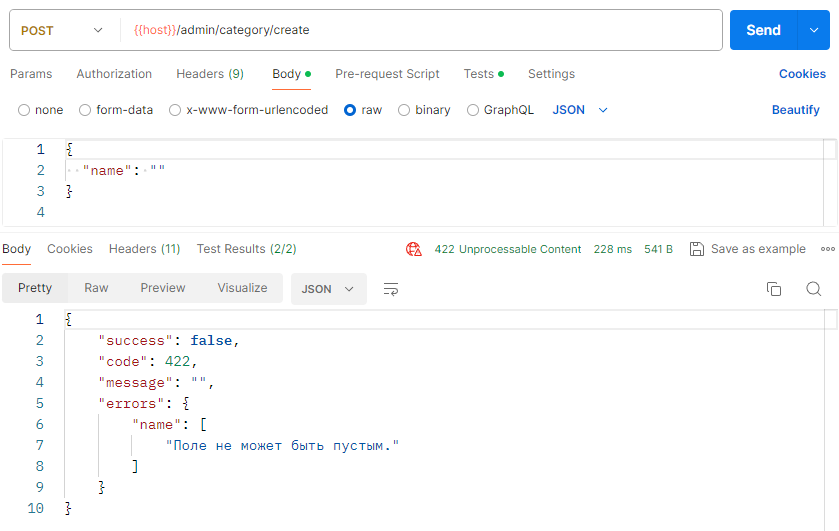


Рисунок 78 – Ошибка валидации данных

**Создание товара.** Запрос предназначен для создания нового товара. В случае успеха (см. рис. 39), в случае ошибки валидации данных (см. рис. 40).

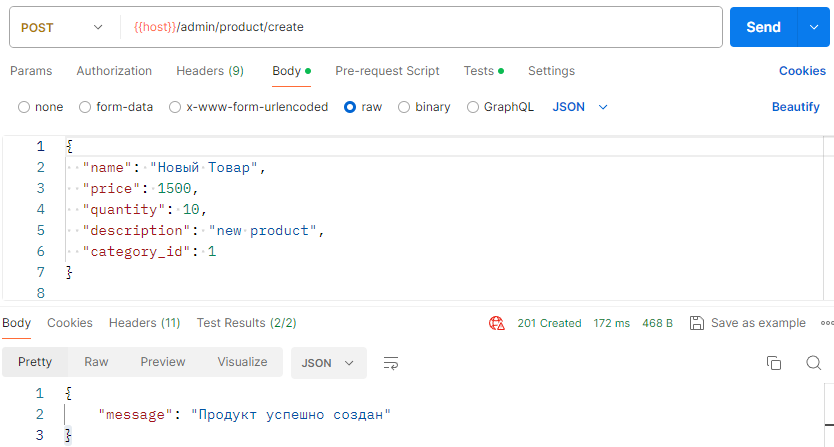
****

Рисунок 39 – Создание товара

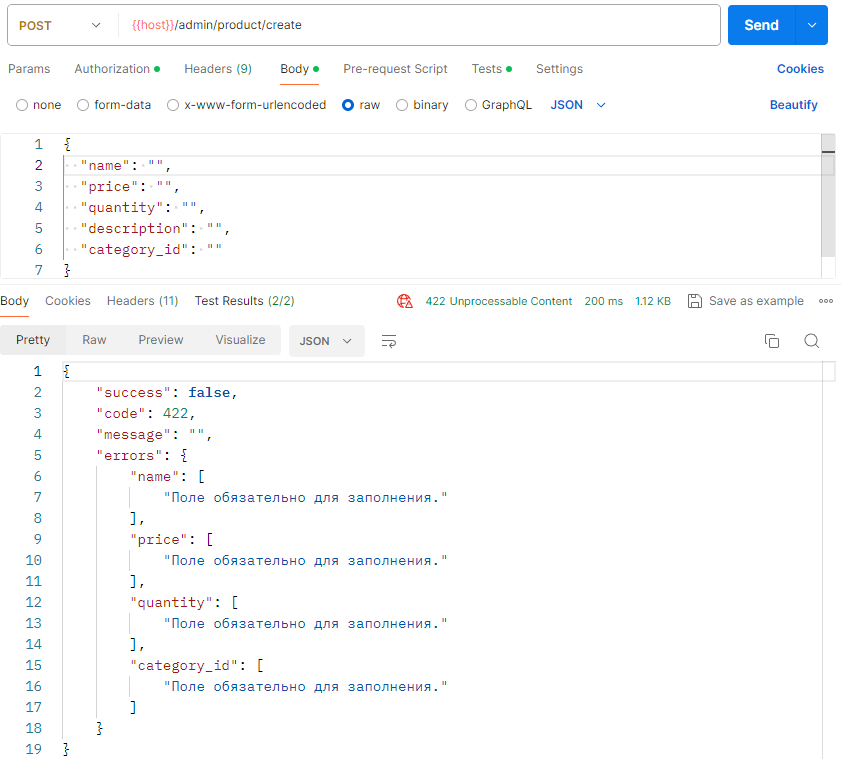
****

Рисунок 40 – Ошибка валидации данных

**Редактирование категории.** Запрос предназначен для редактирования уже существующей категории. В случае успеха (см. рис. 41), в случае ошибки валидации данных (см. рис. 42). При попытке изменить несуществующую категорию (см. рис. 43).

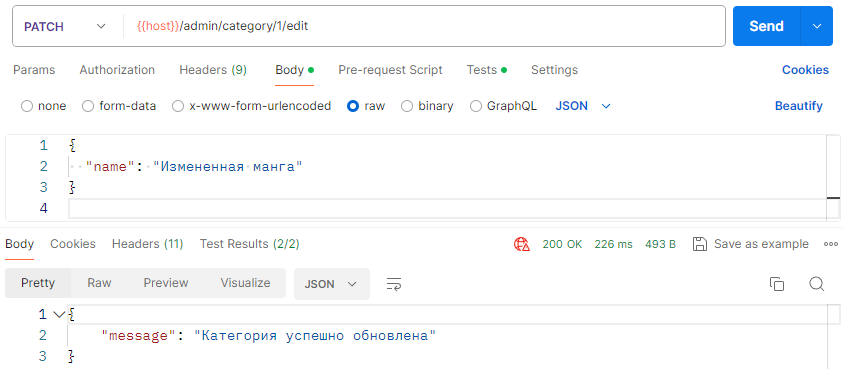
****

Рисунок 41 – Редактирование категории

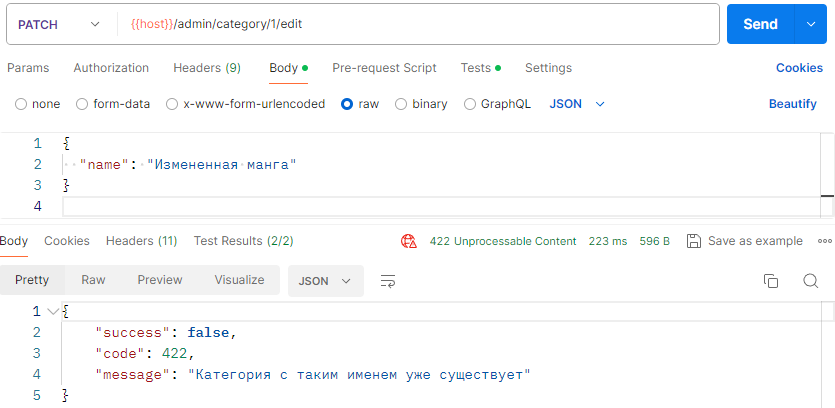
****

Рисунок 42 – Ошибка валидации данных

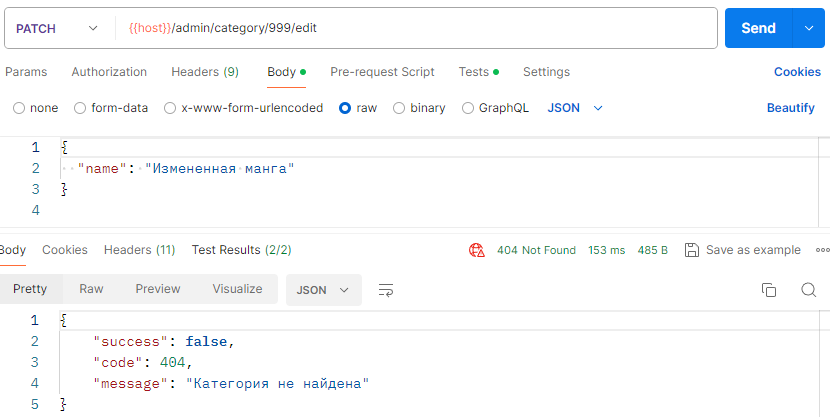
****

Рисунок 43 – Ошибка редактирования категории

**Редактирование товара.** Запрос предназначен для редактирования уже существующего товара. В случае успеха (см. рис. 44), в случае ошибки валидации данных (см. рис. 45). При попытке изменить несуществующий товар (см. рис. 46).

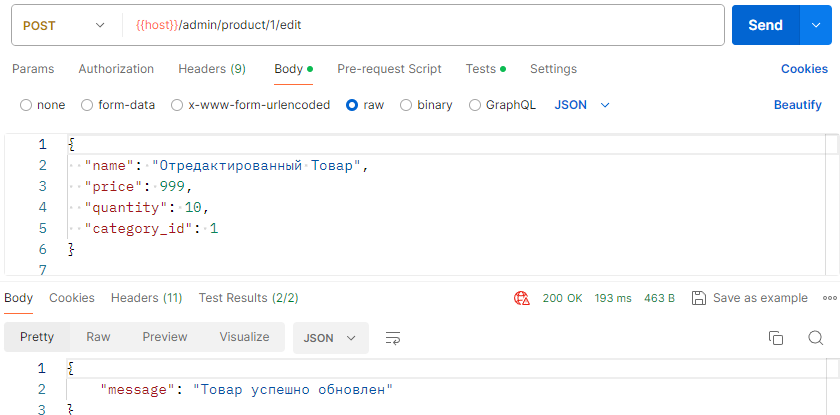
****

Рисунок 44 – Редактирование товара

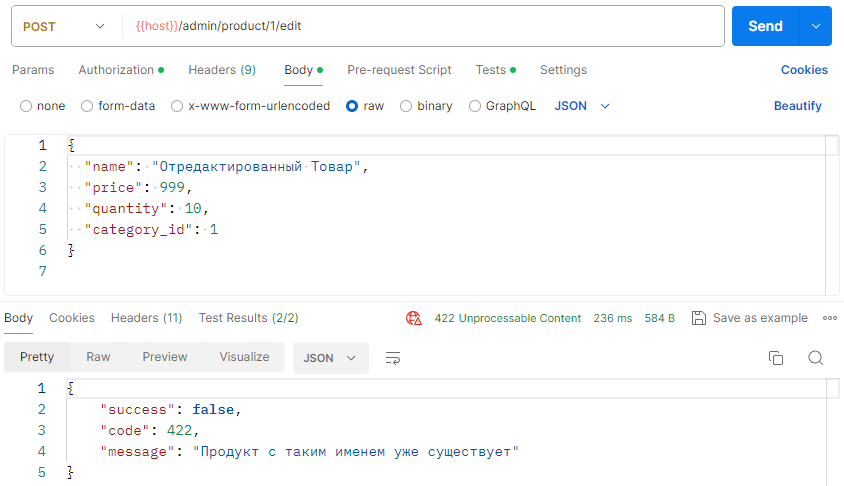
****

Рисунок 45 – Ошибка валидации данных

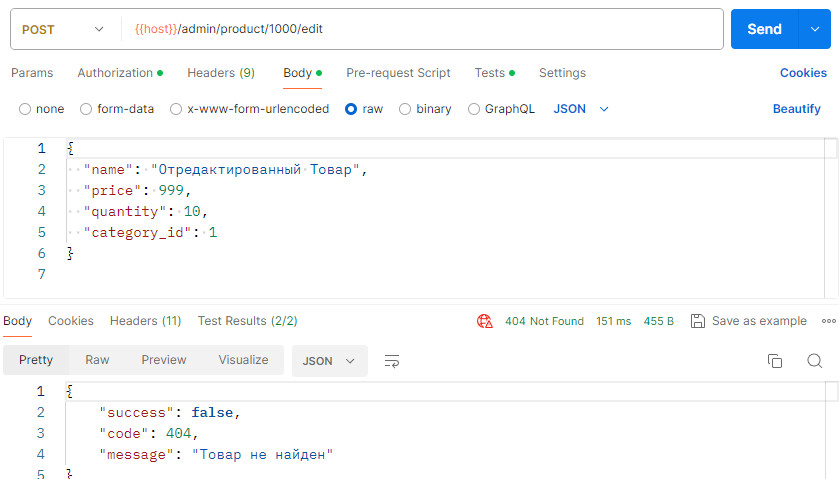
****

Рисунок 46 – Ошибка редактирования несуществующего товара

**Удаление категории.** Запрос предназначен для удаления уже существующей категории. В случае успеха (см. рис. 47). При попытке удалить несуществующую категорию (см. рис. 48).

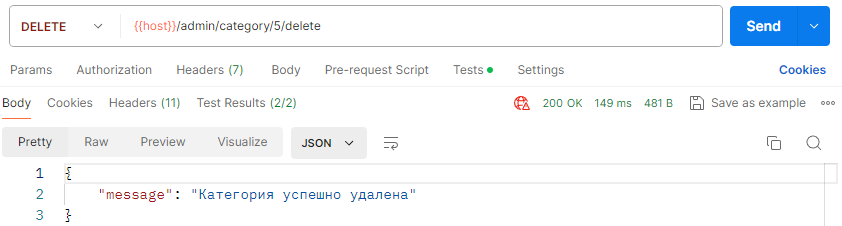
****

Рисунок 47 – Удаление категории

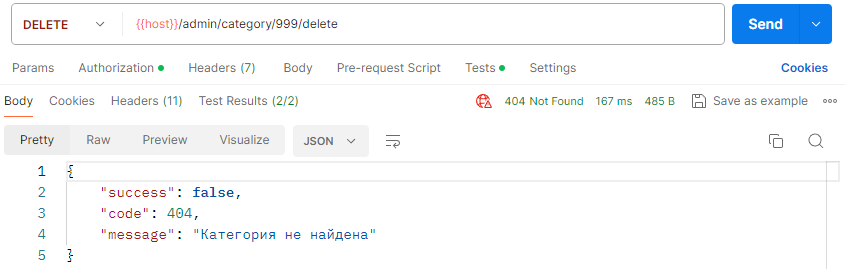
****

Рисунок 48 – Ошибка удаления категории

**Удаление товара.** Запрос предназначен для удаления уже существующего товара. В случае успеха (см. рис. 49). При попытке удалить несуществующий товар (см. рис. 50).

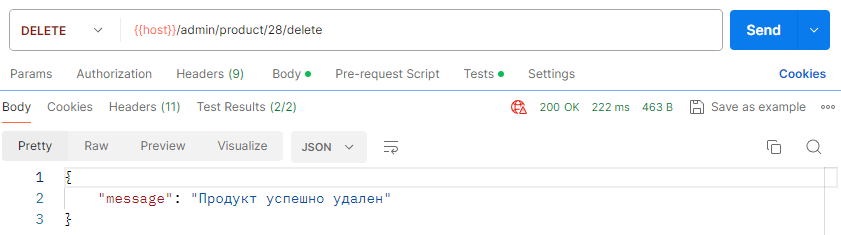
****

Рисунок 49 – Удаление товара

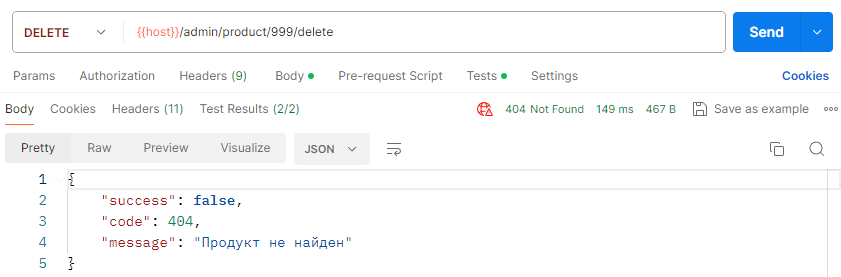
****

Рисунок 50 – Ошибка удаления несуществующего товара

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта были успешно реализованы все поставленные задачи, связанные с разработкой интернет-магазина REST API «Anibloom».

В рамках проекта был проведен анализ предметной области, что позволило глубже понять особенности функционирования интернет-магазина и определить требования к разрабатываемому API.

На основе проведенного анализа было составлено техническое задание, в котором были определены основные функциональные и нефункциональные требования к сервису API.

Была спроектирована и реализована база данных, которая позволяет эффективно хранить и управлять информацией о продуктах, пользователях, заказах и других сущностях интернет-магазина.

Также был спроектирован и разработан функционал информационного продукта, включая возможность регистрации и аутентификации пользователей, управление профилем пользователя, оформление заказов, просмотр каталога товаров и многое другое.

В результате выполнения курсового проекта были значительно улучшены навыки веб-разработки, а также приобретены новые знания и опыт в области проектирования и разработки REST API. Полученный опыт будет полезен в дальнейшей профессиональной деятельности и развитии веб-разработчика.

Таким образом, цель курсового проекта была полностью достигнута, а выполнение всех задач позволило получить ценный опыт и знания в области разработки веб-сервисов.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

***Используемые интернет-источники:***

* Официальная документация Laravel / Режим доступа: <https://laravel.su/docs/10.x/documentation>, открытый.
* Руководство по PHP / Режим доступа: <https://metanit.com/php/tutorial/>, открытый.
* Руководство по MySql / Режим доступа: <https://metanit.com/sql/mysql/>, открытый.
* Список кодов состояний HTTP / Режим доступа: <https://en.wikipedia.org/wiki/List_of_HTTP_status_codes>, открытый.
* Гайды на YouTuve по Postman / Режим доступа: <https://youtube.com/playlist?list=PLzf74dZga1fQwc2fKWemxdwrzsx3wlh-v&si=Li-SKn23ah7OY4t0>, открытый.