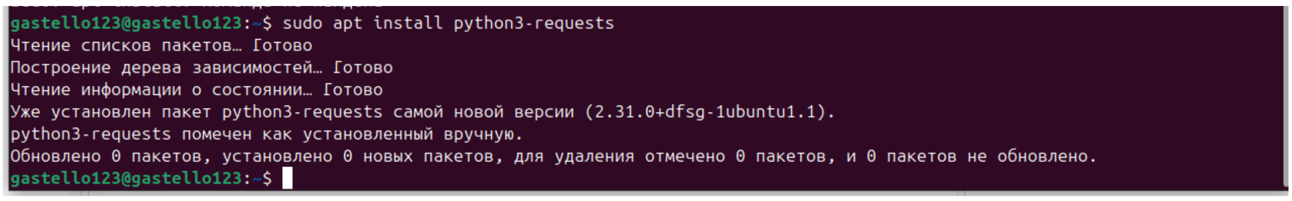
Тестрирование программного обеспечения
Лабораторная работа 5
Разработка автотестов
для API Redfish с использованием PyTest

Дорогин Никита ИП-312

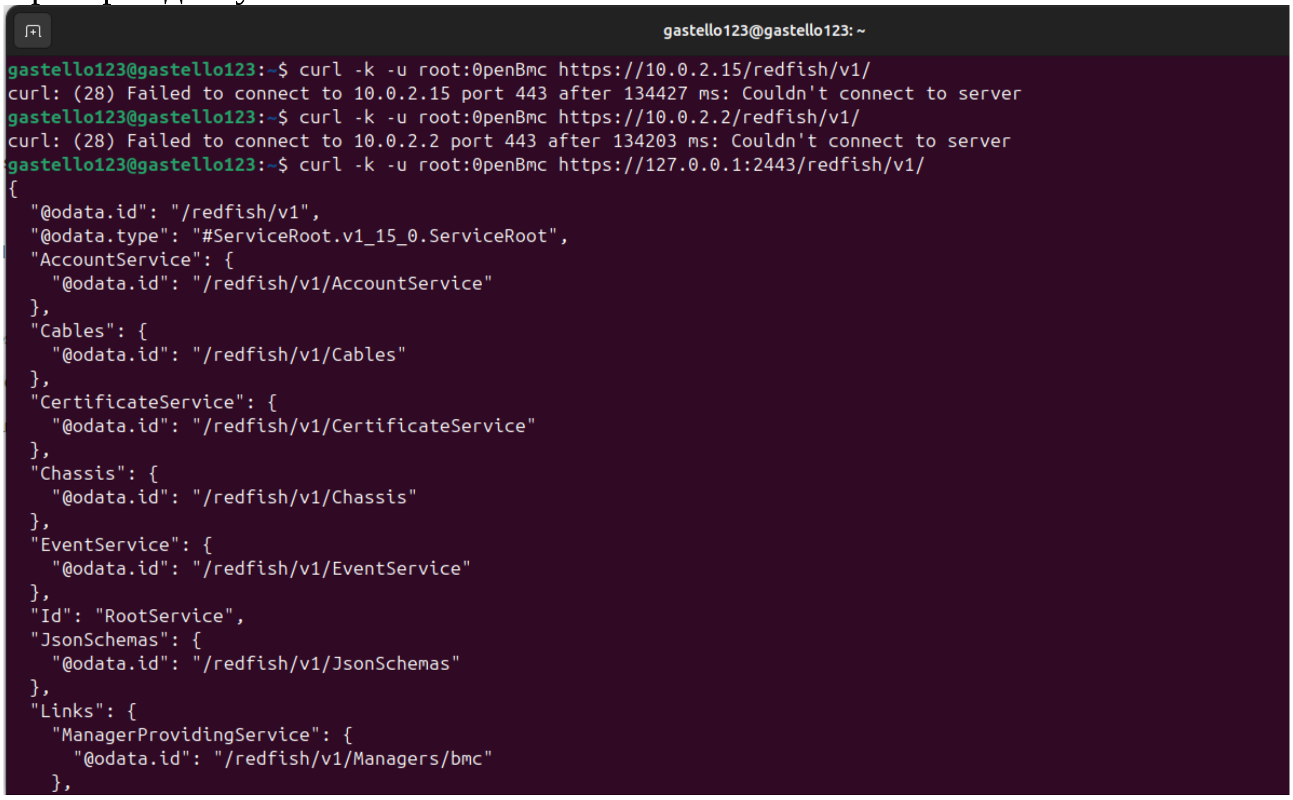
ЧАСТЬ 1: Подготовка окружения.

Установка зависимости pytest-requests:



```
gastello123@gastello123:~$ sudo apt install python3-requests
Чтение списков пакетов… Готово
Построение дерева зависимостей… Готово
Чтение информации о состоянии… Готово
Уже установлен пакет python3-requests самой новой версии (2.31.0+dfsg-1ubuntu1.1).
python3-requests помечен как установленный вручную.
Обновлено 0 пакетов, установлено 0 новых пакетов, для удаления отмечено 0 пакетов, и 0 пакетов не обновлено.
gastello123@gastello123:~$
```

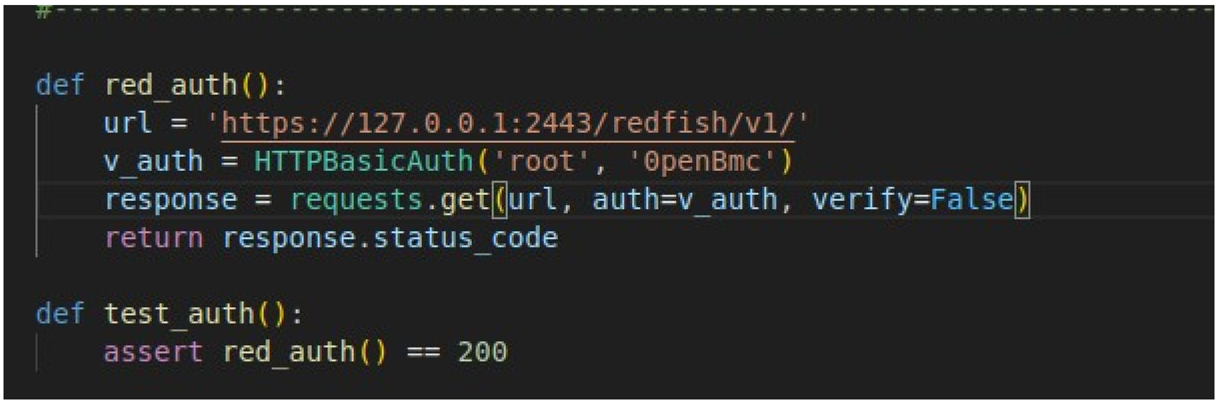
Проверка доступности:



```
gastello123@gastello123:~$ curl -k -u root:0penBmc https://10.0.2.15/redfish/v1/
curl: (28) Failed to connect to 10.0.2.15 port 443 after 134427 ms: Couldn't connect to server
gastello123@gastello123:~$ curl -k -u root:0penBmc https://10.0.2.2/redfish/v1/
curl: (28) Failed to connect to 10.0.2.2 port 443 after 134203 ms: Couldn't connect to server
gastello123@gastello123:~$ curl -k -u root:0penBmc https://127.0.0.1:2443/redfish/v1/
{
  "@odata.id": "/redfish/v1",
  "@odata.type": "#ServiceRoot.v1_15_0.ServiceRoot",
  "AccountService": {
    "@odata.id": "/redfish/v1/AccountService"
  },
  "Cables": {
    "@odata.id": "/redfish/v1/Cables"
  },
  "CertificateService": {
    "@odata.id": "/redfish/v1/CertificateService"
  },
  "Chassis": {
    "@odata.id": "/redfish/v1/Chassis"
  },
  "EventService": {
    "@odata.id": "/redfish/v1/EventService"
  },
  "Id": "RootService",
  "JsonSchemas": {
    "@odata.id": "/redfish/v1/JsonSchemas"
  },
  "Links": {
    "ManagerProvidingService": {
      "@odata.id": "/redfish/v1/Managers/bmc"
    },
```

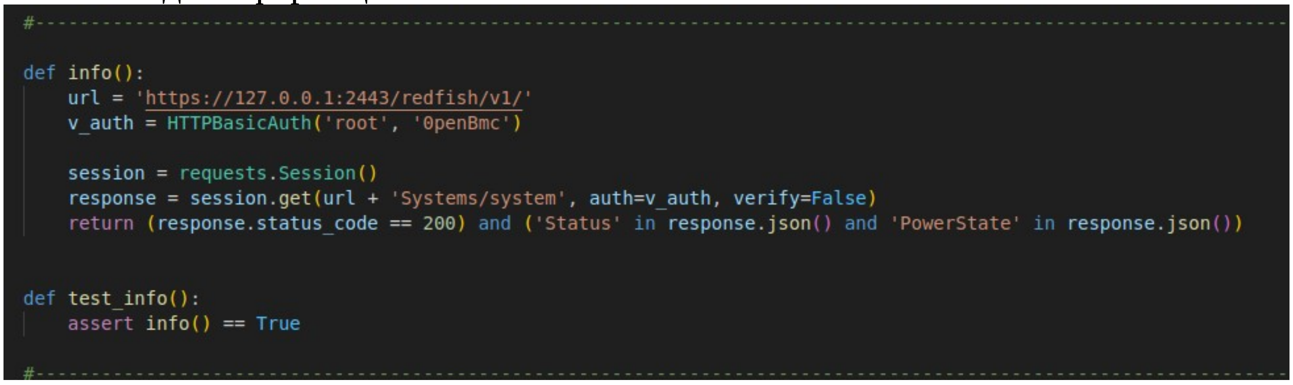
ЧАСТЬ 2: Написание автотестов с использованием PyTest.

Тест авторизации:



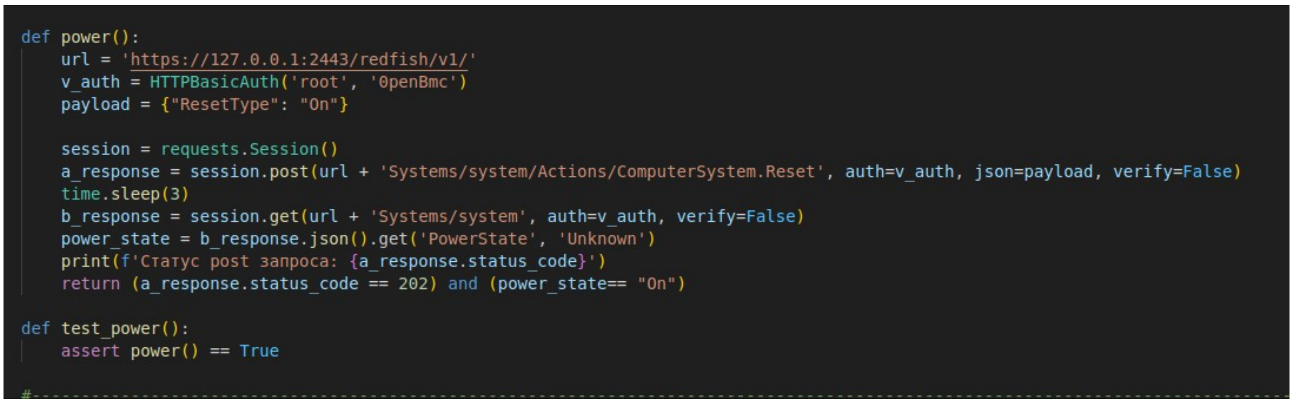
```python
#------------------------------------------------------------------------

def red_auth():
    url = 'https://127.0.0.1:2443/redfish/v1/'
    v_auth = HTTPBasicAuth('root', '0penBmc')
    response = requests.get(url, auth=v_auth, verify=False)
    return response.status_code

def test_auth():
    assert red_auth() == 200
```

Тест вывода информации о системе:



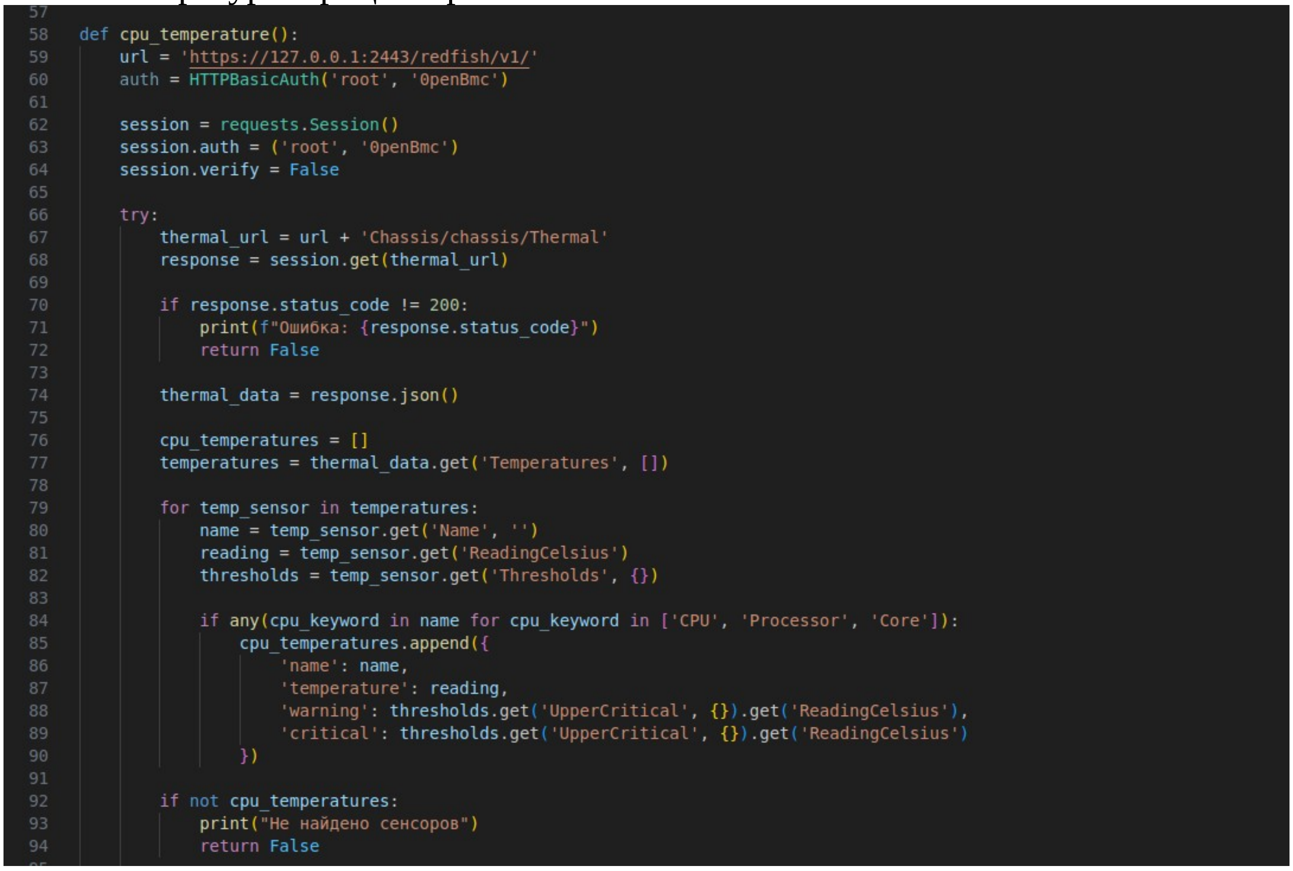
```python
#------------------------------------------------------------------------

def info():
    url = 'https://127.0.0.1:2443/redfish/v1/'
    v_auth = HTTPBasicAuth('root', '0penBmc')

    session = requests.Session()
    response = session.get(url + 'Systems/system', auth=v_auth, verify=False)
    return (response.status_code == 200) and ('Status' in response.json() and 'PowerState' in response.json())


def test_info():
    assert info() == True

#------------------------------------------------------------------------
```

Тест включения питания:

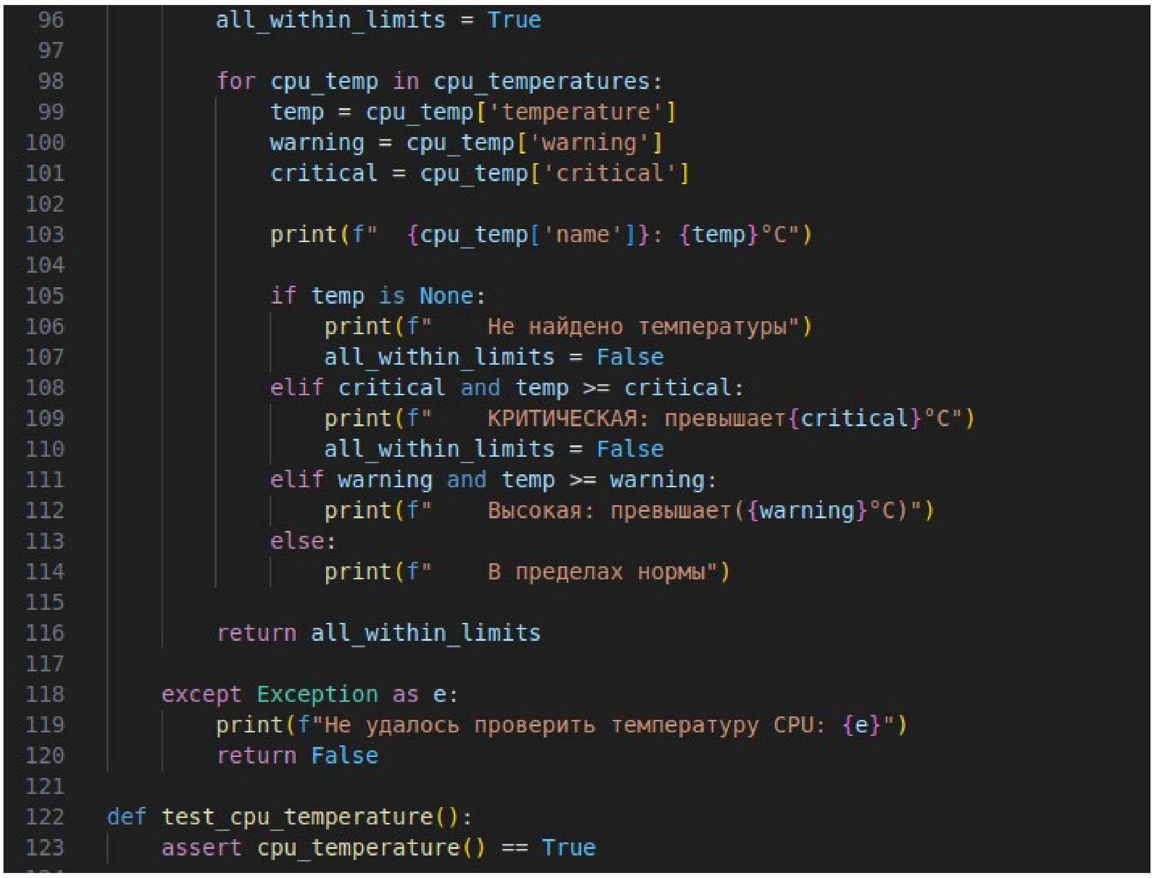


```python
def power():
    url = 'https://127.0.0.1:2443/redfish/v1/'
    v_auth = HTTPBasicAuth('root', '0penBmc')
    payload = {"ResetType": "On"}

    session = requests.Session()
    a_response = session.post(url + 'Systems/system/Actions/ComputerSystem.Reset', auth=v_auth, json=payload, verify=False)
    time.sleep(3)
    b_response = session.get(url + 'Systems/system', auth=v_auth, verify=False)
    power_state = b_response.json().get('PowerState', 'Unknown')
    print(f'Статус post запроса: {a_response.status_code}')
    return (a_response.status_code == 202) and (power_state== "On")

def test_power():
    assert power() == True

#------------------------------------------------------------------------
```

Тест температуры процессора:

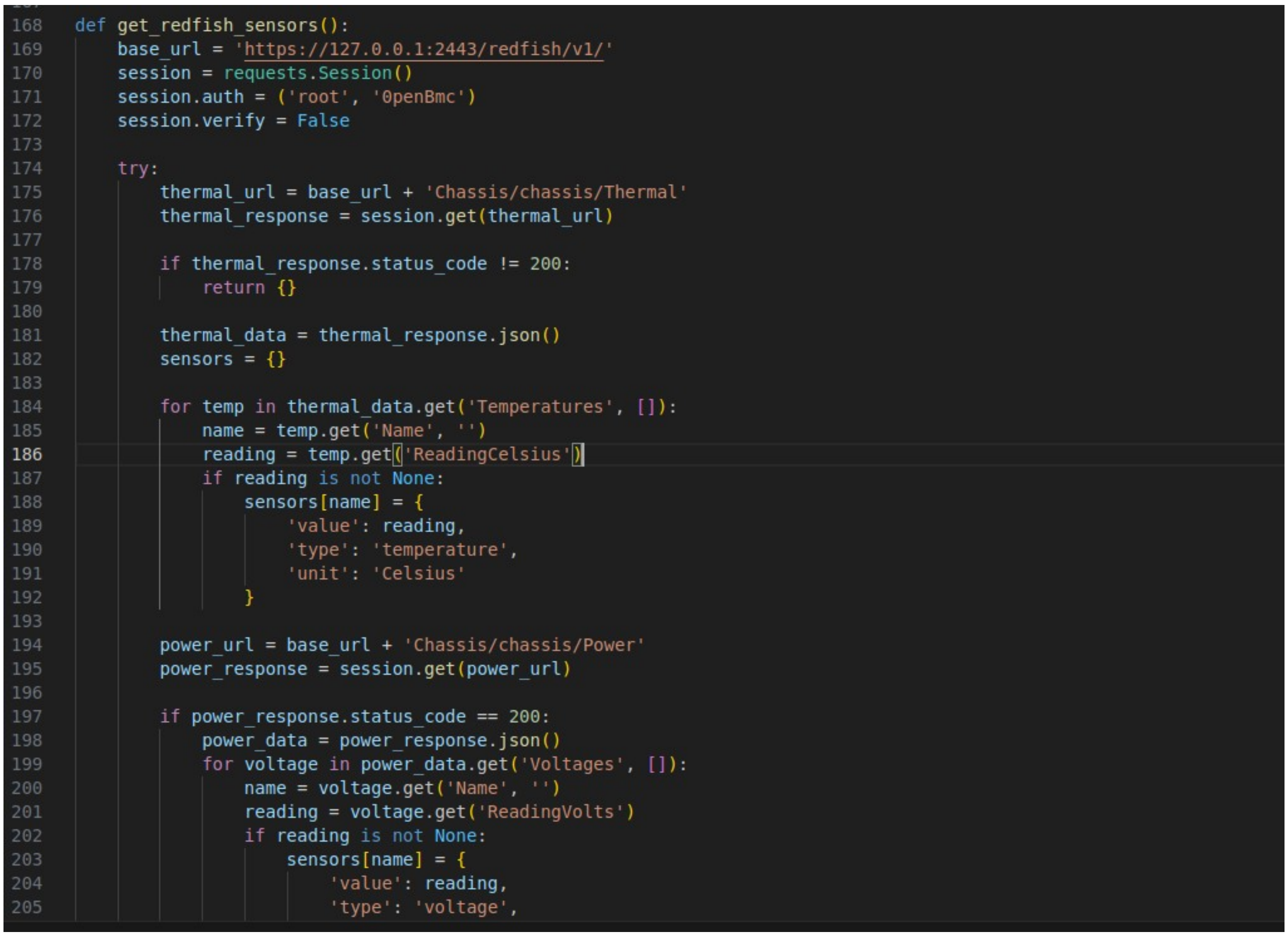


```python
def cpu_temperature():
    url = 'https://127.0.0.1:2443/redfish/v1/'
    auth = HTTPBasicAuth('root', '0penBmc')

    session = requests.Session()
    session.auth = ('root', '0penBmc')
    session.verify = False

    try:
        thermal_url = url + 'Chassis/chassis/Thermal'
        response = session.get(thermal_url)

        if response.status_code != 200:
            print(f"Ошибка: {response.status_code}")
            return False

        thermal_data = response.json()

        cpu_temperatures = []
        temperatures = thermal_data.get('Temperatures', [])

        for temp_sensor in temperatures:
            name = temp_sensor.get('Name', '')
            reading = temp_sensor.get('ReadingCelsius')
            thresholds = temp_sensor.get('Thresholds', {})

            if any(cpu_keyword in name for cpu_keyword in ['CPU', 'Processor', 'Core']):
                cpu_temperatures.append({
                    'name': name,
                    'temperature': reading,
                    'warning': thresholds.get('UpperCritical', {}).get('ReadingCelsius'),
                    'critical': thresholds.get('UpperCritical', {}).get('ReadingCelsius')
                })

        if not cpu_temperatures:
            print("Не найдено сенсоров")
            return False
```

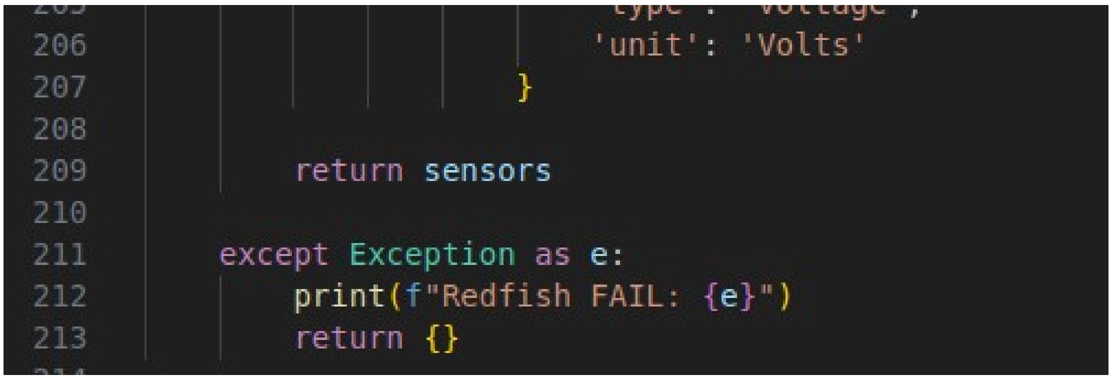


```python
        all_within_limits = True

        for cpu_temp in cpu_temperatures:
            temp = cpu_temp['temperature']
            warning = cpu_temp['warning']
            critical = cpu_temp['critical']

            print(f"  {cpu_temp['name']}: {temp}°C")

            if temp is None:
                print(f"    Не найдено температуры")
                all_within_limits = False
            elif critical and temp >= critical:
                print(f"    КРИТИЧЕСКАЯ: превышает{critical}°C")
                all_within_limits = False
            elif warning and temp >= warning:
                print(f"    Высокая: превышает({warning}°C)")
            else:
                print(f"    В пределах нормы")

        return all_within_limits

    except Exception as e:
        print(f"Не удалось проверить температуру CPU: {e}")
        return False

def test_cpu_temperature():
    assert cpu_temperature() == True
```

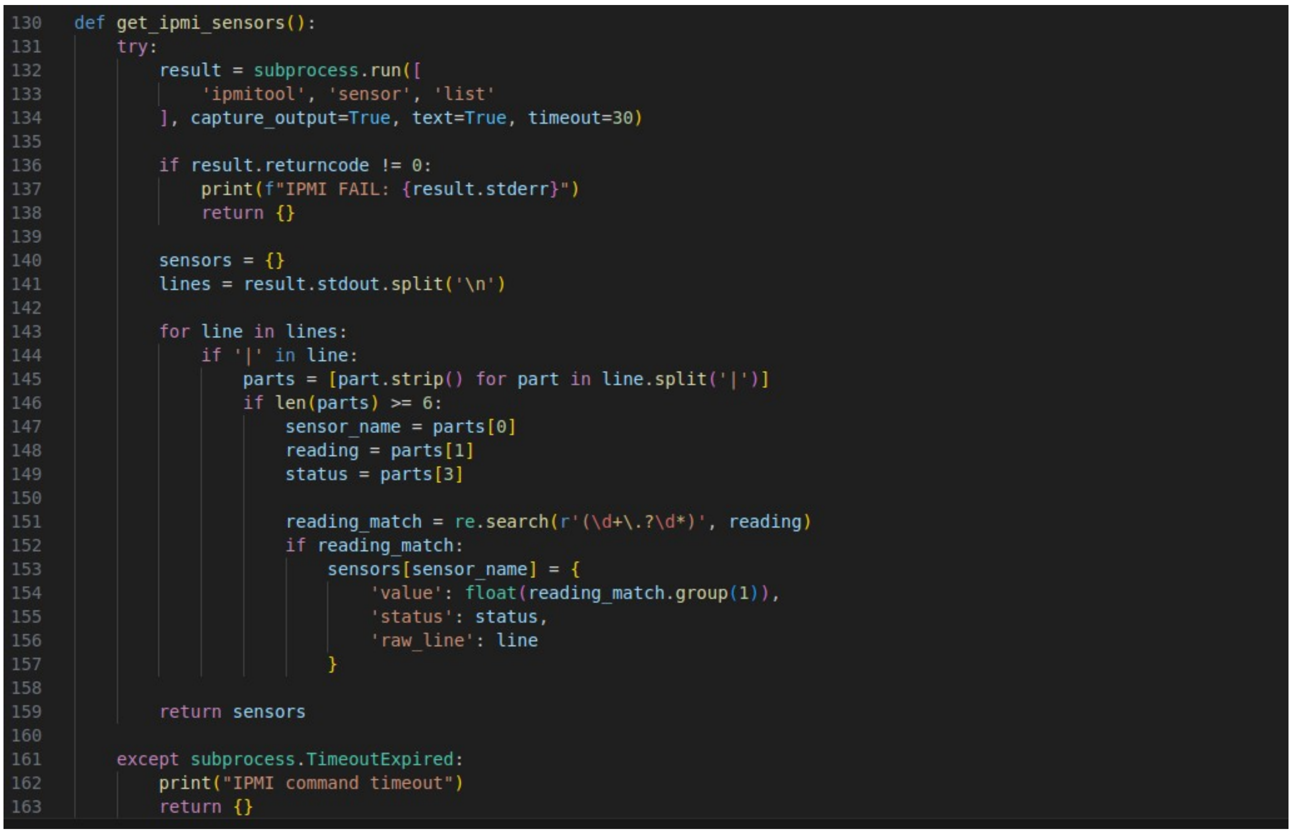
Тест соответствия данных IPMI и Redfish:



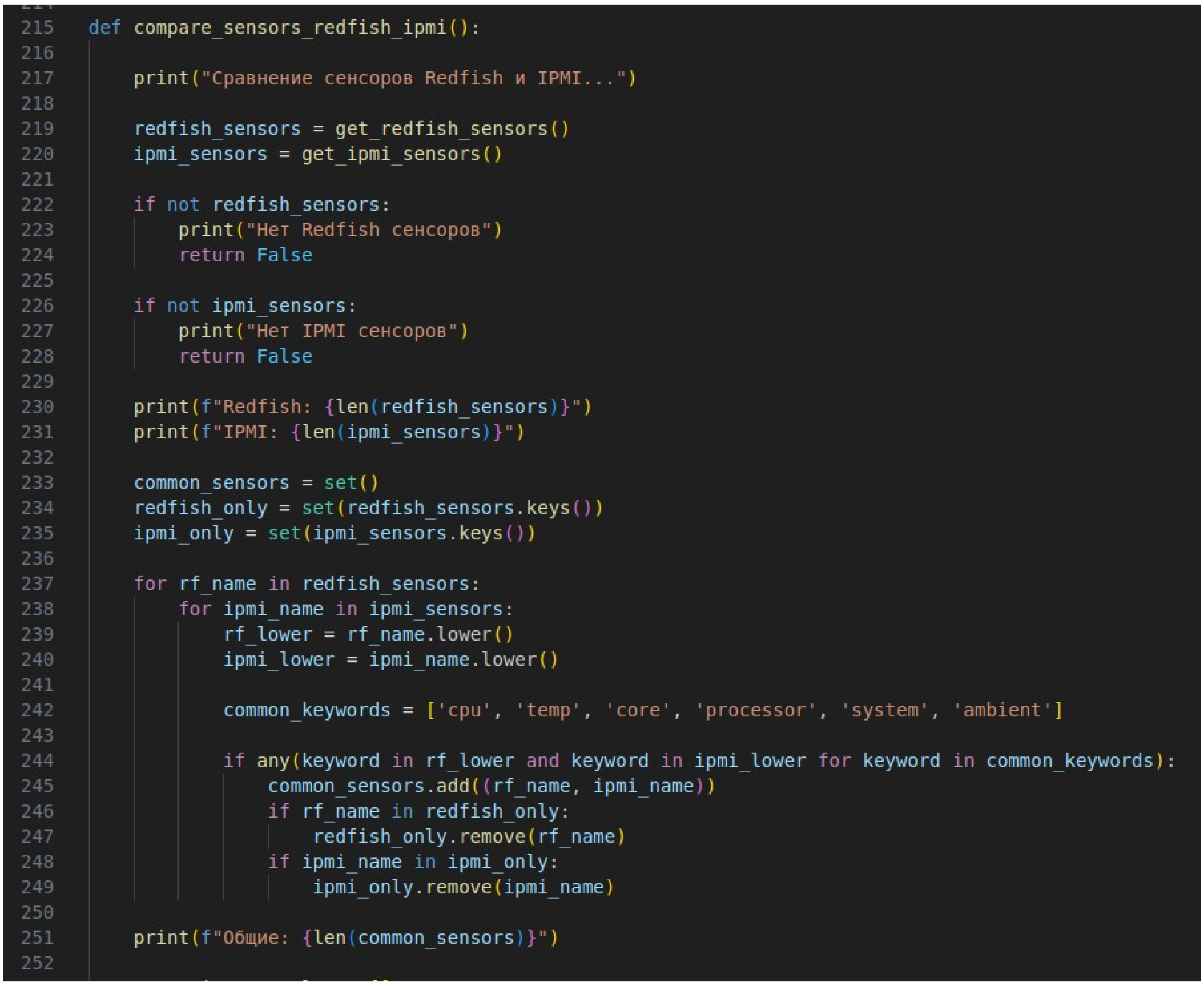
```python
def get_redfish_sensors():
    base_url = 'https://127.0.0.1:2443/redfish/v1/'
    session = requests.Session()
    session.auth = ('root', '0penBmc')
    session.verify = False

    try:
        thermal_url = base_url + 'Chassis/chassis/Thermal'
        thermal_response = session.get(thermal_url)

        if thermal_response.status_code != 200:
            return {}

        thermal_data = thermal_response.json()
        sensors = {}

        for temp in thermal_data.get('Temperatures', []):
            name = temp.get('Name', '')
            reading = temp.get('ReadingCelsius')
            if reading is not None:
                sensors[name] = {
                    'value': reading,
                    'type': 'temperature',
                    'unit': 'Celsius'
                }

        power_url = base_url + 'Chassis/chassis/Power'
        power_response = session.get(power_url)

        if power_response.status_code == 200:
            power_data = power_response.json()
            for voltage in power_data.get('Voltages', []):
                name = voltage.get('Name', '')
                reading = voltage.get('ReadingVolts')
                if reading is not None:
                    sensors[name] = {
                        'value': reading,
                        'type': 'voltage',
```



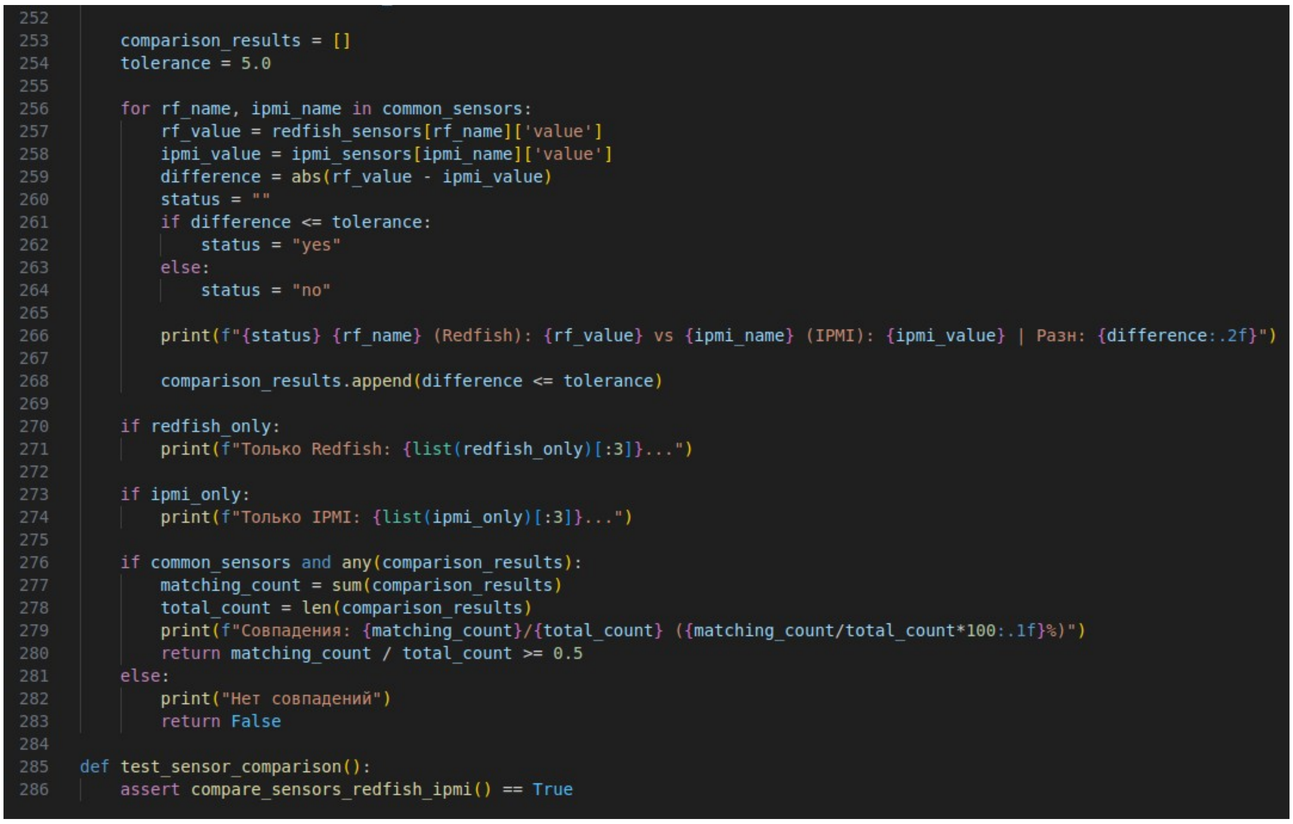
```python
                        'unit': 'Volts'
                    }

        return sensors

    except Exception as e:
        print(f"Redfish FAIL: {e}")
        return {}
```



```python
def get_ipmi_sensors():
    try:
        result = subprocess.run([
            'ipmitool', 'sensor', 'list'
        ], capture_output=True, text=True, timeout=30)

        if result.returncode != 0:
            print(f"IPMI FAIL: {result.stderr}")
            return {}

        sensors = {}
        lines = result.stdout.split('\n')

        for line in lines:
            if '|' in line:
                parts = [part.strip() for part in line.split('|')]
                if len(parts) >= 6:
                    sensor_name = parts[0]
                    reading = parts[1]
                    status = parts[3]

                    reading_match = re.search(r'(\d+\.?\d*)', reading)
                    if reading_match:
                        sensors[sensor_name] = {
                            'value': float(reading_match.group(1)),
                            'status': status,
                            'raw_line': line
                        }

        return sensors

    except subprocess.TimeoutExpired:
        print("IPMI command timeout")
        return {}
```



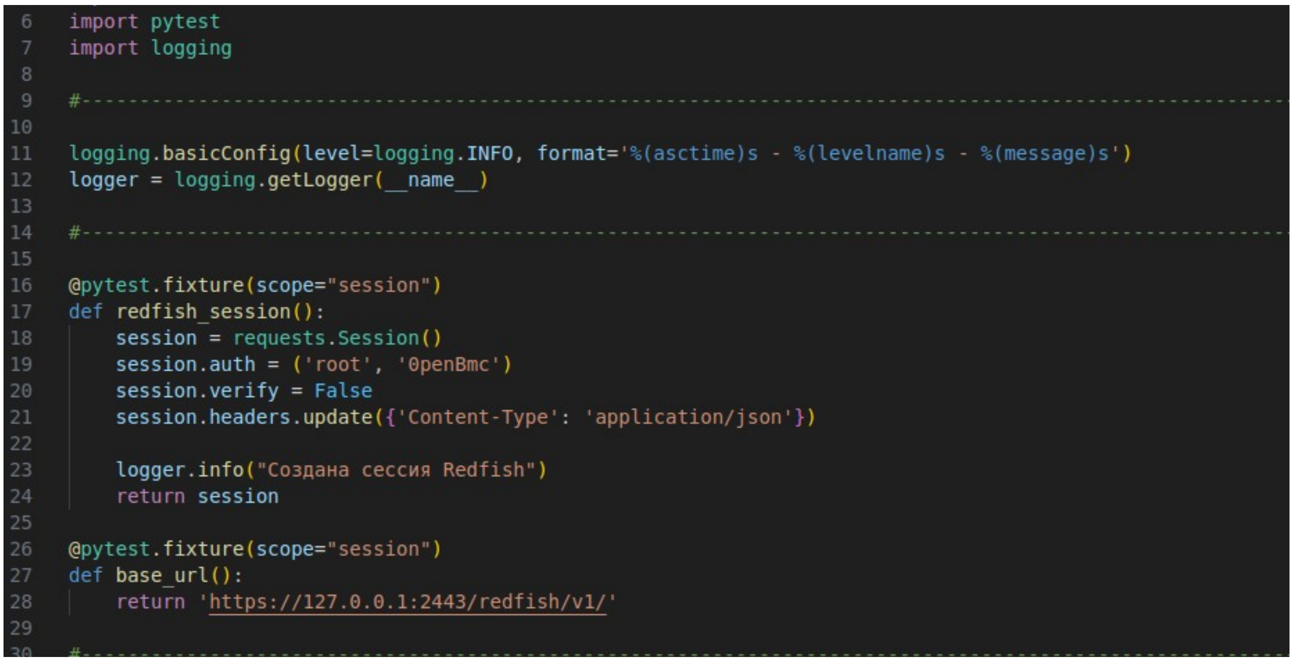
```python
def compare_sensors_redfish_ipmi():

    print("Сравнение сенсоров Redfish и IPMI...")

    redfish_sensors = get_redfish_sensors()
    ipmi_sensors = get_ipmi_sensors()

    if not redfish_sensors:
        print("Нет Redfish сенсоров")
        return False

    if not ipmi_sensors:
        print("Нет IPMI сенсоров")
        return False

    print(f"Redfish: {len(redfish_sensors)}")
    print(f"IPMI: {len(ipmi_sensors)}")

    common_sensors = set()
    redfish_only = set(redfish_sensors.keys())
    ipmi_only = set(ipmi_sensors.keys())

    for rf_name in redfish_sensors:
        for ipmi_name in ipmi_sensors:
            rf_lower = rf_name.lower()
            ipmi_lower = ipmi_name.lower()

            common_keywords = ['cpu', 'temp', 'core', 'processor', 'system', 'ambient']

            if any(keyword in rf_lower and keyword in ipmi_lower for keyword in common_keywords):
                common_sensors.add((rf_name, ipmi_name))
                if rf_name in redfish_only:
                    redfish_only.remove(rf_name)
                if ipmi_name in ipmi_only:
                    ipmi_only.remove(ipmi_name)

    print(f"Общие: {len(common_sensors)}")

```



```python

    comparison_results = []
    tolerance = 5.0

    for rf_name, ipmi_name in common_sensors:
        rf_value = redfish_sensors[rf_name]['value']
        ipmi_value = ipmi_sensors[ipmi_name]['value']
        difference = abs(rf_value - ipmi_value)
        status = ""
        if difference <= tolerance:
            status = "yes"
        else:
            status = "no"

        print(f"{status} {rf_name} (Redfish): {rf_value} vs {ipmi_name} (IPMI): {ipmi_value} | Разн: {difference:.2f}")

        comparison_results.append(difference <= tolerance)

    if redfish_only:
        print(f"Только Redfish: {list(redfish_only)[:3]}...")

    if ipmi_only:
        print(f"Только IPMI: {list(ipmi_only)[:3]}...")

    if common_sensors and any(comparison_results):
        matching_count = sum(comparison_results)
        total_count = len(comparison_results)
        print(f"Совпадения: {matching_count}/{total_count} ({matching_count/total_count*100:.1f}%)")
        return matching_count / total_count >= 0.5
    else:
        print("Нет совпадений")
        return False

def test_sensor_comparison():
    assert compare_sensors_redfish_ipmi() == True
```

Часть 3: Организация кода тестов в файле test_redfish.py

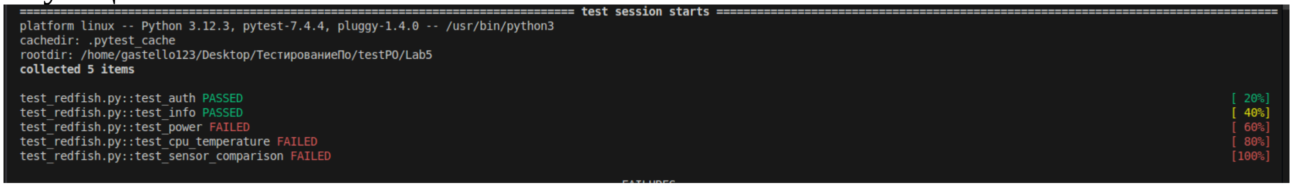
Добавляем логгер и фикстуры:



```python
import pytest
import logging

#------------------------------------------------------------------------------------------------------------

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

#------------------------------------------------------------------------------------------------------------

@pytest.fixture(scope="session")
def redfish_session():
    session = requests.Session()
    session.auth = ('root', '0penBmc')
    session.verify = False
    session.headers.update({'Content-Type': 'application/json'})

    logger.info("Создана сессия Redfish")
    return session

@pytest.fixture(scope="session")
def base_url():
    return 'https://127.0.0.1:2443/redfish/v1/'

```

Изменения в функциях тестов, связанные с этим, смотрите по ссылке на GitHub.

Как итог, большинство этих не тестов не сможет пройти из-за проблемы эмуляции:



```
========================================================================= test session starts ==========================================================================
platform linux -- Python 3.12.3, pytest-7.4.4, pluggy-1.4.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/gastello123/Desktop/ТестированиеПо/testPO/Lab5
collected 5 items

test_redfish.py::test_auth PASSED                                                                                                                                 [ 20%]
test_redfish.py::test_info PASSED                                                                                                                                 [ 40%]
test_redfish.py::test_power FAILED                                                                                                                                [ 60%]
test_redfish.py::test_cpu_temperature FAILED                                                                                                                      [ 80%]
test_redfish.py::test_sensor_comparison FAILED                                                                                                                    [100%]
```

Ссылка на GitHub: https://github.com/NekitD/testPO/blob/main/Lab5/test_redfish.py