ЛАБОРАТОРНА РОБОТА №4

Розробка графічного інтерфейсу користувача для виведення даних

Дисципліна «Технології Smart-систем»

1. Тема, мета та результати

Тема: Розробка графічного інтерфейсу користувача для виведення даних.

Мета: вивчити принципи побудови GUI Smart-систем і реалізувати інтерфейс для перегляду телеметрії, збереженої в базі даних під час ЛР №2.

Результат: робочий застосунок із таблицею вимірювань, фільтрами, двома графіками, індикаторами стану, автооновленням, синхронізацією з Cloud Firestore та сповіщеннями про критичні значення; макет і протокол тестування.

2. Необхідні програмні засоби

ПК із Windows, macOS або Linux; Python 3.10+; редактор коду та браузер. Рекомендовано Streamlit + pandas + Plotly, стандартний модуль sqlite3, пакет firebase-admin. Фізичні датчики не потрібні. Використайте smart_lab2.db з ЛР №2: таблиці devices, metrics, measurements.

Створіть окрему теку smart_lab4, скопіюйте до неї smart_lab2.db. Для Windows / macOS / Linux у активованому віртуальному середовищі виконайте:

```
python -m pip install streamlit pandas plotly firebase-admin
python -m streamlit run dashboard.py
```

Примітка: для інсталяції залежностей спочатку потрібно створити й активувати venv, як у ЛР №2. Для хмарного етапу потрібні власний проєкт Firebase і дозволений доступ до Cloud Firestore; витрати залежать від плану та фактичного використання.

3. Короткі теоретичні відомості

3.1. Інтерфейс Smart-системи

GUI відображає поточні показники, історію, тривоги та доступні дії користувача. Для часових рядів переважно застосовують лінійний графік, для порівнянь — стовпчиковий, для поточного стану — значення з позначенням одиниць і статусом свіжості. Однакові кольори й одиниці повинні мати сталий зміст. Дані можуть оновлюватися опитуванням (polling), через WebSocket або події БД.

3.2. Порівняння бібліотек

Засіб	Переваги	Обмеження
Tkinter	Постачається з Python, десктопний GUI	Графіки та вебдоступ потребують додаткових засобів
PySide6 / PyQt	Гнучкий багатовіконний інтерфейс	Більший обсяг коду, умови ліцензування
Streamlit	Швидкий вебдашборд мовою Python	Менше контролю над складними сценаріями UI
Dash	Потужні інтерактивні Plotly-графіки	Потрібно опанувати callbacks

3.3. Архітектура даних і Cloud Firestore

Використаємо поділ на доступ до даних (SQLite/Firestore), логіку вибору та перевірки (Python) і представлення (Streamlit). Cloud Firestore зберігає документи в колекціях, а не SQL-таблиці. Для прикладу колекція measurements містить документи з полями device_id, metric_name, value, measured_at. Синхронізація з SQLite має бути ідемпотентною: повторне передавання того самого рядка не створює дублікатів.

4. Практична частина

4.1. Підготовка та макет інтерфейсу

Перевірте наявність smart_lab2.db. Якщо база містить лише 9 рядків, повторіть збір з ЛР №2, щоб отримати часовий ряд. Створіть у Figma або diagrams.net три wireframe-екрани: огляд, історія, налаштування. Передбачте вибір пристрою, метрики та періоду; покажіть місце для повідомлення про відсутність даних.

4.2. Зчитування SQLite та побудова дашборда

Створіть dashboard.py. Код нижче використовує фактичну схему ЛР №2 і не потребує зміни таблиць. Вибір пристрою, метрики й днів здійснюється в боковій панелі. Кнопка «Оновити» повторно виконує запит; для автоматичного оновлення використайте фрагмент у наступному пункті.

```
import sqlite3
from datetime import datetime, timedelta, timezone
import pandas as pd
import plotly.express as px
import streamlit as st

DB = "smart_lab2.db"
st.set_page_config(page_title="Smart telemetry", layout="wide")
st.title("Моніторинг Smart-системи")

@st.cache_data(ttl=5)
def load_data():
    with sqlite3.connect(DB) as conn:
        return pd.read_sql_query("""
            SELECT m.id, m.device_id, d.name AS device_name,
                   m.metric_name, m.value, mt.unit, m.measured_at
            FROM measurements m
            JOIN devices d ON d.id = m.device_id
            JOIN metrics mt ON mt.name = m.metric_name
            ORDER BY m.measured_at DESC
        """, conn)

df = load_data()
if df.empty:
    st.warning("У базі ще немає вимірювань")
    st.stop()
df["time"] = pd.to_datetime(df["measured_at"], utc=True,
                            errors="coerce")
```

```
df = df.dropna(subset=["time"])
if df.empty:
    st.error("Немає коректних часових міток")
    st.stop()

devices = sorted(df["device_id"].unique())
device = st.sidebar.selectbox("Пристрій", devices)
metrics = sorted(df.loc[df.device_id == device,
                        "metric_name"].unique())
metric = st.sidebar.selectbox("Метрика", metrics)
days = st.sidebar.selectbox("Період (днів)", [1, 7, 30, 365], index=2)
if st.sidebar.button("Оновити"):
    load_data.clear()
    st.rerun()

cutoff = datetime.now(timezone.utc) - timedelta(days=days)
f = df[(df.device_id == device) &
       (df.time >= cutoff)].copy()
metric_df = f[f.metric_name == metric].sort_values("time")
if metric_df.empty:
    st.info("За обраний період даних немає")
    st.stop()
latest = metric_df.iloc[-1]
st.metric(f"{metric} ({latest['unit']})", f"{latest['value']:.1f}")
st.plotly_chart(px.line(metric_df, x="time", y="value",
                        title="Динаміка метрики"), use_container_width=True)
avg = f.groupby("metric_name", as_index=False)["value"].mean()
st.plotly_chart(px.bar(avg, x="metric_name", y="value",
                       title="Середні значення (різні одиниці!)"),
                use_container_width=True)
st.caption("Порівняння метрик із різними одиницями — лише демонстрація; "
           "для змістовного аналізу групуйте однакові одиниці.")
st.dataframe(f.sort_values("time", ascending=False),
             use_container_width=True, hide_index=True)

THRESHOLDS = {"temperature": 28, "humidity": 65}
for name, limit in THRESHOLDS.items():
    rows = f[f.metric_name == name].sort_values("time")
    if not rows.empty and rows.iloc[-1]["value"] > limit:
        st.error(f"Поріг перевищено: {name} > {limit}")
```

Для порівняльного графіка зберігайте обрані одиниці вимірювання; порівнювати температуру в °C та вологість у % на одній шкалі методологічно некоректно. У власній роботі побудуйте змістовний другий графік: порівняння температури різних пристроїв або середню температуру за днями.

4.3. Автоматичне оновлення

Виконайте одну з реалізацій: використайте підтримуваний у вашій версії Streamlit фрагмент із run_every="10s" для частини, що читає БД, або застосуйте спеціальний компонент автооновлення. Не створюйте нескінченний while-цикл у головному потоці Streamlit. При демонстрації запустіть збір ЛР №2 в іншому терміналі та перевірте появу нових точок не пізніше одного інтервалу оновлення.

```
# Приклад у версіях Streamlit із підтримкою st.fragment:
@st.fragment(run_every="10s")
def live_status():
    load_data.clear()
    fresh = load_data()
    st.caption(f"Записів у БД: {len(fresh)}")
# Викличте live_status() у нижній частині dashboard.py.
```

4.4. Синхронізація SQLite → Cloud Firestore

У Firebase Console створіть проєкт, увімкніть Cloud Firestore у тестовому навчальному середовищі та отримайте облікові дані сервісного акаунта для Admin SDK. JSON-файл секретів зберігайте поза репозиторієм і не додавайте до звіту. Встановіть змінну середовища GOOGLE_APPLICATION_CREDENTIALS із локальним шляхом до ключа. Для реального клієнтського застосунку потрібні Firebase Authentication та обмежувальні Security Rules; Admin SDK обходить правила клієнта й має працювати тільки в довіреному процесі.

Створіть sync_firestore.py і запустіть після інсталяції firebase-admin:

```
import sqlite3
import firebase_admin
from firebase_admin import firestore

firebase_admin.initialize_app()  # GOOGLE_APPLICATION_CREDENTIALS
cloud = firestore.client()
with sqlite3.connect("smart_lab2.db") as conn:
    rows = conn.execute("""
        SELECT id, device_id, metric_name, value, measured_at
        FROM measurements
    """).fetchall()
for row_id, dev, metric, value, ts in rows:
    # Стабільний ідентифікатор = повторний запуск без дублікатів.
    cloud.collection("measurements").document(str(row_id)).set({
        "device_id": dev, "metric_name": metric,
        "value": value, "measured_at": ts
    })
print("Синхронізовано:", len(rows))
```

Відображення з Firestore: у програмі GUI виконайте cloud.collection("measurements").stream(), перетворіть документи на DataFrame і застосуйте ті самі фільтри. Для великих наборів замість читання всієї колекції використовуйте server-side фільтрацію, індекси й пагінацію. Порівняйте кількість документів після двох запусків синхронізації. Ідентифікатори рядків зберігають унікальність у межах однієї локальної БД; для кількох джерел використовуйте складений ключ із device_id або власний глобальний ID.

4.5. Тестування

Тест	Що перевірити	Очікувана поведінка
Порожня БД	Немає rows у measurements	Інформаційне повідомлення без падіння
Нова телеметрія	Додати записи через ЛР №2	Дані з’являються після оновлення
Перевищений поріг	Температура понад 28 °C	Відображається попередження
Обсяг даних	1 000, 10 000 та 100 000 рядків	Зафіксувати час завантаження,

		уникати зависання
Firestore недоступний	Від’єднати мережу або прибрати credentials	Контрольована помилка, локальні дані доступні

Проведіть коротке UX-тестування з 2–3 користувачами: попросіть знайти поточну температуру, переглянути історію та визначити тривогу. Зафіксуйте складнощі й запропонуйте одне покращення. Не публікуйте ключі Firebase і персональні дані.

5. Завдання для самостійного виконання

1. Порівняйте Tkinter, PySide/PyQt, Streamlit або Dash за 4–5 критеріями.

2. Побудуйте три wireframe-екрани: огляд, історія, налаштування.

3. Реалізуйте застосунок, що читає саме базу smart_lab2.db із ЛР №2.

4. Додайте фільтри пристрою/періоду, сортування таблиці, два осмислені графіки та індикатор стану.

5. Реалізуйте оновлення за таймером та перевірте його на нових записах.

6. Синхронізуйте SQLite з Cloud Firestore і покажіть читання з хмари, а також захист від повторного запису.

7. Реалізуйте попередження про критичне значення й перевірте його.

8. Виконайте функціональні, навантажувальні й UX-тести; поясніть обмеження прототипу.

6. Вимоги до звіту

1. Тема, мета, короткий опис Smart-системи й використаних даних ЛР №2.

2. Порівняльна таблиця бібліотек і макети трьох екранів.

3. Опис архітектури GUI, SQL-запит та ключові фрагменти коду.

4. Скриншоти фільтрації, таблиці, графіків, автооновлення й сповіщення.

5. Схема колекції Firestore, спосіб синхронізації, підтвердження відсутності дублікатів (без секретів).

6. Таблиця тестів, результати UX-перевірки, висновки й обмеження.

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

7. Контрольні питання

1. Які функції виконує GUI у Smart-системі?

2. Чим відрізняються Tkinter, PySide/PyQt і Streamlit/Dash?

3. Чому лінійний графік підходить для часових рядів?

4. У чому суть MVC/MVVM та поділу на шари?

5. Чим SQLite відрізняється від Cloud Firestore?

6. Як реалізувати оновлення за подією й polling?

7. Як визначити свіжість телеметрії та не сплутати її зі станом пристрою?

8. Як організувати порогові сповіщення?

9. Як оптимізувати великі таблиці й запити?

10. Чому Admin SDK не можна вбудовувати в публічний клієнт?

8. Рекомендовані джерела

• Streamlit: https://docs.streamlit.io/

• Python sqlite3: https://docs.python.org/3/library/sqlite3.html

• Plotly Python: https://plotly.com/python/

• Firebase Firestore: https://firebase.google.com/docs/firestore

• Firebase Admin Python: https://firebase.google.com/docs/admin/setup