|  |  |
| --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования «Полоцкий государственный университет» | |
|  | Факультет информационных технологий  Кафедра технологий программирования |
| Лабораторная работа №3 по курсу «Разработка и анализ требований»  «Учебное руководство по работе с API: Основы, интеграция и практика. Реализация API»  Вариант 5 | |
| Выполнил | Студент гр. 21-ИТ-1  Чиникайло А.П. |
| Проверил | Преподаватель  Васильева Д.М. |
| Полоцк, 2023г. | |

**Цель работы:** ознакомиться с понятием API (Application Programming Interface), понять, как оно используется, и предоставить примеры реализации API. Также, предоставить задание и контрольные вопросы для проверки понимания темы.

**Ход работы**

**Задание:**

1. Проектирование API

Напишите спецификацию RESTful API для простого блокнот-приложения, включая эндпоинты для создания, чтения, обновления и удаления заметок.

1. Использование API

Используя публичное API (например, JSONPlaceholder), создайте клиентское приложение, которое получает и отображает список пользователей. Bспользуйте библиотеку для HTTP-запросов (например, requests для Python).

1. Разработка RESTful API

Создать RESTful API, которое позволит управлять товарами в интернет-магазине. API должно предоставлять функциональность для создания, чтения, обновления и удаления товаров. Каждый товар должен иметь следующие характеристики: название, цена, описание и уникальный идентификатор(ID).

1. Интеграция с внешним API (ПО ВАРИАНТУ)

**Вариант 1**

Интегрируйтесь с OpenWeather API для создания приложения для получения и отображения погоды.

1. Аутентификация и авторизация

Расширьте функциональность вашего API(Задание 4), добавив аутентификацию и авторизацию пользователей. Реализуйте систему пользовательских аккаунтов и токенов доступа.

1. Реализация спецификации RESTful API для простого блокнот-приложения представлена в листинге 1.

Листинг 1 – Спецификации RESTful API для простого блокнот-приложения.

openapi: 3.0.0

info:

title: Блокнот-приложение

version: 1.0.0

servers:

- url: http://127.0.0.1:5000/notepad

paths:

/notes:

get:

summary: Получить список всех заметок

responses:

'200':

description: Успешный запрос

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/Note'

post:

summary: Создать новую заметку

requestBody:

required: true

content:

Продолжение листинга 1

application/json:

schema:

$ref: '#/components/schemas/Note'

responses:

'201':

description: Заметка успешно создана

content:

application/json:

schema:

$ref: '#/components/schemas/Note'

/notes/{id}:

get:

summary: Получить информацию о заметке по идентификатору

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

'200':

description: Успешный запрос

content:

application/json:

schema:

$ref: '#/components/schemas/Note'

put:

summary: Обновить информацию о заметке

parameters:

- name: id

in: path

required: true

schema:

type: string

requestBody:

required: true

content:

application/json:

schema:

$ref: '#/components/schemas/Note'

responses:

'200':

description: Заметка успешно обновлена

content:

application/json:

schema:

$ref: '#/components/schemas/Note'

delete:

summary: Удалить заметку

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

Продолжение листинга 1

'204':

description: Заметка успешно удалена

components:

schemas:

Note:

type: object

properties:

id:

type: string

title:

type: string

content:

type: string

required:

- title

- content

1. При выполнении данного задания было создано клиентское приложение, используя язык программирования Python, публичное API JSONPlaceholder и библиотеку для HTTP-запросов request, которое получает и отображает список пользователей. Реализация данного приложения представлена в листинге 2.

Листинг 2 – Реализация клиентского приложения, которое получает и отображает список пользователей.

import request

def get\_users():

response =

requests.get('https://jsonplaceholder.typicode.com/users')

if response.status\_code == 200:

users = response.json()

return users

else:

print('Ошибка при получении списка пользователей')

return []

users = get\_users()

for user in users:

print(f'Имя пользователя: {user["name"]}, Email: {user["email"]}')

Работа программы получения и отображения списка пользователей представлена рисунке 1.

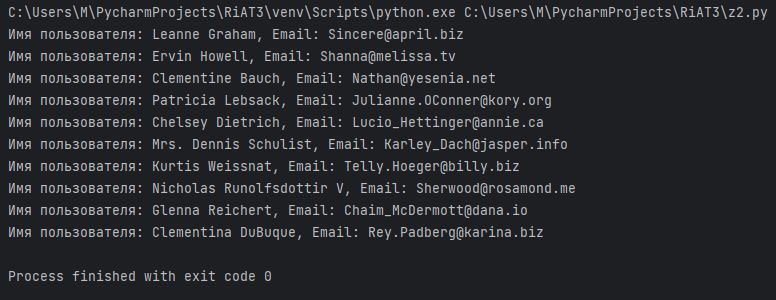


Рисунок 1 – Работа программы получения и отображения списка пользователей

1. При выполнении данного задания было реализовано RESTful API, которое позволит управлять товарами в интернет-магазине. Данное API было создано на языке программирования Python при помощи библиотеки Flask. Реализация данного приложения представлена в листинге 3.

Листинг 3 – Реализация клиентского приложения, которое позволит управлять товарами в интернет-магазине.

from flask import Flask, jsonify, request

app = Flask(\_\_name\_\_)

products = [

{"id": 1, "name": "Товар 1", "price": 1000, "description": "Описание

товара 1"},

{"id": 2, "name": "Товар 2", "price": 2000, "description": "Описание

товара 2"}

]

@app.route('/products', methods=['GET'])

def get\_products():

return jsonify(products)

@app.route('/products/<int:product\_id>', methods=['GET'])

def get\_product(product\_id):

for product in products:

if product['id'] == product\_id:

return jsonify(product)

return jsonify({"message": "Товар не найден"}), 404

@app.route('/products', methods=['POST'])

def create\_product():

new\_product = {

"id": len(products) + 1,

"name": request.json['name'],

"price": request.json['price'],

"description": request.json['description']

}

products.append(new\_product)

return jsonify(new\_product), 201

@app.route('/products/<int:product\_id>', methods=['PUT'])

def update\_product(product\_id):

for product in products:

if product['id'] == product\_id:

product['name'] = request.json['name']

product['price'] = request.json['price']

product['description'] = request.json['description']

return jsonify(product)

return jsonify({"message": "Товар не найден"}), 404

@app.route('/products/<int:product\_id>', methods=['DELETE'])

def delete\_product(product\_id):

for product in products:

if product['id'] == product\_id:

products.remove(product)

return jsonify({"message": "Товар удален"})

return jsonify({"message": "Товар не найден"}), 404

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

Работа программы, а именно запросы создания, чтения, обновления и удаления товаров были проверены при помощи HTTP-клиента для тестирования API Postman. Все запросы и ответы на них представлены на рисунках 2-8.

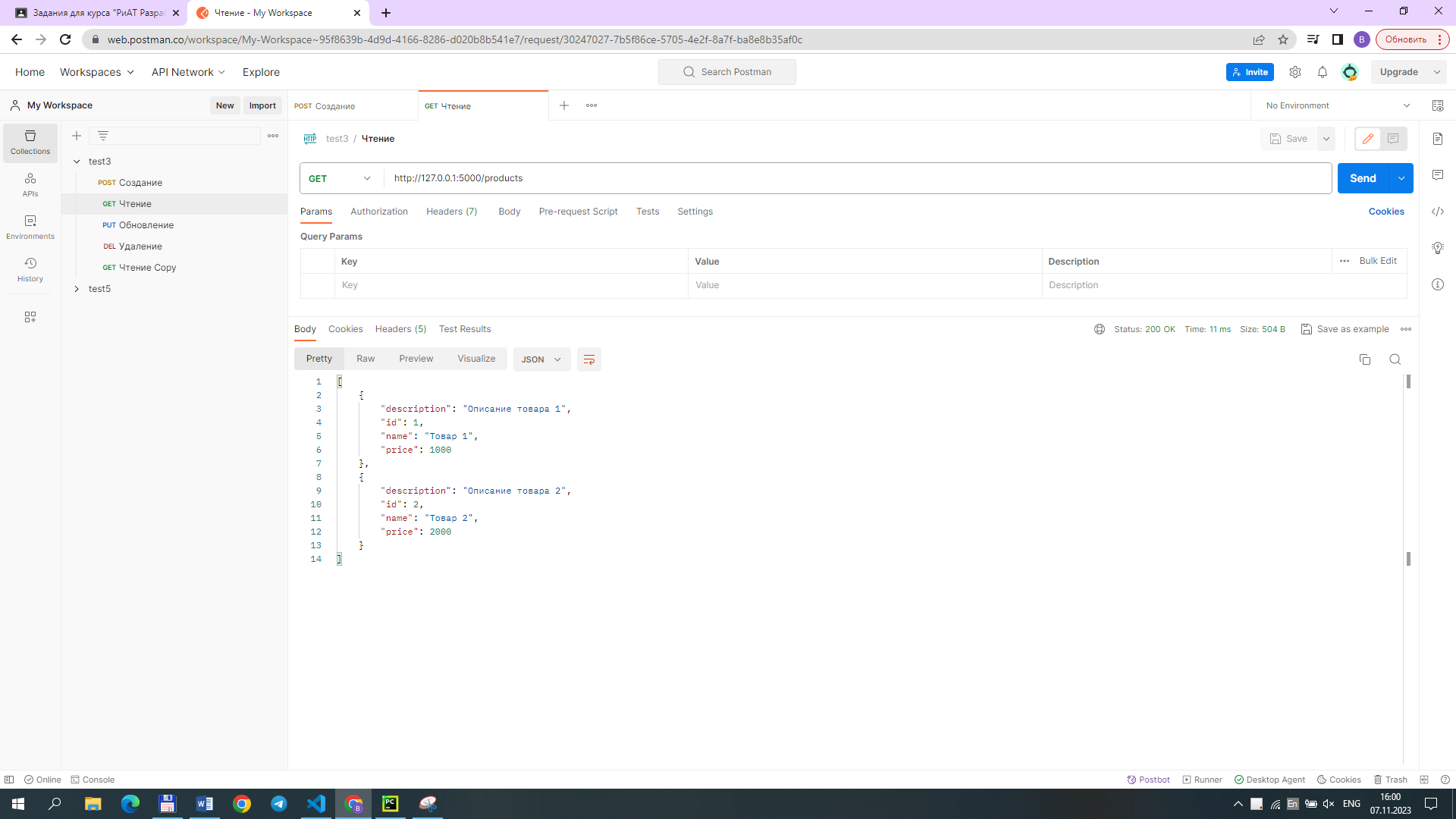


Рисунок 2 – GET-запрос на чтение всех товаров

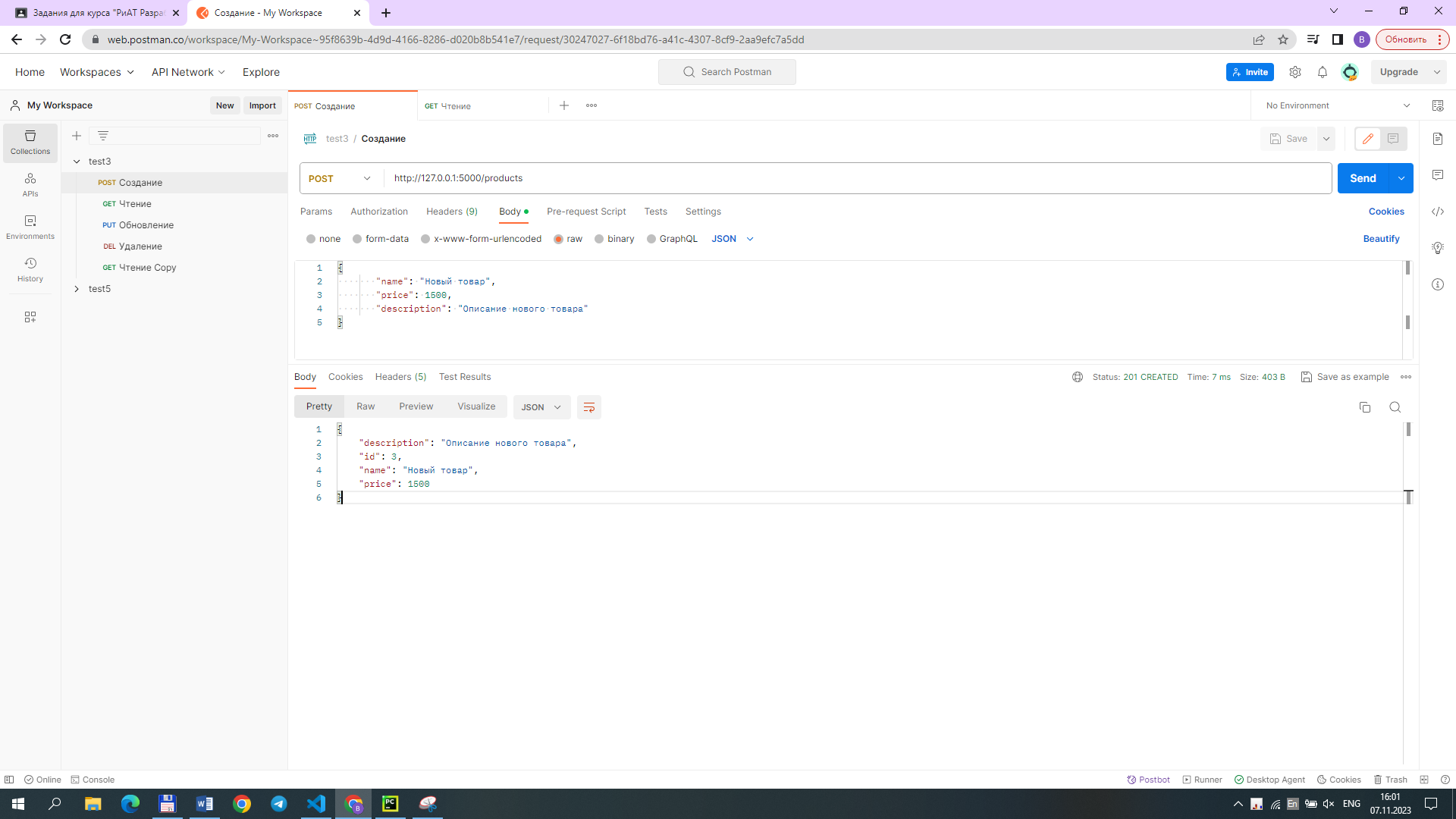


Рисунок 3 – POST-запрос на создание товара

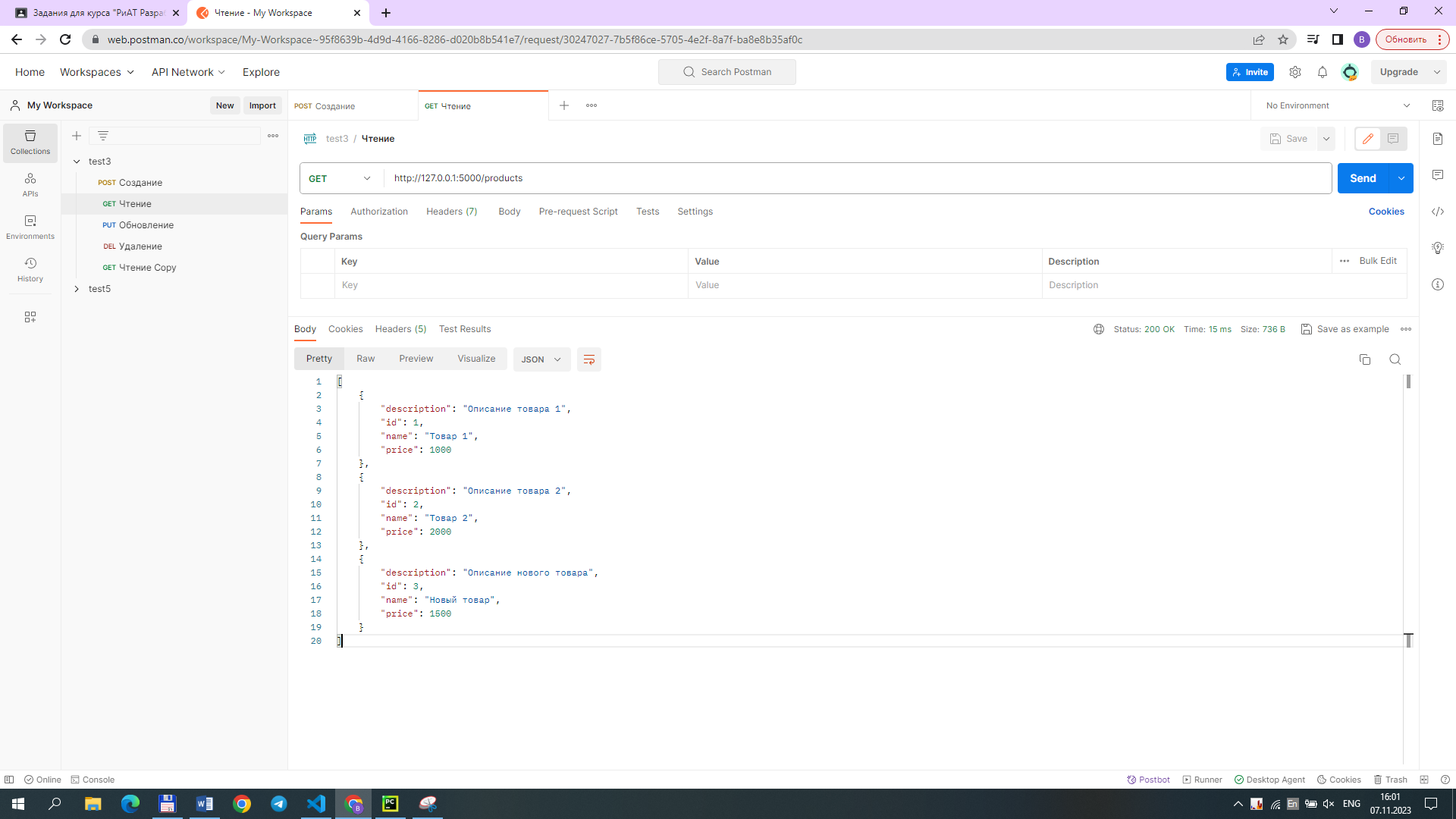


Рисунок 4 – GET-запрос на чтение всех товаров после добавления

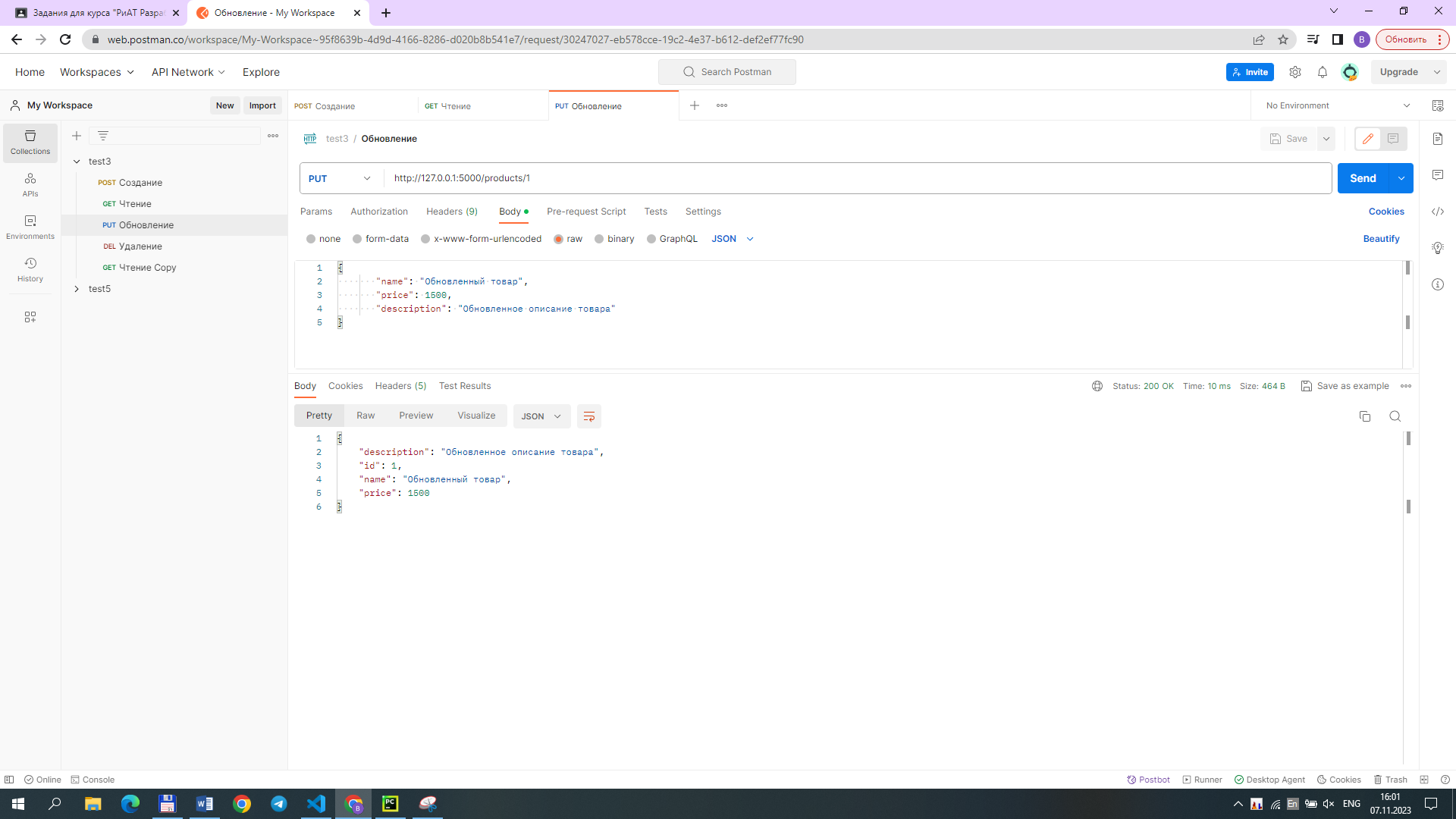


Рисунок 5 – PUT-запрос на обновление товара

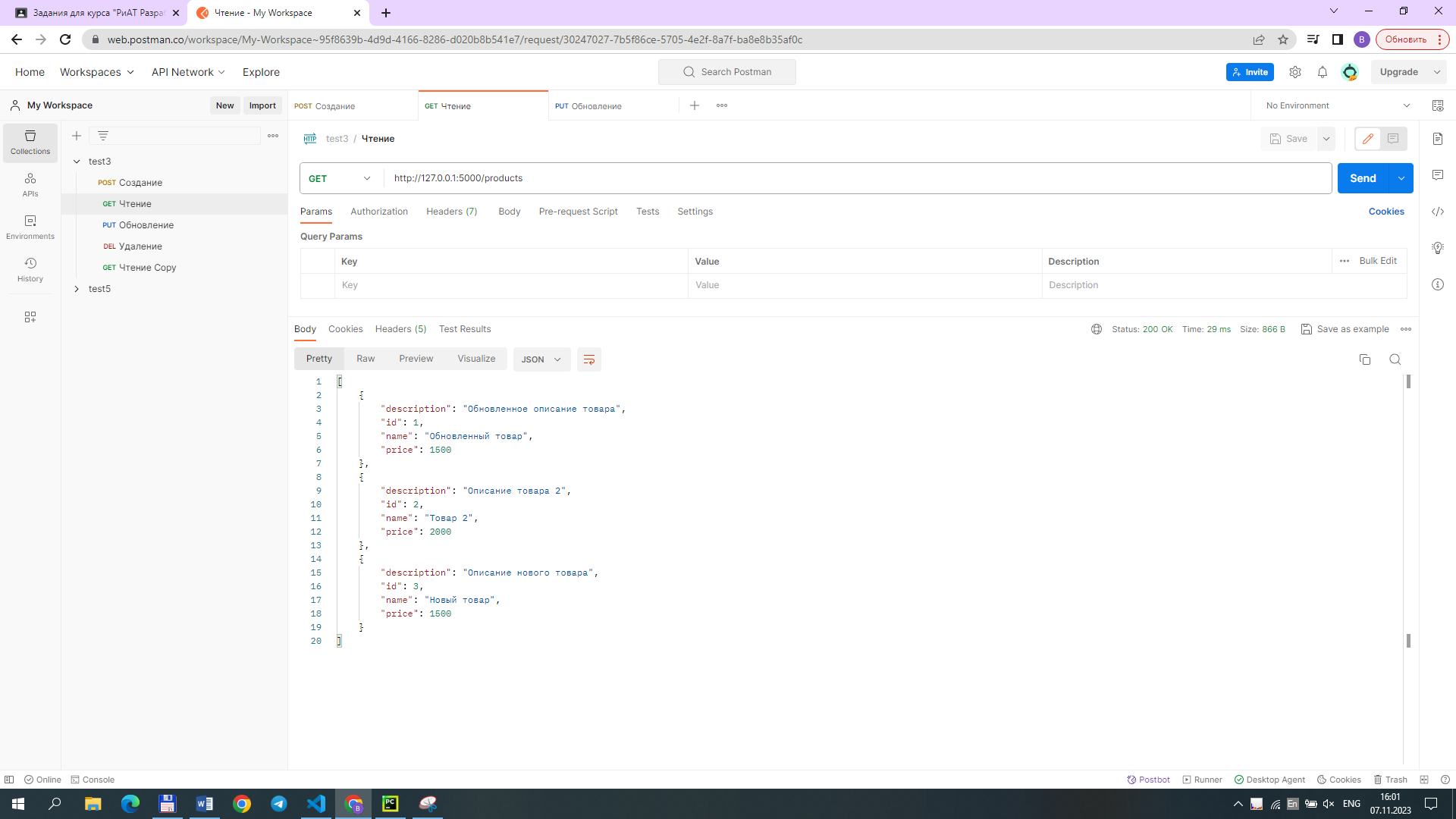


Рисунок 6 – GET-запрос на чтение всех товаров после обновления

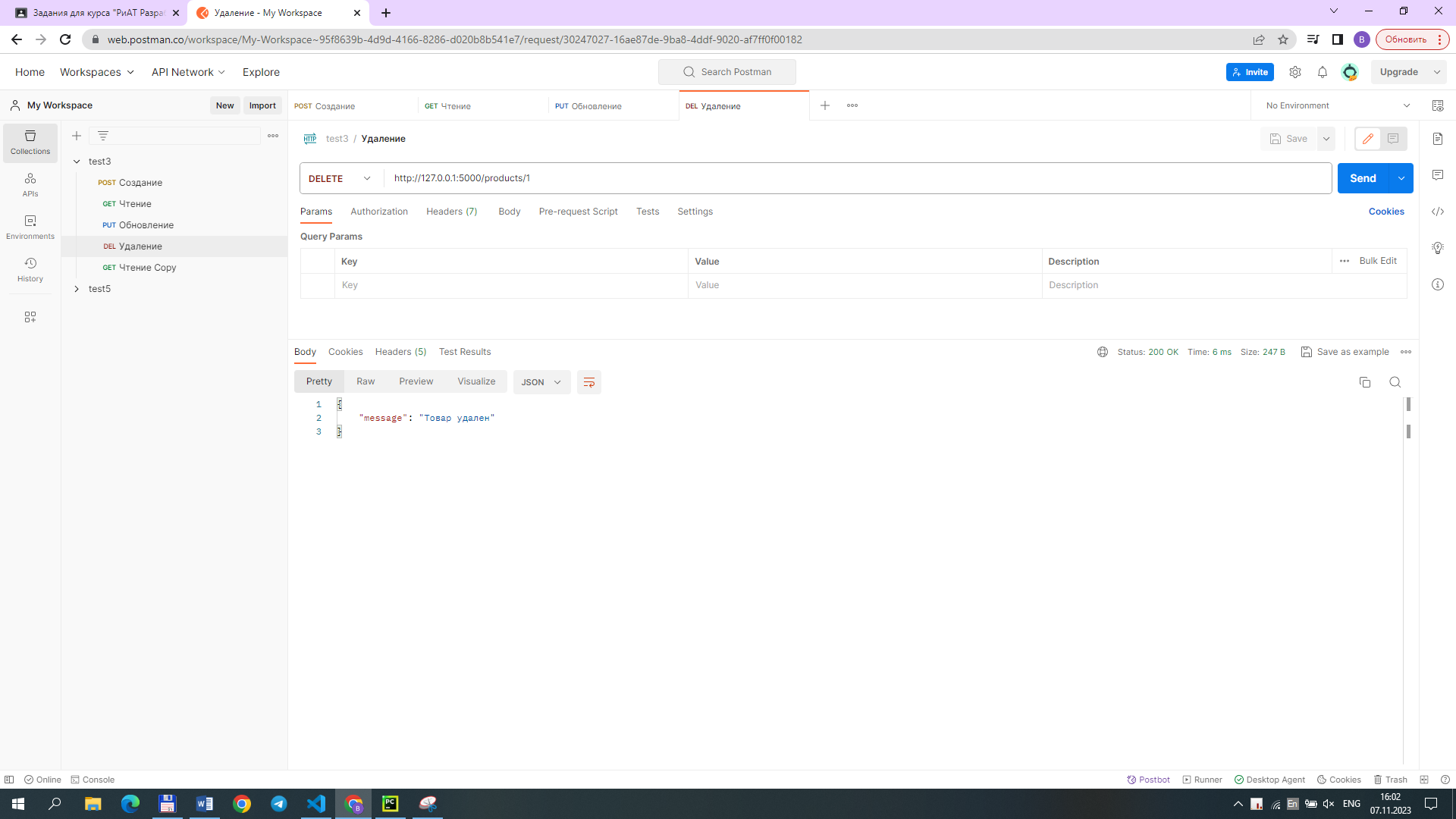


Рисунок 7 – DELETE-запрос на удаление товара

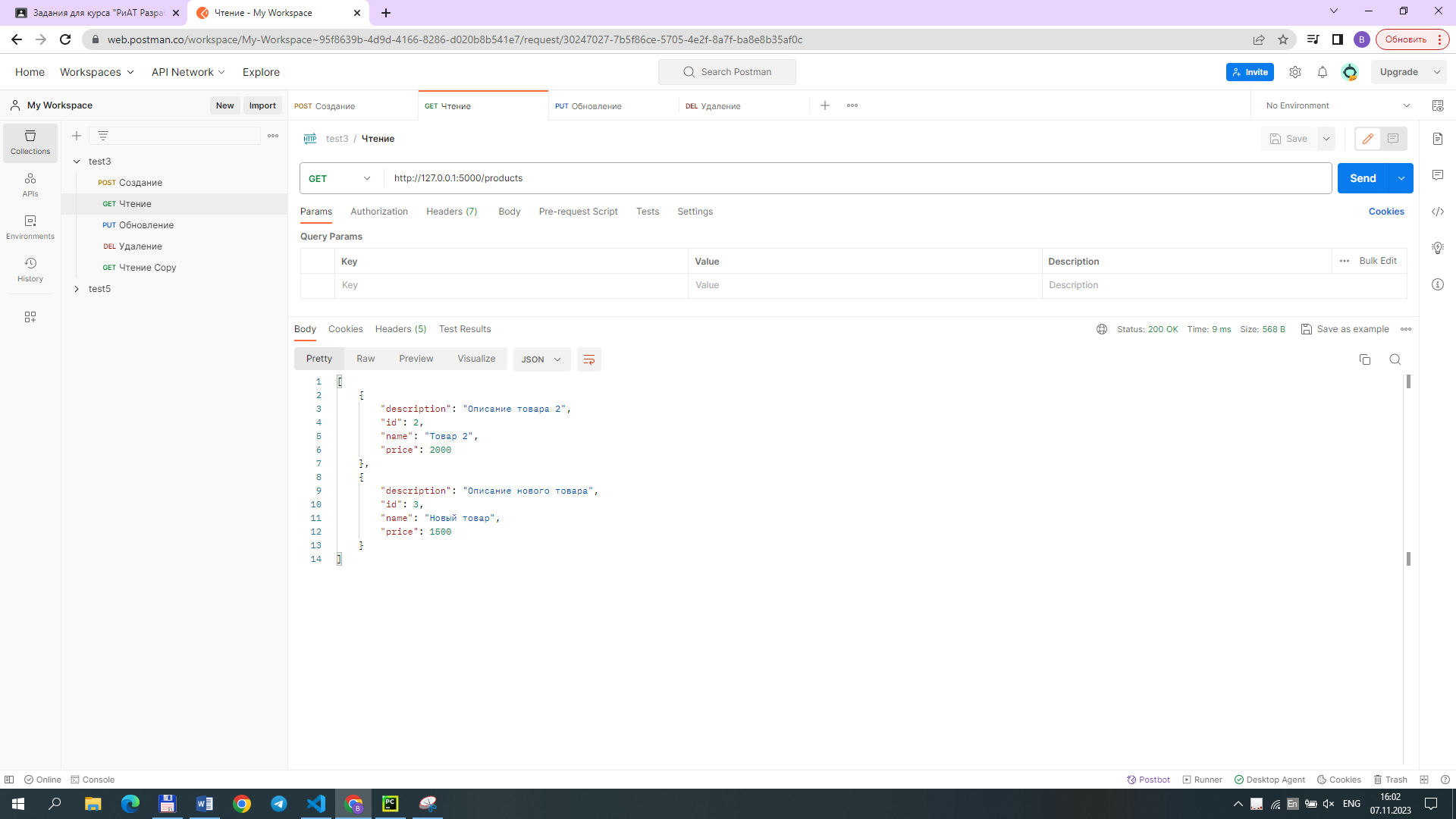


Рисунок 8 – GET-запрос на чтение всех товаров после удаления

1. При выполнении данного задания была произведена интеграция с News API для создания приложения для получения и отображения последних новостей. Реализация данного приложения представлена в листинге 4.

Листинг 4 – Реализация клиентского приложения для получения и отображения последних новостей.

import requests  
  
api\_key = '50bdc11c753481547c61352a79e3d757'  
city = 'Navapolatsk'  
url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}'  
response = requests.get(url)

Продолжение листинга 4

if response.status\_code == 200:  
 data = response.json()  
 print(f'Погода в городе {city}:')  
 print(f'Температура: {data["main"]["temp"]}K')  
 print(f'Описание: {data["weather"][0]["description"]}')  
else:  
 print('Ошибка при получении данных о погоде')

Работа программы получения и отображения погоды Новополоцка рисунке 9.

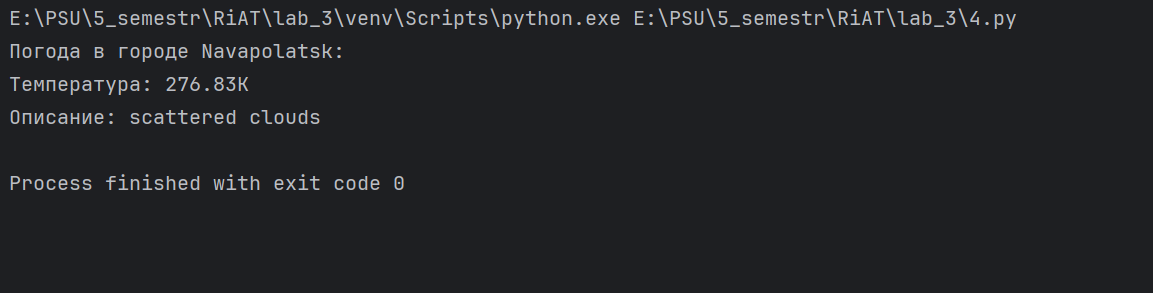


Рисунок 9 – Работа программы получения и отображения погоды Новополоцка

1. При выполнении данного задания к ранее созданному API были добавлены аутентификацию и авторизацию пользователей. Реализация данного приложения представлена в листинге 5.

Листинг 5 – Реализация клиентского приложения для получения и отображения последних новостей.

import requests  
from flask import Flask, request, jsonify  
from flask\_httpauth import HTTPBasicAuth  
  
app = Flask(\_\_name\_\_)  
auth = HTTPBasicAuth()  
users = {  
 "user1": "123",  
 "user2": "321"  
}  
@auth.verify\_password  
def verify\_password(username, password):  
 if not username or not password:  
 return False  
 if username in users and users[username] == password:  
 return username  
 return False  
  
@app.route('/weather', methods=['GET'])  
@auth.login\_required  
def get\_weather():  
 city = 'Navapolatsk'  
 api\_key = '50bdc11c753481547c61352a79e3d757'  
 url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}'  
 response = requests.get(url)

Продолжение листинга 5

data = response.json()  
 return jsonify(data)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run()

Работа обновленной программы получения и отображения погоды Новополоцка представлена рисунках 10-11. На рисунке 10 отображен ответ на запрос авторизованного пользователя. На рисунке 11 – неавторизованного.

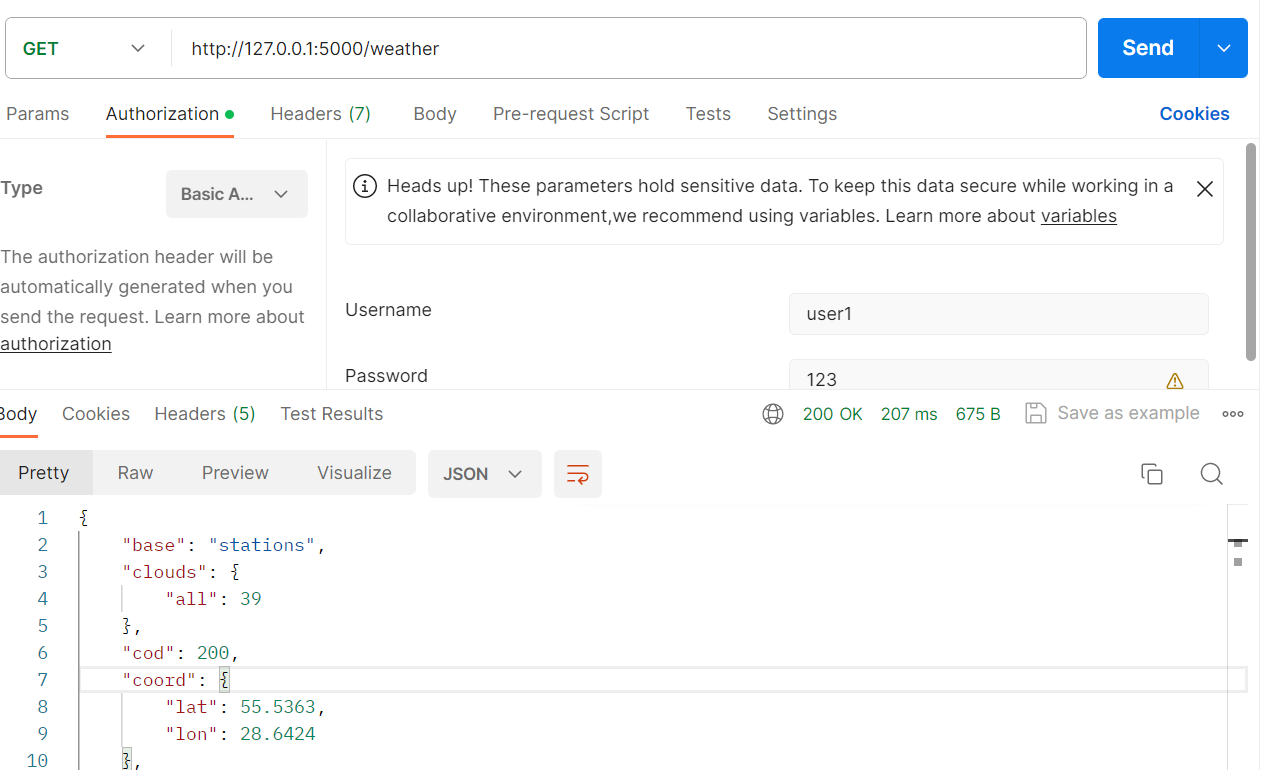


Рисунок 10 – Получение погоды Новополоцка авторизованным пользователем

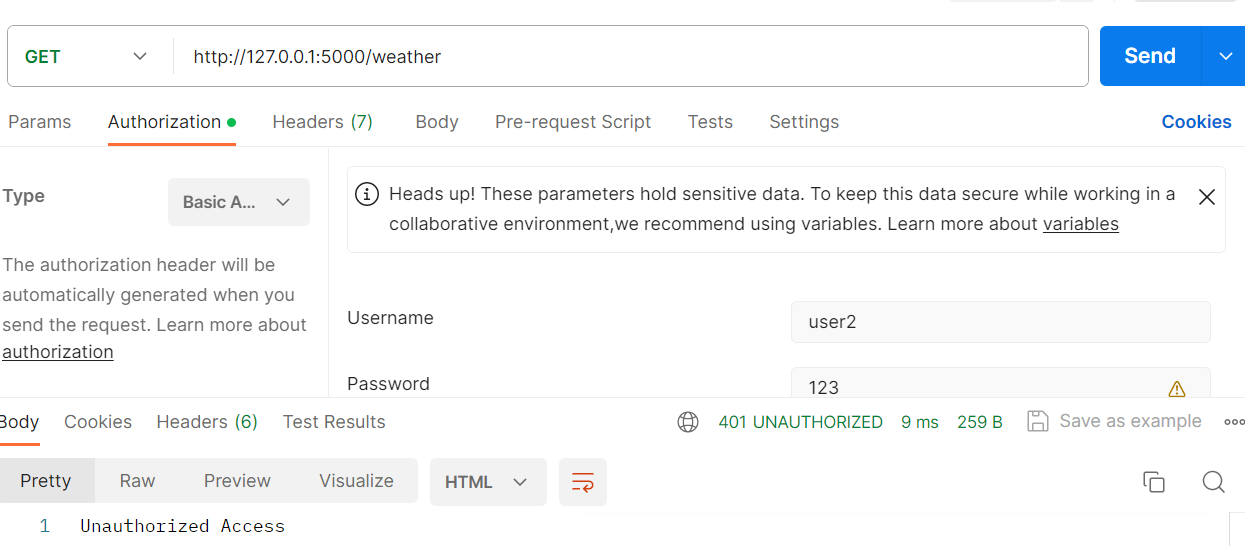


Рисунок 11 – Получение погоды Новополоцка неавторизованным пользователем

1. **Что такое API?**

API – это набор методов, функций и данных, который позволяет разработчикам взаимодействовать с определенным программным продуктом, приложением или сервисом.

1. **Какую роль играют API в разработке ПО?**

* Абстракция: API скрывает сложность реализации определенной функциональности, предоставляя простой интерфейс для использования. Это позволяет разработчикам сосредотачиваться на более высокоуровневых задачах.
* Совместимость: API определяет стандарты и протоколы, что облегчает интеграцию различных компонентов в одно целое.
* Расширяемость: API может быть расширено и улучшено без изменения самого приложения. Это позволяет сделать приложение более гибким и поддающимся развитию.

1. **Где используются API?**

API используется во множестве областей разработки программного обеспечения:

* веб-сервисы;
* библиотеки и фреймворки;
* операционные системы;
* интернет вещей (IoT).

1. **Какие примеры популярных типов API вы можете назвать?**

Существует несколько распространенных типов API, каждый из которых ориентирован на определенные задачи:

* HTTP API;
* REST API;
* SOAP API.

1. **Почему API важен в разработке программного обеспечения?**

API важны в разработке программного обеспечения, потому что они обеспечивают модульность, переиспользуемость и расширяемость программного кода. Они позволяют разработчикам использовать функциональность, предоставляемую другими программами или сервисами, без необходимости писать весь код с нуля. Это сокращает время разработки, повышает эффективность и упрощает интеграцию различных компонентов программного обеспечения.

1. **Какую роль играют API в интеграции приложений?**

API играет роль моста между разными приложениями и сервисами, позволяя им взаимодействовать, обмениваться данными и совместно решать задачи.

1. **Какие принципы проектирования RESTful API?**

* всё представляется как ресурсы;
* для операция с ресурсами используются HTTP методы;
* ресурсы могут иметь разное представление, такие как JSON, XML, и др;
* не хранят состояние клиента на сервере между запросами;
* позволяет устанавливать связи между ресурсами.

1. **Что такое спецификация API? Какие инструменты используются для её создания?**

Спецификация API - это документация, описывающая структуру и поведение API.

1. **Какие языки программирования могут использоваться для реализации API?**

Для реализации API можно использовать различные языки программирования, такие как Python, Java, JavaScript, Ruby, C#, PHP и другие.

1. **Какие фреймворки широко используются для разработки API?**

Некоторые широко используемые фреймворки для разработки API включают Flask и Django для Python, Node.js для JavaScript, Ruby on Rails для Ruby, Spring для Java и Laravel для PHP.

**Вывод:** в ходе выполнения лабораторной работы были изучены основы работы с API, принципы проектирования RESTful API, интеграция с внешними API, а также вопросы аутентификации и авторизации в API. Эти навыки и знания являются важными для разработки программного обеспечения, особенно при работе с внешними сервисами и создании клиент-серверных приложений.