МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

Московский авиационный институт

«Национальный исследовательский университет»

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304: «Вычислительные машины, системы и сети»

Отчёт по лабораторной работе №21 «Сервис с Rest API»

по учебной дисциплине «Web-технологии»

Выполнил:

студент группы М3О-107БВ-24

Коркина К.В.

Приняла: Масленикова Т.Ю.

Титов Ю.П.

Москва 2025

1. Создать простой REST API сервис для управления списком задач (To-Do list). Реализовать методы для создания, чтения, обновления и удаления задач.

from flask import Flask, request, jsonify, Response

import os

import json

import jwt

import datetime

from functools import wraps

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

**# Конфигурация**

app.config["SECRET\_KEY"] = "your\_secret\_key\_here"

app.config["JWT\_ALGORITHM"] = "HS256"

**# Файлы данных**

TASKS\_FILE = 'tasks.json'

USERS\_FILE = 'users.json'

def read\_json(file\_path):

if not os.path.exists(file\_path):

return []

try:

with open(file\_path, 'r', encoding='utf-8') as f:

return json.load(f)

except (json.JSONDecodeError, FileNotFoundError):

return []

def write\_json(data, file\_path):

with open(file\_path, 'w', encoding='utf-8') as f:

json.dump(data, f, ensure\_ascii=False, indent=2)

def update\_task\_by\_id(task\_id, update\_data):

tasks = read\_json(TASKS\_FILE)

for task in tasks:

if task["id"] == task\_id:

task.update(update\_data)

write\_json(tasks, TASKS\_FILE)

return True

return False

def generate\_token(user\_id):

payload = {

"sub": str(user\_id),

"exp": datetime.datetime.utcnow() + datetime.timedelta(days=30)

}

return jwt.encode(payload, app.config["SECRET\_KEY"], algorithm=app.config["JWT\_ALGORITHM"])

def token\_required(f):

@wraps(f)

def decorated(\*args, \*\*kwargs):

data = request.get\_json()

token = data.get('token')

if not token:

return jsonify({"error": "Token is missing"}), 401

try:

data = jwt.decode(token, app.config["SECRET\_KEY"], algorithms=[app.config["JWT\_ALGORITHM"]])

user\_id = data["sub"]

except:

return jsonify({"error": "Invalid token"}), 401

return f(user\_id, \*args, \*\*kwargs)

return decorated

**#Регистрация нового пользователя**

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

def register():

data = request.get\_json()

users = read\_json(USERS\_FILE)

if not data or 'username' not in data or 'password' not in data:

return jsonify({"error": "Username and password are required"}), 400

if any(user['username'] == data['username'] for user in users):

return jsonify({"error": "Username already exists"}), 400

new\_user = {

"id": len(users) + 1,

"username": data['username'],

"password": data['password']

}

users.append(new\_user)

write\_json(users, USERS\_FILE)

return jsonify({"message": "User registered successfully"}), 201

**#Выдача токена**

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

def login():

data = request.get\_json()

users = read\_json(USERS\_FILE)

if not data or 'username' not in data or 'password' not in data:

return jsonify({"error": "Username and password are required"}), 400

user = next((u for u in users if u['username'] == data['username'] and u['password'] == data['password']), None)

if not user:

return jsonify({"error": "Invalid credentials"}), 401

token = generate\_token(user['id'])

return jsonify({"token": token})

**#GET-запрос на все задачи**

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

@token\_required

def get\_all\_tasks(user\_id):

tasks = read\_json(TASKS\_FILE)

return jsonify(tasks)

**#GET-запрос на задачу по её id**

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

@token\_required

def get\_task(user\_id, task\_id):

tasks = read\_json(TASKS\_FILE)

task = next((t for t in tasks if t['id'] == task\_id and t.get('user\_id') == user\_id), None)

if task:

return Response(

json.dumps(task, ensure\_ascii=False),

mimetype='application/json'

)

return jsonify({"error": "Task not found"}), 404

**#Создать новую задачу**

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

@token\_required

def create\_task(user\_id):

tasks = read\_json(TASKS\_FILE)

if not request.json or 'title' not in request.json:

return jsonify({"error": "Title is required"}), 400

max\_id = max(task['id'] for task in tasks) if tasks else 0

new\_task = {

"id": max\_id + 1,

"user\_id": user\_id,

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

"done": request.json.get('done', False)

}

tasks.append(new\_task)

write\_json(tasks, TASKS\_FILE)

return jsonify(new\_task), 201

**#Обновить задачу по её id**

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

@token\_required

def update\_task(user\_id, task\_id):

tasks = read\_json(TASKS\_FILE)

task = next((t for t in tasks if t['id'] == task\_id and t.get('user\_id') == user\_id), None)

if not task:

return jsonify({"error": "Task not found"}), 404

if not request.json:

return jsonify({"error": "No data provided"}), 400

task['title'] = request.json.get('title', task['title'])

task['done'] = request.json.get('done', task['done'])

write\_json(tasks, TASKS\_FILE)

return jsonify(task)

**#Удалить задачу**

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

@token\_required

def delete\_task(user\_id, task\_id):

tasks = read\_json(TASKS\_FILE)

task = next((t for t in tasks if t['id'] == task\_id and t.get('user\_id') == user\_id), None)

if not task:

return jsonify({"error": "Task not found"}), 404

tasks.remove(task)

write\_json(tasks, TASKS\_FILE)

return jsonify({"result": "Task deleted"})

**# Фильтрация данных по выполнению**

@app.route("/tasks/filtered", methods=["GET"])

@token\_required

def get\_filtered\_task(user\_id):

tasks = read\_json(TASKS\_FILE)

data = []

for task in tasks:

if task["done"]:

data.append(task)

return jsonify(data)

**# Сортировка данных по title**

@app.route("/tasks/sorted", methods=["GET"])

@token\_required

def get\_sorted\_task(user\_id):

tasks = read\_json(TASKS\_FILE)

sorted\_tasks = sorted(tasks, key=lambda x: x["title"].lower())

return jsonify(sorted\_tasks)

**# Экспорт данных в формат JSON**

@app.route("/tasks/export/json", methods=["GET"])

@token\_required

def export\_json(user\_id):

tasks = read\_json(TASKS\_FILE)

return jsonify(tasks)

**# Экспорт данных в формат CSV**

@app.route("/tasks/export/csv", methods=["GET"])

@token\_required

def export\_csv(user\_id):

tasks = read\_json(TASKS\_FILE)

csv\_data = ",".join(tasks[0].keys()) + "\n"

for item in tasks:

csv\_data += ",".join(map(str, item.values())) + "\n"

return csv\_data

**#Добавить комментарий к задаче**

@app.route("/tasks/<int:task\_id>/comment", methods=["PUT"])

@token\_required

def add\_comment(user\_id, task\_id):

tasks = read\_json(TASKS\_FILE)

task = next((t for t in tasks if t["id"] == task\_id), None)

if not task:

return {"error": "Task not found"}, 404

comment\_text = request.json.get("text")

if not comment\_text:

return {"error": "Comment text is required"}, 400

if "comments" not in task:

task["comments"] = []

task["comments"].append({

"text": comment\_text

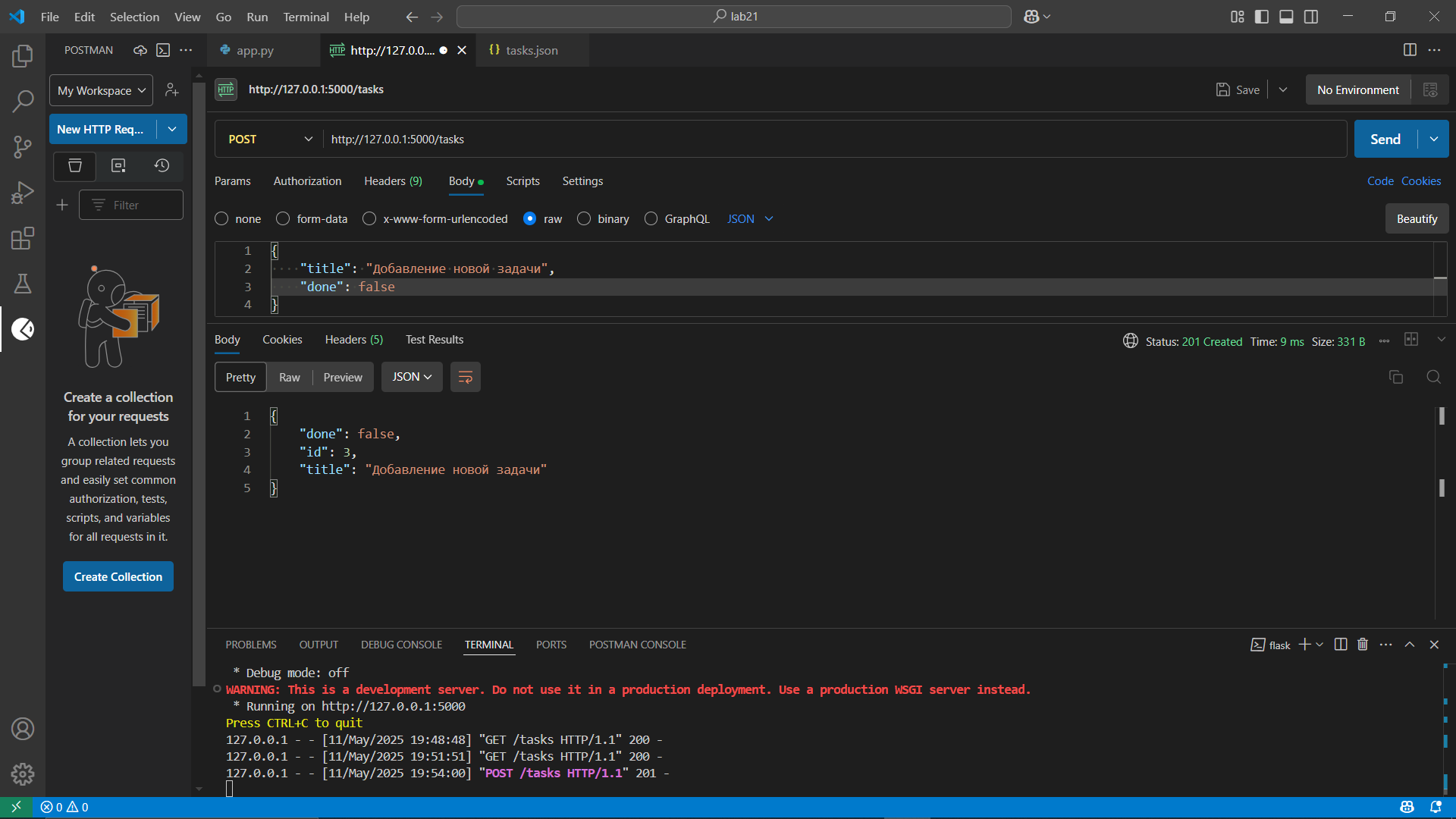
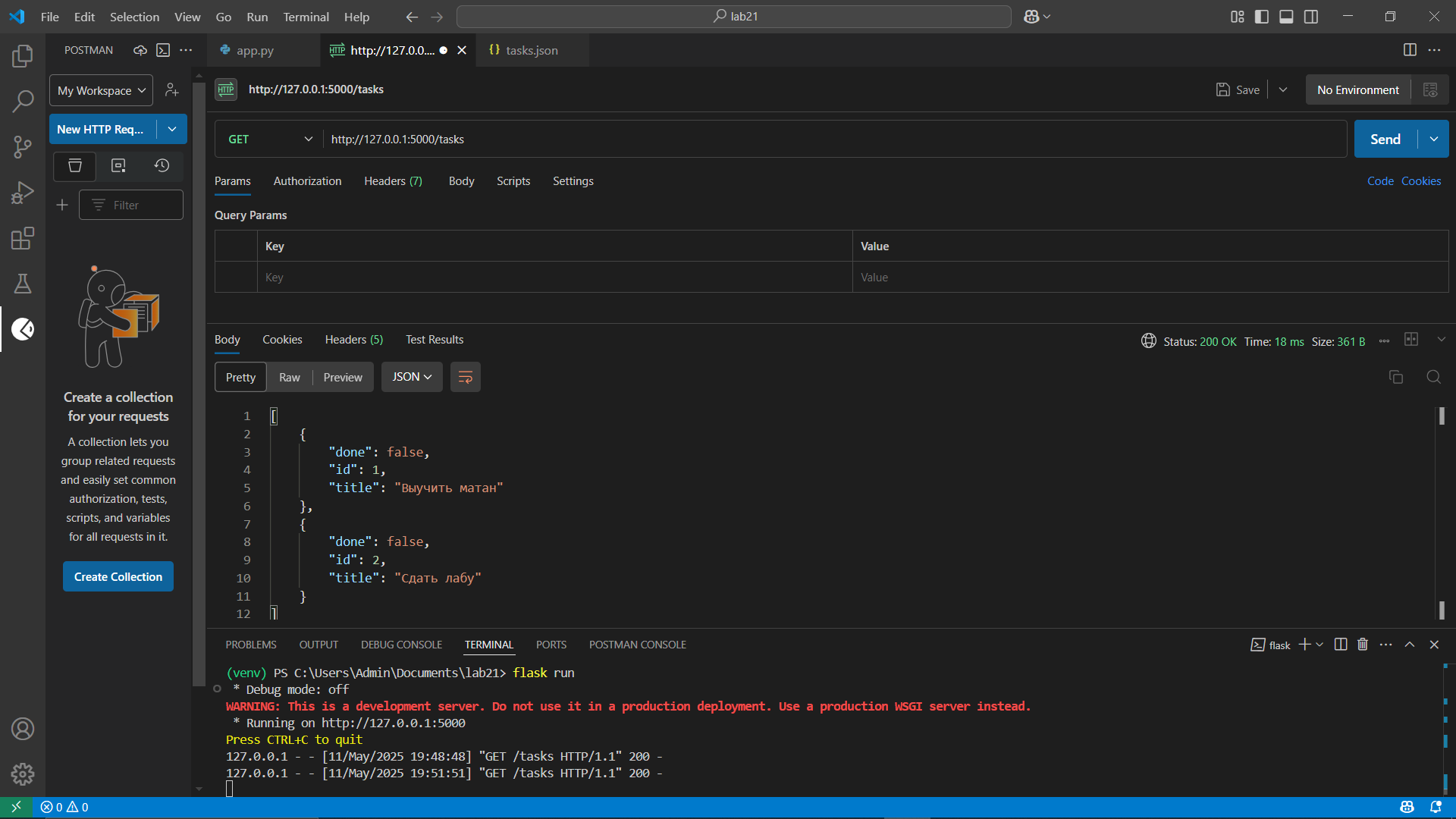
})

write\_json(tasks, TASKS\_FILE)

return {"message": "Comment added successfully"}, 200

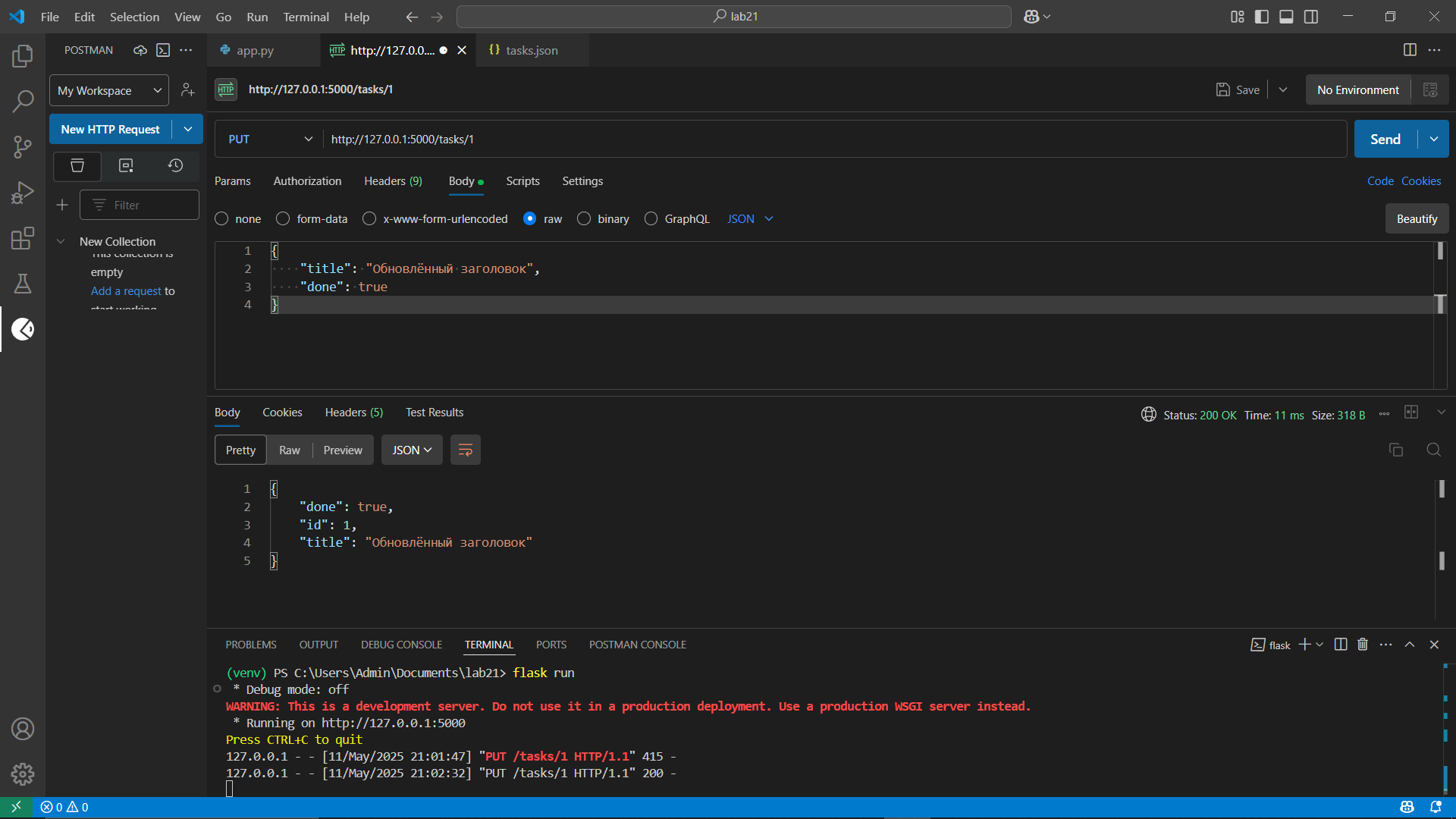
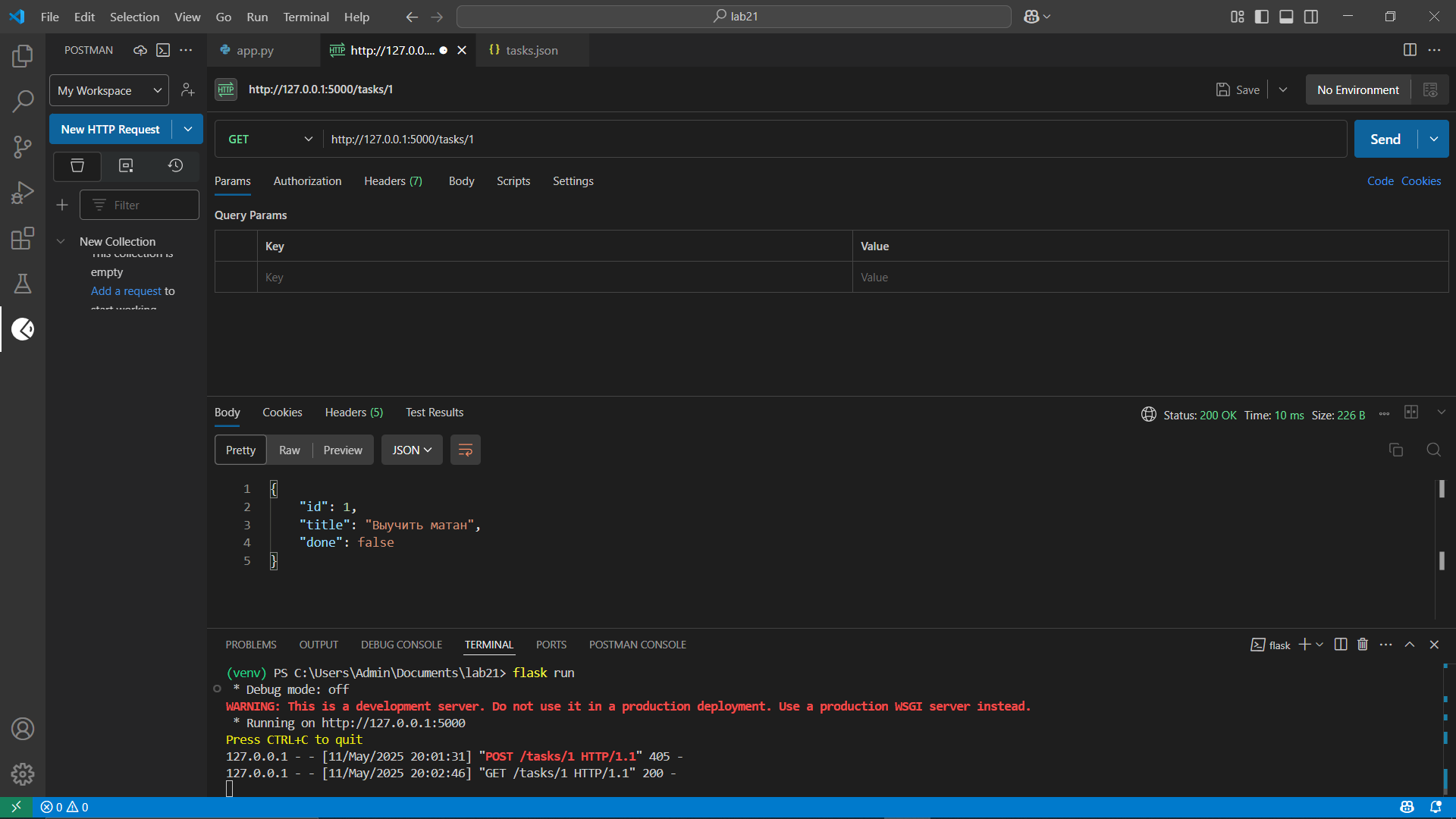
if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)



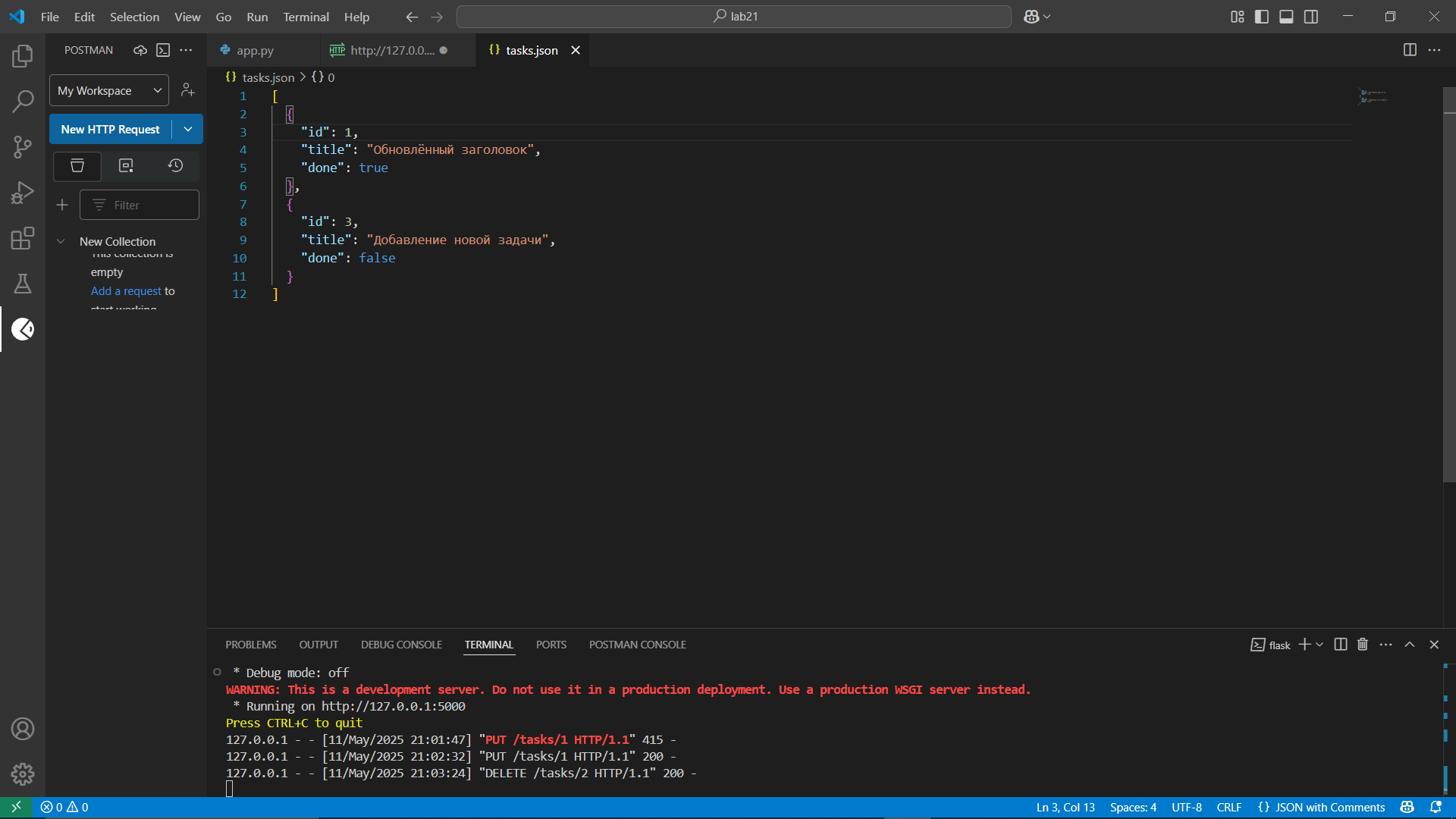
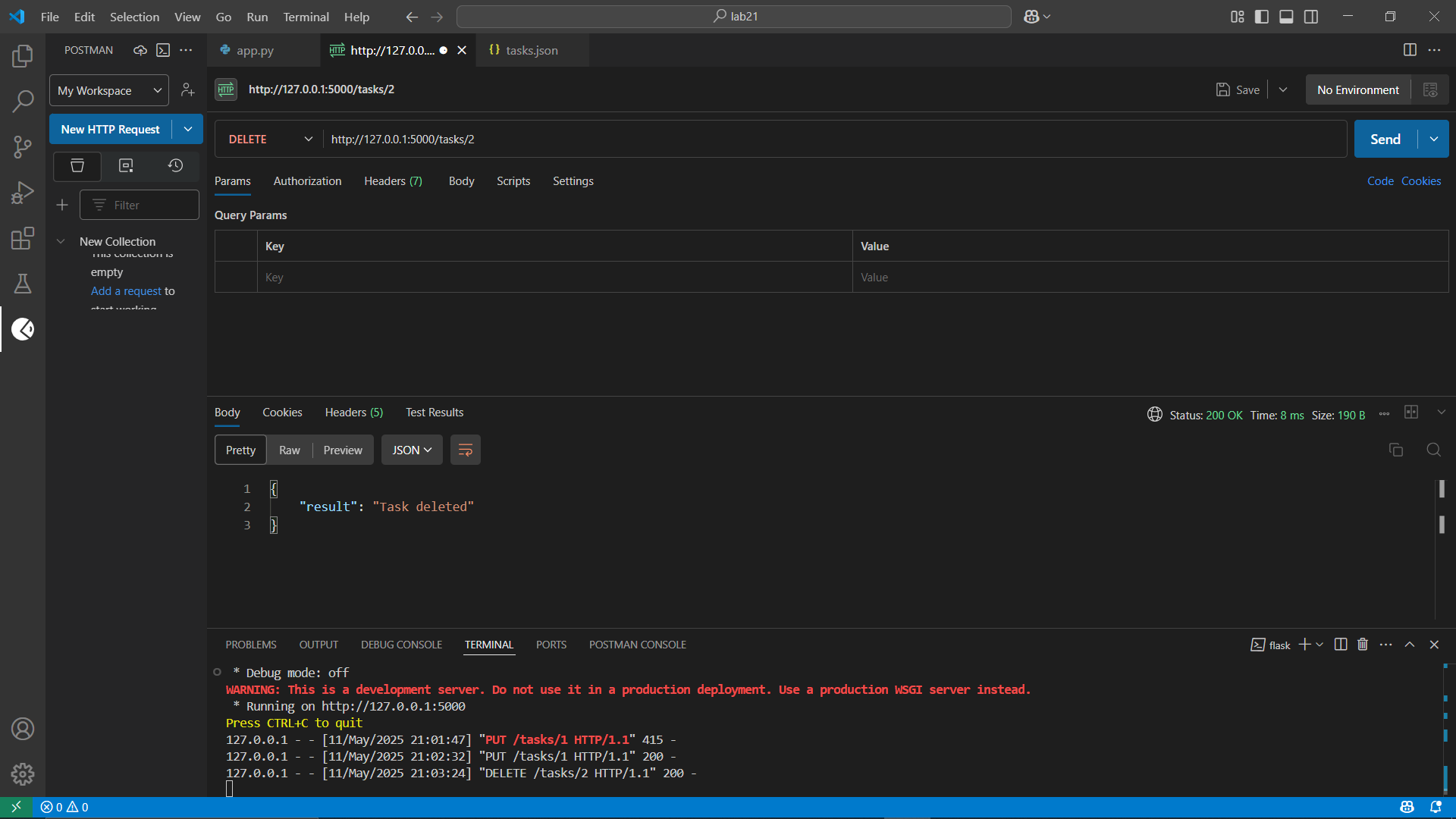
Добавить новую задачу (POST)

Получить все задачи (GET)



Получить задачу по её ID (GET)

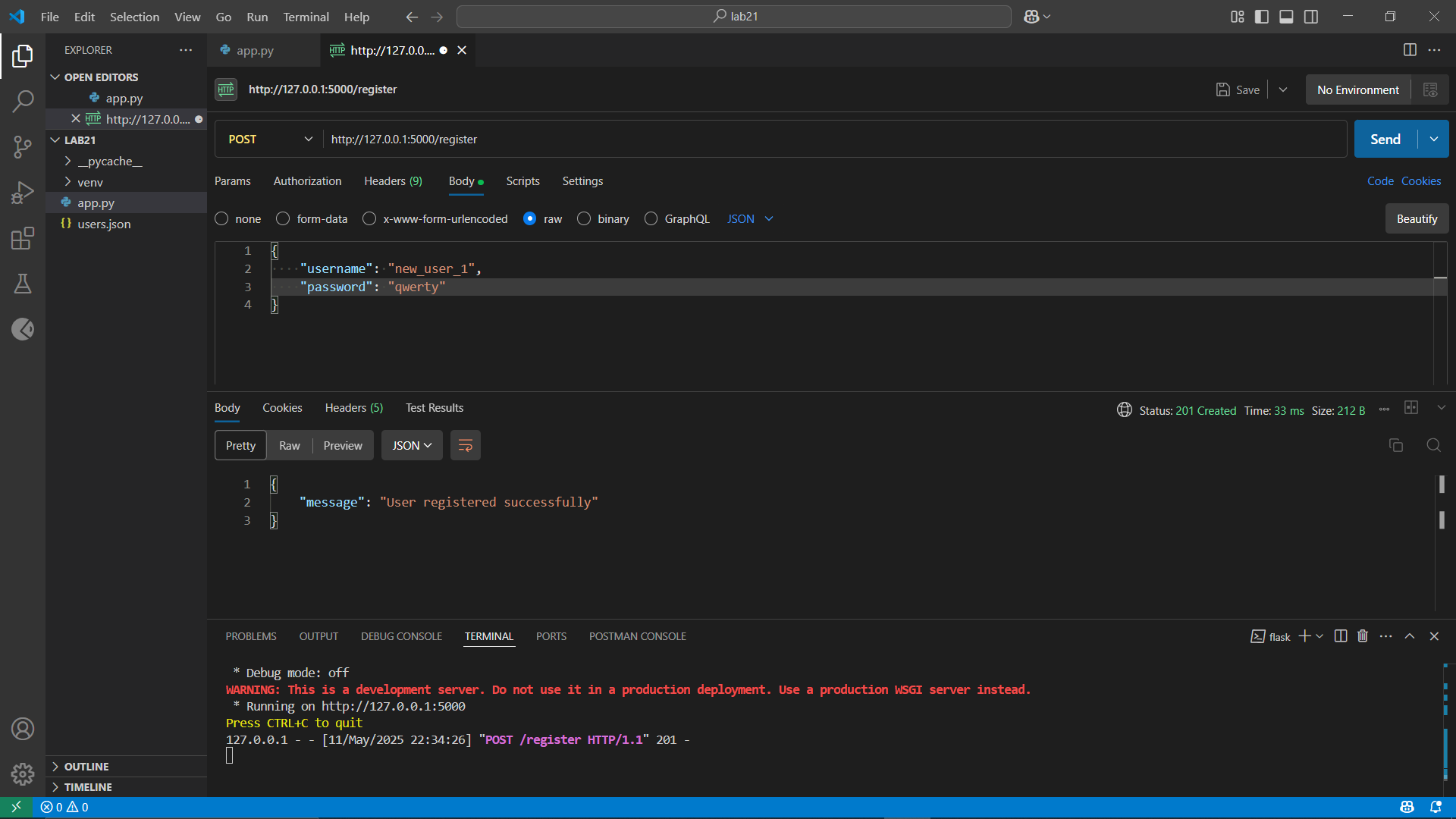
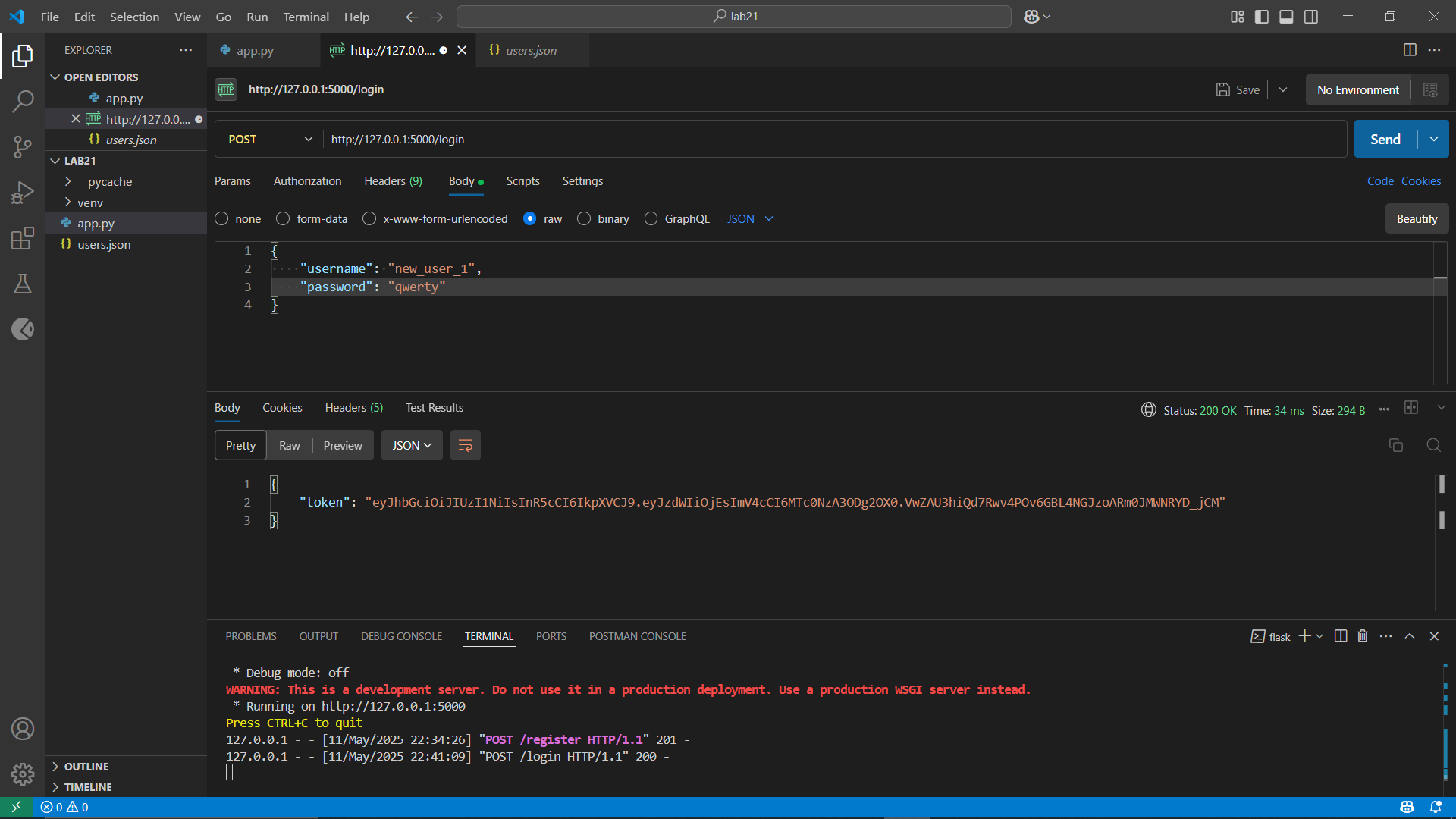
Отредактировать задачу (по id) (PUT)



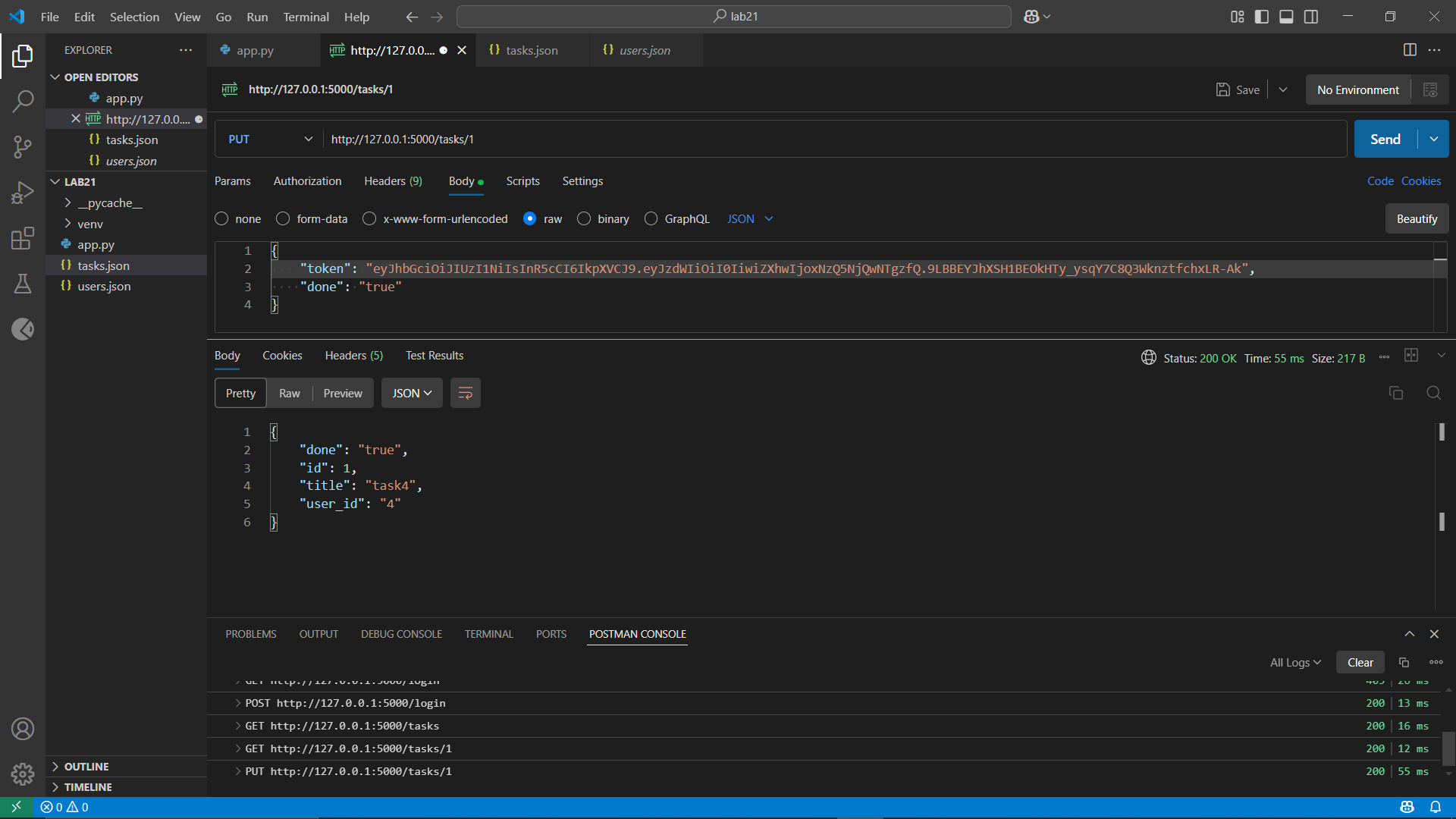
Список задач после удаления одной из них

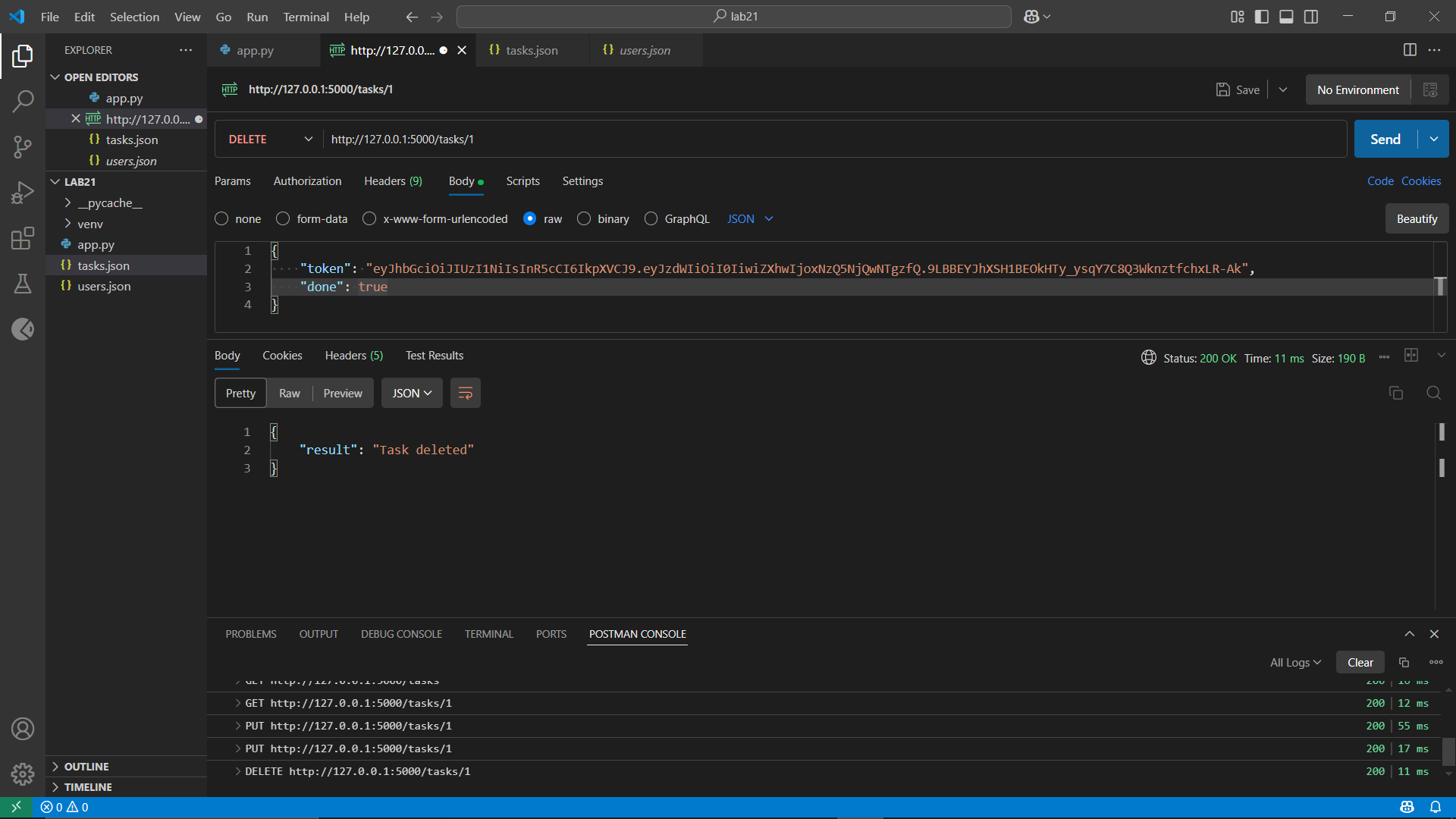
Удалить задачу (DELETE)

1. Добавить функционал аутентификации пользователей в REST API сервисе. Пользователь должен получить токен доступа для выполнения операций с задачами.



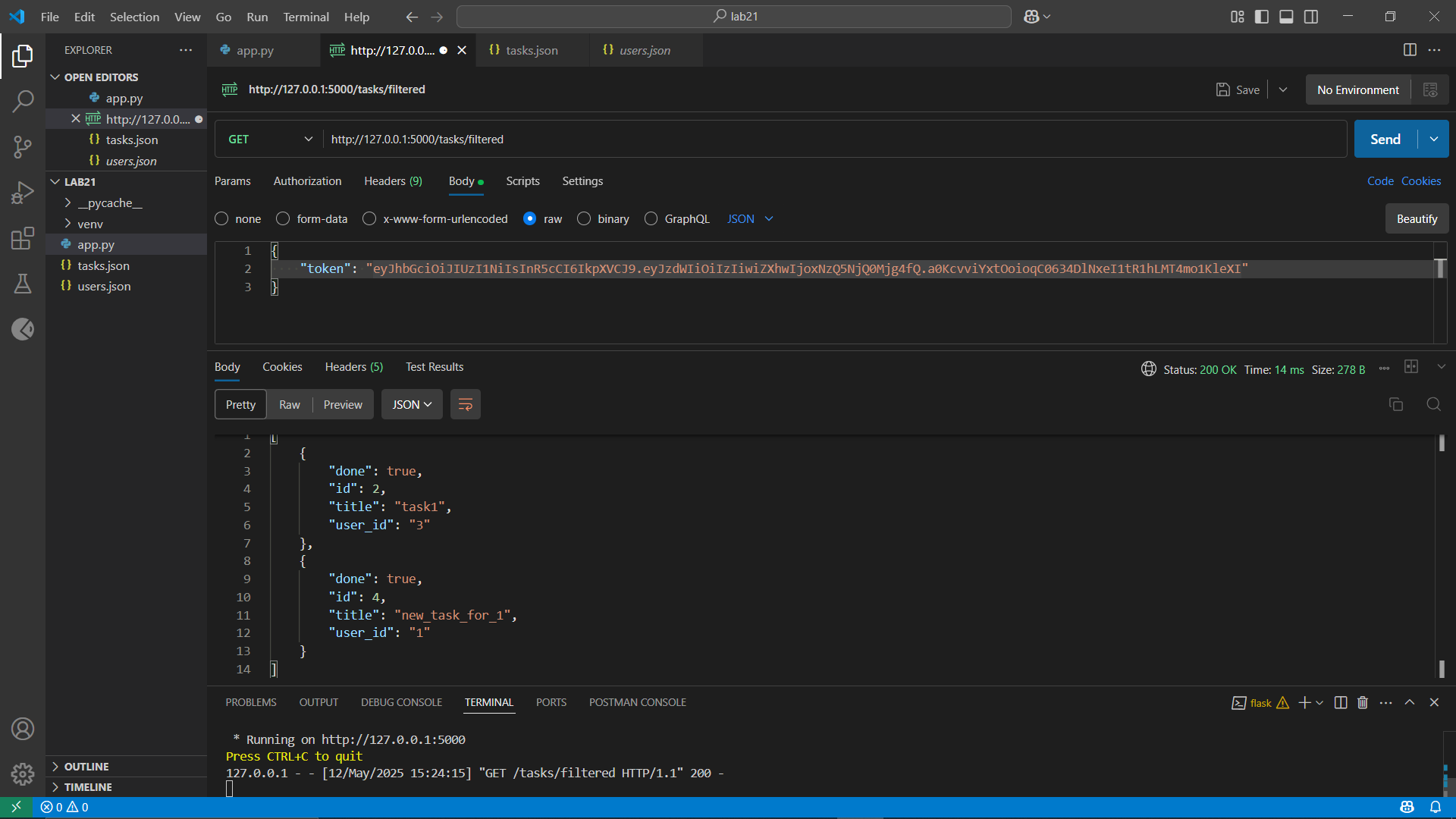
Регистрация пользователя (register) и получение токена (login)



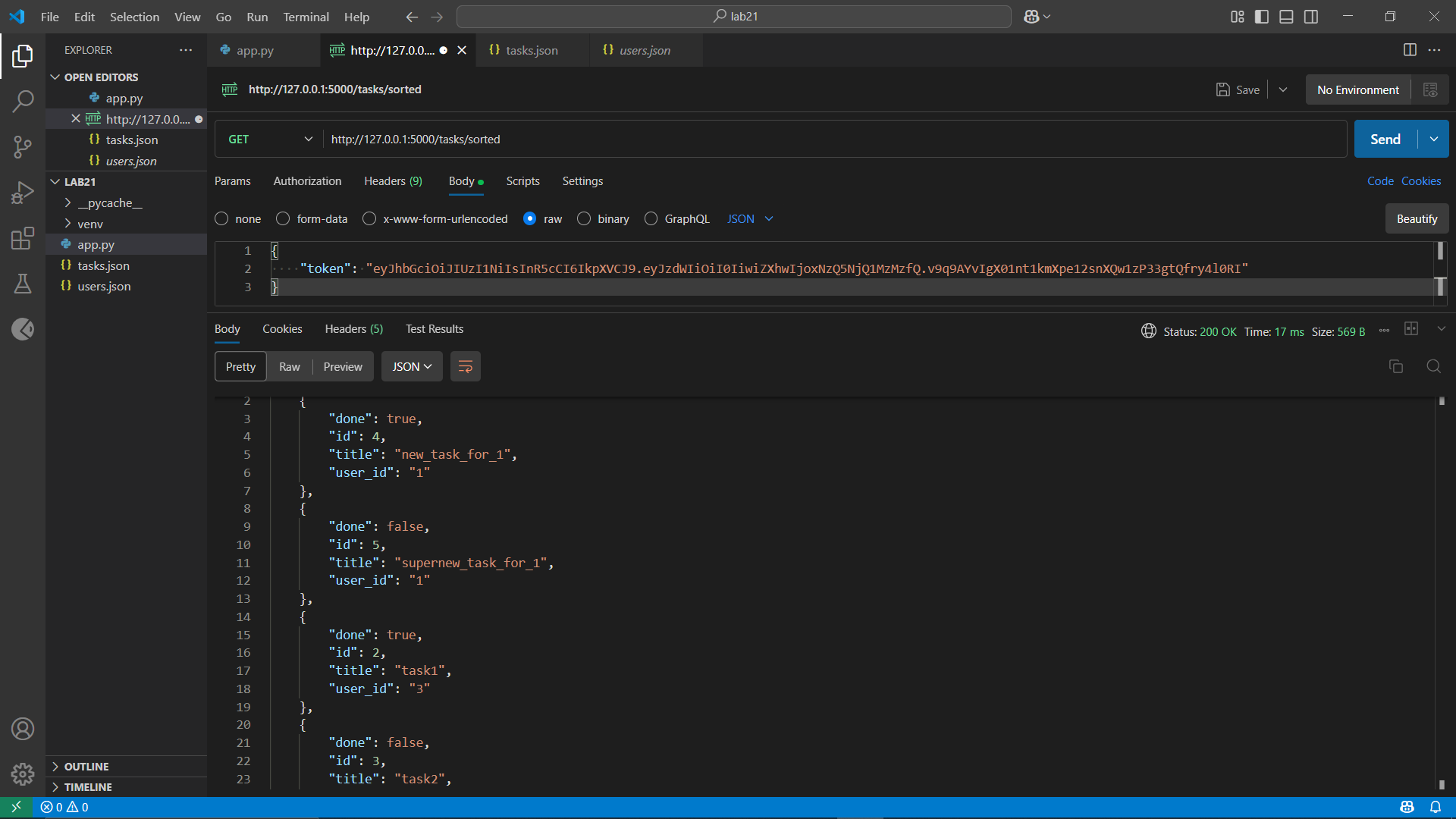


Работа с функциями (уже с проверкой токена)

1. Реализовать возможность фильтрации и сортировки задач в REST API сервисе. Добавить параметры запроса для выборки задач по определенным критериям.

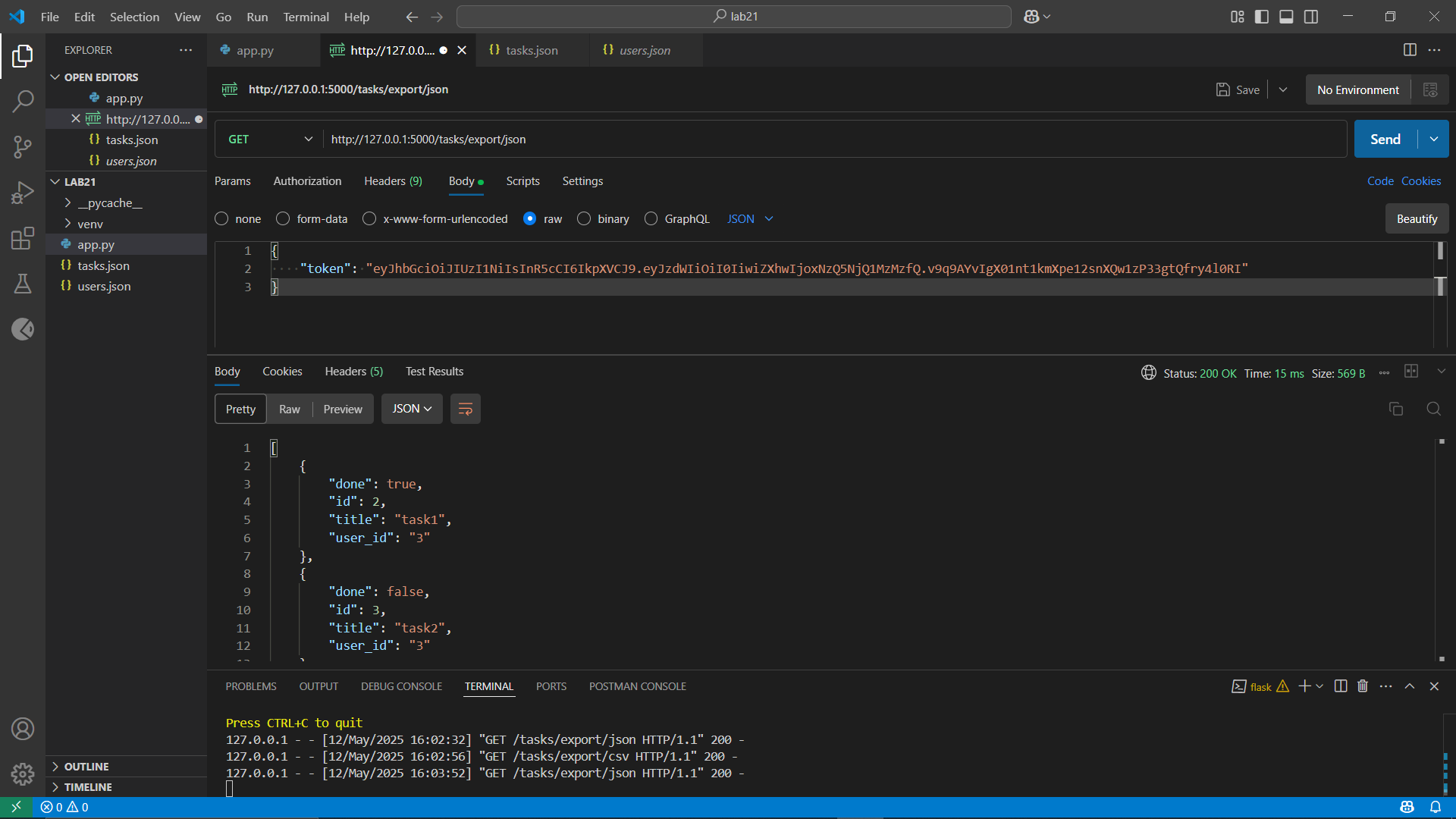
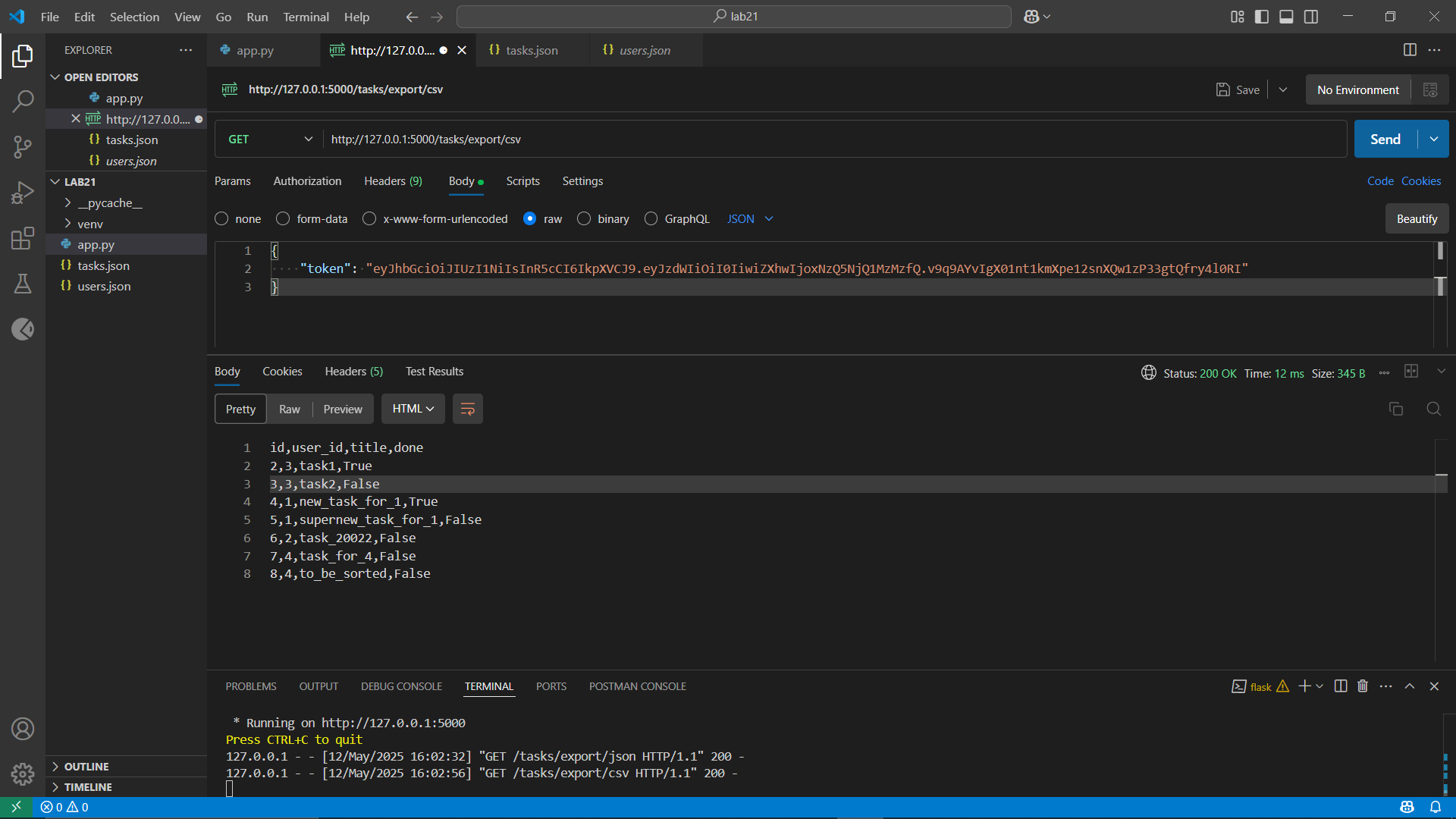


Фильтр на все задачи, помеченные как true



Сортировка названий задач по алфавиту

1. Создать метод для экспорта данных в различные форматы (например, JSON, CSV) в REST API сервисе. Пользователь может получить данные в нужном формате для дальнейшей обработки.



1. Добавить возможность комментирования задач в REST API сервисе. Пользователь может оставлять комментарии к задачам и видеть комментарии других пользователей.

