Министерство цифрового развития, связи и массовых коммуникаций Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Отчёт

Лабораторная работа №5

«Разработка автотестов для API Redfish с использованием PyTest»

Выполнил студент

группы ИП-311:
Терентьев Андрей

Проверил:

Сентюров Святослав Алеесеевич

Новосибирск 2025 г.

Цель работы

Освоить написание автоматизированных тестов для REST API на примере Redfish API в OpenBMC, используя PyTest. Научиться отправлять HTTP-запросы, проверять их корректность и анализировать ответы сервера.

Ход работы

Установил PyTest.

```
anr1@u24:~$ pytest --version
pytest 7.4.4
anr1@u24:~$
```

Проверил доступность OpenBMC.

```
anr1@u24:~$ curl -k -u root:0penBmc https://localhost: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"
    },
    "Sessions": {
      "@odata.id": "/redfish/v1/SessionService/Sessions"
    }
  },
  "Managers": {
    "@odata.id": "/redfish/v1/Managers"
  },
  "Name": "Root Service",
```

Написал тесты

Фикстура с авторизацией.

```python
import subprocess
import pytest
import requests
from requests.auth import HTTPBasicAuth
import time
import logging

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

BMC_IP = "localhost"
BASE_URL = f"https://{BMC_IP}:2443"
USERNAME = "root"
PASSWORD = "0penBmc"

requests.packages.urllib3.disable_warnings()

@pytest.fixture
def session():
    session = requests.Session()
    session.auth = HTTPBasicAuth(USERNAME, PASSWORD)
    session.verify = False
    return session
```

Тест аутентификации

Отправляет GET-запрос к /redfish/v1/SessionService

Проверяет, что статус ответа = 200 (успех). Отправляет POST-запрос для создания сессии с логином/паролем. Проверяет, что статус ответа = 200 или 201 (сессия создана). Проверяет, что в ответе есть ID сессии и имя пользователя в ответе соответствует отправленному.

```python
    return session

def test_redfish_authentication(session):
    logger.info("Тест аутентификации Redfish")
    try:
        response = session.get(f"{BASE_URL}/redfish/v1/SessionService")
        assert response.status_code == 200

        sessions_url = f"{BASE_URL}/redfish/v1/SessionService/Sessions"
        auth_data = {"UserName": USERNAME, "Password": PASSWORD}
        response = session.post(sessions_url, json=auth_data)

        assert response.status_code in [200, 201]
        session_data = response.json()
        assert "Id" in session_data
        assert session_data["UserName"] == USERNAME
        logger.info("Аутентификация успешна")

    except Exception as e:
        logger.error(f"Ошибка аутентификации: {e}")
        raise
```

Тест информация о системе

```
        logger.error(f"Ошибка аутентификации: {e}")
        raise

def test_system_info(session):
    logger.info("Тест информации о системе")
    try:
        response = session.get(f"{BASE_URL}/redfish/v1/Systems/system")
        assert response.status_code == 200

        system_info = response.json()
        assert "PowerState" in system_info
        assert "Status" in system_info
        logger.info(f"Состояние питания: {system_info.get('PowerState')}")

    except Exception as e:
        logger.error(f"Ошибка получения информации о системе: {e}")
        raise
```

Отправляет GET-запрос к /redfish/v1/Systems/system. Проверяет, что статус ответа = 200.Проверяет наличие поля "PowerState" в JSON-ответе. Проверяет наличие поля "Status" в JSON-ответе

Тест управления питанием

```
def test_power_control(session):
    logger.info("Тест управления питанием")
    try:
        power_url = f"{BASE_URL}/redfish/v1/Systems/system/Actions/ComputerSystem.Reset"
        power_data = {"ResetType": "On"}

        response = session.post(power_url, json=power_data)
        assert response.status_code in [200, 202, 204]
        logger.info(f"Команда питания отправлена, статус: {response.status_code}")

        time.sleep(5)
        response = session.get(f"{BASE_URL}/redfish/v1/Systems/system")
        power_state = response.json().get("PowerState")

        assert power_state in ["On", "Off", "PoweringOn", "PoweringOff"]
        logger.info(f"Текущее состояние питания: {power_state}")

    except Exception as e:
        logger.error(f"Ошибка управления питанием: {e}")
        raise
```

Отправляет POST-запрос к /redfish/v1/Systems/system/Actions/ComputerSystem.Reset.

Передает параметр {"ResetType": "On"} для включения сервера

Проверяет, что статус ответа = 200, 202 или 204 (команда принята)

Ждет 5 секунд для применения изменений. Запрашивает текущее состояние системы. Проверяет, что состояние питания корректное (On, Off, PoweringOn, PoweringOff)

Тест на температуру

```python
def test_thermal_subsystem_structure(session):
    logger.info("Тест thermal subsystem")
    try:
        response = session.get(f"{BASE_URL}/redfish/v1/Chassis/chassis/ThermalSubsystem")
        thermal_data = response.json()

        assert "Name" in thermal_data
        assert "Status" in thermal_data

        status = thermal_data["Status"]
        assert status.get("State") in ["Enabled", "Disabled", "StandbyOff"]

        has_fans = "Fans" in thermal_data and len(thermal_data["Fans"]) > 0
        has_temps = "Temperatures" in thermal_data and len(thermal_data["Temperatures"]) > 0

        assert has_fans or has_temps, "Должны быть данные о вентиляторах или температуре"
        logger.info(f"Thermal subsystem: fans={has_fans}, temps={has_temps}")

    except Exception as e:
        logger.error(f"Ошибка thermal subsystem: {e}")
        raise
```

Отправляет GET-запрос к /redfish/v1/Chassis/chassis/ThermalSubsystem. Проверяет наличие поля "Name" в ответе. Проверяет наличие поля "Status" в ответе. Проверяет, что статус системы имеет допустимое значение. Проверяет наличие данных о вентиляторах ИЛИ температурах. Убеждается, что есть хотя бы какие-то thermal-данные

```python
def test_processors_vs_ipmi(session):
    logger.info("Тест согласованности Redfish и IPMI")
    try:
        response = session.get(f"{BASE_URL}/redfish/v1/Systems/system")
        system_data = response.json()

        redfish_count = system_data.get("ProcessorSummary", {}).get("Count", 0)
        logger.info(f"Redfish CPU count: {redfish_count}")

        ipmi_cmd = [
            'ipmitool', '-I', 'lanplus', '-H', 'localhost',
            '-p', '2623', '-U', 'root', '-P', '0penBmc',
            'sensor', 'list'
        ]

        result = subprocess.run(ipmi_cmd, capture_output=True, text=True)
        if result.returncode != 0:
            logger.warning(f"IPMI команда завершилась с ошибкой: {result.stderr}")

        ipmi_output = result.stdout

        ipmi_cpu_sensors = len([line for line in ipmi_output.split('\n') if 'cpu' in line.lower()])

        assert redfish_count > 0 or ipmi_cpu_sensors > 0, "Нет данных о процессорах ни в Redfish, ни в IPMI"
        logger.info("Согласованность процессоров проверена")

    except Exception as e:
        logger.error(f"Ошибка сравнения Redfish и IPMI: {e}")
```

Добавлены логи и обработка ошибок