ЛАБОРАТОРНА РОБОТА №2
API Smart-пристроїв

Дисципліна «Технології Smart-систем» • Змістовий модуль 2

1. Тема, мета та результати

Тема: API Smart-пристроїв.

Мета: вивчити принципи взаємодії програм із Smart-пристроями через REST API; отримати телеметрію у форматі JSON, реалізувати перевірку й розбір даних, збереження до реляційної бази даних та періодичний збір показників.

Після виконання студент має продемонструвати: коректний HTTP-запит; обробку статусів і помилок; файл JSON; схему БД із трьох таблиць; програму збору показників без дублювання; результат SQL-запиту до накопичених даних.

2. Необхідні програмні засоби

- ПК із Windows, macOS або Linux; Python 3.10 або новіший; редактор коду; доступ до термінала.
- Бібліотека requests; SQLite входить до стандартної бібліотеки Python. Фізичний ESP32 чи датчики не потрібні.

Підготовка середовища (Windows PowerShell):

```
py -m venv .venv
.\.venv\Scripts\Activate.ps1
python -m pip install requests
```

Підготовка середовища (macOS / Linux):

```
python3 -m venv .venv
source .venv/bin/activate
python -m pip install requests
```

Примітка. Якщо PowerShell забороняє активацію середовища, можна виконувати .\.venv\Scripts\python.exe без активації. Використовуйте лише локальний навчальний сервер; наведений демо-ключ не придатний для реальних сервісів.

3. Короткі теоретичні відомості

API (Application Programming Interface) — формалізований спосіб взаємодії програм. REST API зазвичай використовує HTTP, адреси ресурсів і методи GET (читання), POST (створення), PUT/PATCH (оновлення), DELETE (видалення). У Smart-системах пристрій або платформа може повертати телеметрію температури, вологості, стану батареї та інших параметрів.

JSON — текстовий формат обміну даними у вигляді об’єктів, масивів і скалярних значень. XML — альтернативний ієрархічний формат. Клієнт повинен перевіряти коди HTTP, структуру JSON, типи полів, часові мітки та враховувати можливість пропущених значень.

Код HTTP	Значення	Дія клієнта
200	Запит успішний	Прочитати та перевірити JSON
401 / 403	Немає авторизації / доступу	Перевірити ключ та права
404	Ресурс не знайдений	Перевірити URL
429	Перевищено ліміт запитів	Витримати паузу, врахувати Retry-After
500–503	Помилка сервера	Обмежене повторення з паузою

Для автентифікації використовують API key, Bearer token або OAuth 2.0. Захищені реальні інтеграції повинні працювати через HTTPS; ключі не зберігають у публічному репозиторії. Навчальний приклад нижче працює тільки на localhost.

Рекомендовані приклади для порівняння: Home Assistant REST API (довготривалий токен), ThingSpeak Read API (публічні та приватні канали), Open-Meteo API (публічні метеодані без ключа). Open-Meteo надає дані зовнішнього середовища, а не локального фізичного Smart-пристрою.

4. Практична частина

4.1. Підготовка локального API Smart-пристрою

Створіть папку smart_lab2 та файл smart_device_server.py. Сервер імітує телеметрію контролера теплиці, слухає тільки 127.0.0.1:8000 і віддає JSON за GET /api/telemetry. Значення датчиків генеруються штучно.

Приклад 1. Файл smart_device_server.py

```
import json
import os
import random
from datetime import datetime, timezone
from http.server import BaseHTTPRequestHandler, HTTPServer

API_KEY = os.getenv("SMART_API_KEY", "lab-demo-key")

class DeviceHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.headers.get("X-API-Key") != API_KEY:
            self.send_json(401, {"error": "Unauthorized"})
            return
        if self.path != "/api/telemetry":
            self.send_json(404, {"error": "Not found"})
            return

        data = {
            "device": {
                "id": "greenhouse-esp32-01",
                "name": "Smart greenhouse",
                "location": "greenhouse"
            },
            "timestamp": datetime.now(timezone.utc).isoformat(),
            "measurements": {
                "temperature": round(random.uniform(19, 30), 1),
                "humidity": round(random.uniform(40, 70), 1),
                "battery": round(random.uniform(85, 100), 1)
            }
        }
        self.send_json(200, data)

    def send_json(self, status, data):
        body = json.dumps(data).encode("utf-8")
        self.send_response(status)
        self.send_header("Content-Type", "application/json; charset=utf-8")
        self.send_header("Content-Length", str(len(body)))
        self.end_headers()
        self.wfile.write(body)

HTTPServer(("127.0.0.1", 8000), DeviceHandler).serve_forever()
```

Запустіть сервер у першому вікні термінала:

```
python smart_device_server.py
```

Очікувана структура успішної відповіді:

```
{
  "device": {"id": "greenhouse-esp32-01",
          "name": "Smart greenhouse", "location": "greenhouse"},
  "timestamp": "2026-10-13T10:00:00+00:00",
  "measurements": {"temperature": 24.3,
              "humidity": 56.7, "battery": 93.4}
}
```

Примітка. Значення та timestamp у прикладі наведено для ілюстрації; під час запуску вони будуть іншими. Сервер працює, доки ви не завершите його комбінацією Ctrl+C.

4.2. Отримання JSON і створення бази даних

У другому терміналі створіть файл lab2_client.py. Він надсилає HTTP-запит з X-API-Key, перевіряє помилки, зберігає останню відповідь до JSON-файлу, записує три метрики до SQLite та повторює збір тричі. UNIQUE обмежує повторне внесення одного й того самого вимірювання.

Приклад 2. Файл lab2_client.py

```
import json
import os
import sqlite3
import time
from datetime import datetime
import requests

URL = "http://127.0.0.1:8000/api/telemetry"
KEY = os.getenv("SMART_API_KEY", "lab-demo-key")
UNITS = {"temperature": "C", "humidity": "%", "battery": "%"}

def initialize(conn):
    conn.executescript("""
        PRAGMA foreign_keys = ON;
        CREATE TABLE IF NOT EXISTS devices (
            id TEXT PRIMARY KEY, name TEXT NOT NULL, location TEXT
        );
        CREATE TABLE IF NOT EXISTS metrics (
            name TEXT PRIMARY KEY, unit TEXT NOT NULL
        );
        CREATE TABLE IF NOT EXISTS measurements (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            device_id TEXT NOT NULL REFERENCES devices(id),
            metric_name TEXT NOT NULL REFERENCES metrics(name),
            value REAL NOT NULL, measured_at TEXT NOT NULL,
            UNIQUE(device_id, metric_name, measured_at)
        );
    """)

def save(conn, packet):
    dev = packet["device"]
    stamp = packet["timestamp"]
    datetime.fromisoformat(stamp)  # validation
    values = packet["measurements"]
    with conn:
        conn.execute("INSERT OR IGNORE INTO devices VALUES (?, ?, ?)",
                     (dev["id"], dev["name"], dev["location"]))
        for name, value in values.items():
            if name not in UNITS or isinstance(value, bool) or not isinstance(
                    value, (int, float)):
                raise ValueError("Invalid metric: " + name)
            conn.execute("INSERT OR IGNORE INTO metrics VALUES (?, ?)",
                         (name, UNITS[name]))
            conn.execute("""INSERT OR IGNORE INTO measurements
                (device_id, metric_name, value, measured_at)
                VALUES (?, ?, ?, ?)""",
                (dev["id"], name, value, stamp))

def main():
    with sqlite3.connect("smart_lab2.db") as conn:
        initialize(conn)
        for i in range(3):
            try:
                response = requests.get(
                    URL, headers={"X-API-Key": KEY}, timeout=5)
                response.raise_for_status()
                packet = response.json()
                with open("last_response.json", "w", encoding="utf-8") as f:
                    json.dump(packet, f, ensure_ascii=False, indent=2)
                save(conn, packet)
                save(conn, packet)  # repeated insertion must not duplicate
                n = conn.execute(
                    "SELECT COUNT(*) FROM measurements").fetchone()[0]
                print(f"Cycle {i + 1}: database records = {n}")
            except (requests.RequestException, ValueError, KeyError) as exc:
                print("Error:", exc)
            if i < 2:
                time.sleep(2)

if __name__ == "__main__":
    main()
```

Виконайте у другому терміналі:

```
python lab2_client.py
```

У каталозі мають з’явитися smart_lab2.db та last_response.json. Для нової бази кількість рядків measurements після трьох циклів зазвичай дорівнює 9 (3 метрики × 3 вимірювання); якщо запити відбулися в однаковий timestamp, UNIQUE може зменшити кількість рядків.

4.3. Перевірка структури та результатів

Перевірте базу з Python або через DB Browser for SQLite. Приклад окремого SQL-запиту, який потрібно виконати та включити до звіту:

```
SELECT device_id, metric_name, value, measured_at
FROM measurements
ORDER BY measured_at DESC
LIMIT 10;
```

Додатково побудуйте ER-діаграму з трьох таблиць: devices, metrics, measurements. Позначте первинні та зовнішні ключі, а також обмеження UNIQUE(device_id, metric_name, measured_at).

5. Завдання для самостійного виконання

1. Порівняйте документацію двох або трьох API Smart-пристроїв/IoT-платформ: формат відповіді, модель авторизації, ліміти доступу, практичне призначення.

2. Запустіть емулятор пристрою; отримайте дані запитом GET /api/telemetry; збережіть один результат у файлі last_response.json.

3. Поясніть призначення полів JSON та доповніть програмну перевірку значень (наприклад, вологість має бути в межах 0–100 %).

4. Розробіть ER-діаграму та створіть структуру бази даних щонайменше з трьох таблиць.

5. Забезпечте періодичне опитування API та захист від дублювання даних; перевірте повторне збереження одного пакета.

6. Продемонструйте обробку щонайменше двох помилок: 401 (змінений ключ), 404 (неправильний шлях) або недоступний сервер.

7. Додайте одну нову метрику до емулятора (наприклад, освітленість у lux), налаштуйте її збереження та виконайте SQL-запит середнього значення за вибраною метрикою.

8. Опишіть, як замінити локальний емулятор на реальне API ThingSpeak чи Home Assistant (URL, авторизація, структура даних), не розкриваючи секретів доступу.

6. Вимоги до звіту

1. Тема, мета роботи, короткий опис обраного сценарію Smart-системи.

2. Порівняльна таблиця API, короткий виклад принципів REST і JSON.

3. Структура API-відповіді, фрагменти коду з поясненнями та результатами запуску.

4. ER-діаграма, SQL-структура таблиць, приклад SQL-запиту та його результат.

5. Скриншоти/логи успішного запиту, обробки помилки та перевірки відсутності дублікатів.

6. Аналіз обмежень навчальної моделі (відсутність фізичних датчиків, локальне з’єднання) і висновки.

7. Оформлений звіт завантажити до Системи електронного забезпечення навчання ЗНУ.

7. Контрольні питання

1. Що таке REST API та чим він відрізняється від локального виклику функції?

2. Для чого використовуються HTTP-методи GET, POST, PUT і DELETE?

3. Яка різниця між JSON та XML?

4. Як відрізняються API key, Bearer token і OAuth 2.0?

5. Яку дію повинен виконати клієнт після відповідей 401, 404 та 429?

6. Навіщо використовувати тайм-аут, retry та exponential backoff?

7. Чому timestamp важливий для зберігання телеметрії?

8. Яким способом реалізовано захист від дублювання записів?

9. Чому для часових рядів можуть бути доцільні спеціалізовані бази даних?

10. Як забезпечити безпечне зберігання та передавання ключів доступу?

8. Рекомендовані джерела

- Python Software Foundation. Документація sqlite3: https://docs.python.org/3/library/sqlite3.html
- Requests. Документація: https://requests.readthedocs.io/
- Home Assistant REST API: https://developers.home-assistant.io/docs/api/rest/
- ThingSpeak REST API: https://www.mathworks.com/help/thingspeak/rest-api.html
- Open-Meteo API: https://open-meteo.com/en/docs