Лабораторна робота №8

Відображення вимірювальної інформації у веб-застосунках

Мета: Ознайомитись з основними методами відображення інформації у веб-застосунка, побудова діаграм з вимірювальною інформацією.

8.1 Теоретичні відомості

Параметри запиту URL-адреси – це параметри, що додаються до URL-адреси для передачі додаткової інформації на сервер. Вони слідують за ? символом в URL-адресі та розділяються символом &. Наприклад:

/greet?name=John&age=25

У цьому прикладі nameта age– це параметри запиту зі значеннями Johnта 25відповідно. Параметри запиту дозволяють користувачам взаємодіяти з відповідями веб-сервера та налаштовувати їх без зміни коду на стороні сервера.

Обидва типи параметрів передають дані до веб-застосунків, але вони виконують різні ролі та використовуються в різних контекстах.

Параметри шляху є невід'ємною частиною URL-адреси та використовуються для точного визначення конкретних ресурсів. Наприклад, в URL-адресі /users/123є 123 параметром шляху, який ідентифікує конкретного користувача. Використовуйте параметри шляху для важливих, ієрархічних даних, що визначають ідентичність ресурсу.

Параметри запиту надають додаткову інформацію та йдуть після а ? в URL-адресі. Наприклад, у /products?category=books&sort=price_asc, categoryта sort є параметрами запиту. Вони ідеально підходять для додаткових даних, які налаштовують запит, таких як фільтри та сортування.

Розуміння відмінностей допомагає вам вибрати правильний підхід: параметри шляху для обов'язкових, специфічних для ресурсу даних та параметри запиту для необов'язкових, настроюваних даних.

Flask спрощує обробку параметрів запитів URL-адрес за допомогою request модуля. Ось простий приклад, який показує, як можна витягти параметри запиту за допомогою request.args.get():

```
from flask import request
@app.route('/route', methods=['GET'])
def function():
    # Extract the 'parameter_name' query parameter or use 'default_value' if not provided
    variable = request.args.get('parameter_name', 'default_value')
```

У цьому прикладі ми імпортуємо request модуль з Flask для роботи з параметрами запиту. Вам взагалі не потрібно змінювати декоратор маршруту. Просто витягніть значення параметра запиту parameter_name за допомогою request.args.get('parameter_name', 'default_value'), який також встановлює значення за замовчуванням, якщо параметри не передано. Давайте розберемо це детальніше:

requestМістить усі дані, пов'язані з HTTP-запитом, надісланим до сервера.

argsСтруктура, подібна до словника, всередині об'єкта request, яка містить параметри запиту URL-адреси.

get: Отримує значення, пов'язане із зазначеним ключем (ім'я параметра запиту) у args, з необов'язковим значенням за замовчуванням, якщо ключ відсутній.

Тепер створимо маршрут Flask, який приймає параметри запиту та відповідає JSON-повідомленням.

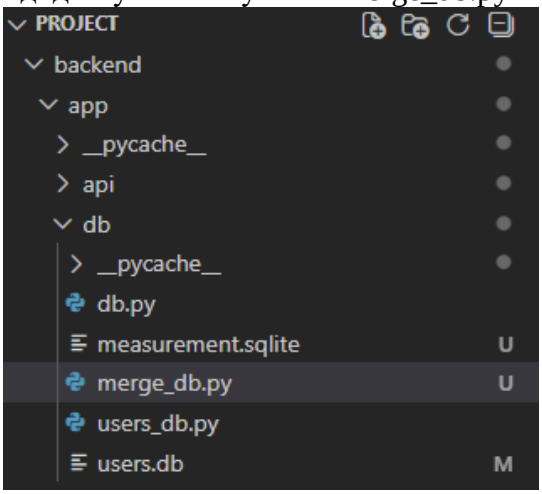
```
from flask import Flask, jsonify, request
app = Flask(__name__)
@app.route('/greet', methods=['GET'])
def greet():
    # Extract the 'name' query parameter or use a default value
    name = request.args.get('name', 'Guest')

    # Return a JSON response with the query parameter
    return jsonify(message=f"Greetings, {name}! Welcome to the query parameter route.")
```

У цій функції маршруту request.args.get('name', 'Guest') використовується для вилучення name параметра запиту, за замовчуванням встановлюючи значення , 'Guest' якщо його не надано. Зрештою, функція повертає об'єкт JSON, який містить значення параметра name.

8.2 Порядок виконання роботи

0. Дочекатись схвалення попереднього PR, стягнути зміни локально з гілки main та створити нову гілку для реалізації нового функціоналу;

1. Виконати злиття двох баз даних users.db та measurement.sqlite використовуючи код з додатку 1. Розмістити файл measurement.sqlite поруч з users.db та створити скрипт merge_db.py в тій же папці та вставити код з додатку 1 та запустити merge_db.py



Після злиття перевірити коректність за допомогою SQlite-viewer (https://inloop.github.io/sqlite-viewer/) завантаживши users.db на сервіс.

2. Зі сторони серверу додати ендпойнти :
- отримання списку існуючих локацій, де встановлені сенсори;

app/api/locations.py

```
from flask import Blueprint, jsonify
from app.db.locations import location_all
bp = Blueprint("locations", __name__)

@bp.get("/all")
def health_check():
  locations = location_all()
  return  jsonify({f"data": locations}), 200
```

- отримання списку існуючих сенсорів;
- отримання списку існуючих сенсорів по локації;
- отримання даних вимірювань по сенсору (тільки час виміру *ts*);
- отримання даних вимірювання по сенсору в заданий інтервал часу *from ts to ts*;

app/api/sensors.py

```
from flask import Blueprint, jsonify, request
from app.db.sensors import sensors_all, get_all_measurements_by_sensor, get_measurement_by_sensor_from_to, sensors_by_location
from datetime import datetime
bp = Blueprint("sensors", __name__)

DATETIME_FORMAT = "%Y-%m-%d %H:%M:%S"

@bp.get("/all")
def all_sensors():
  sensors = sensors_all()
  return jsonify({f"data": sensors}), 200

@bp.get("/all/location/<int:location_id>")
def all_sensors_by_location(location_id):
    sensors = sensors_by_location(location_id)
    return jsonify({f"data": sensors}), 200


@bp.get("/sensor")
def measurement_by_sensor_from_to():
  sensor_id = request.args.get("sensor_id", type=int)
  time_from = request.args.get("time_from" )
  time_to = request.args.get("time_to")
  sensor_info = get_measurement_by_sensor_from_to(sensor_id, time_from, time_to)
  return jsonify({f"data": sensor_info}), 200

@bp.get("/sensor/<int:sensor_id>")
def measurements_by_sensor(sensor_id):
  measurements_sensor = get_all_measurements_by_sensor(sensor_id)
  if not measurements_sensor:
        return jsonify({"error": "No measurements for this sensor"}), 404

  return jsonify({f"data": measurements_sensor}), 200
```

По правилам декомпозиції уся взаємодія з базою даних має знаходитись в окремій папці, тому:

app/db/locations.tsx

```
from app.db.db import get_db
def location_all():
  database = get_db()
  cur = database.execute("SELECT * FROM 'locations'")
  res = cur.fetchall()
  return [dict(row) for row in res]
```

app/db/sensors.tsx

```
from app.db.db import get_db
def sensors_all():
```

```
  database = get_db()
  cur = database.execute("SELECT * FROM 'sensors'")
  res = cur.fetchall()
  return [dict(row) for row in res]

def get_all_measurements_by_sensor(sensor_id):
  database = get_db()
  cur = database.execute("SELECT * FROM 'readings' WHERE readings.sensor_id = ?", (sensor_id,))
  res = cur.fetchall()
  return [dict(row) for row in res]

def get_measurement_by_sensor_from_to(sensor_id, time_from, time_to):
  database = get_db()
  cur = database.execute("""SELECT * FROM readings
                    WHERE sensor_id = ?
                     AND ts BETWEEN ? AND ?
                    ORDER BY ts ASC""",(sensor_id, time_from, time_to,)
)
  res = cur.fetchall()
  return [dict(row) for row in res]

def sensors_by_location(location_id):
  database = get_db()
  cur = database.execute("""SELECT *
                    FROM 'sensors'
                    WHERE location_id = ?
                    ORDER BY code ASC""",(location_id,)
)
  res = cur.fetchall()
  return [dict(row) for row in res]
```

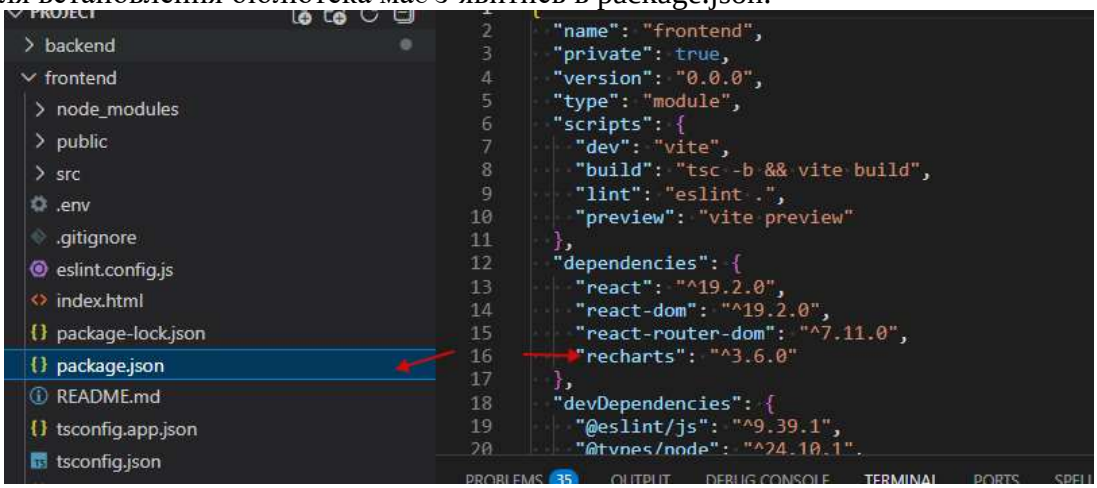
Зареєструвати додаткові роути в __init__.py та імпортувати відповідні модулі sensors та locations

```
app.register_blueprint(sensors, url_prefix="/api/sensors")
app.register_blueprint(locations, url_prefix="/api/locations")
```

3. Зі сторони клієнта реалізувати:

- інсталювати бібліотеку recharts та ознайомитись з документацією *Recharts (*https://recharts.github.io/en-US/api/);

Після встановлення бібліотека має з’явитись в package.json:



Оновити роут ***/dashboard*** на якому буде відображатись графік:

- на графіку потрібно реалізувати випадний список локацій;

- після вибору локації має бути доступний випадний список сенсорів, що встановлений на вибраній локації;

- після вибору сенсору мають бути доступні 2 випадні списки часу вимірювань *from to*

- після введення усіх даних та натискання кнопки «show» має відображатись графік з вибраними метриками (Температура, Вологість, Напруга, рівень CO, Освітленість, та рівень NO_2) Приклад приведено в додатку 2;

src/components/Dashboard/index.tsx

```
import React, { useEffect, useState } from "react";
import {
  ResponsiveContainer,
  LineChart,
  Line,
  CartesianGrid,
  XAxis,
  YAxis,
  Tooltip,
  Legend,
} from "recharts";
import s from "./style.module.css";
import { fetchLocations } from "../../api/locations/locations";
import type { ILocation } from "../../interfaces/locations";
import {
  fetchSensor,
  fetchSensorsList,
  getData,
} from "../../api/sensors/sensors";
import type { ISensorInfo, ISensors } from "../../interfaces/sensors";

type MetricKey =
  | "co_ppm"
  | "humidity_pct"
  | "light_lux"
  | "no2_ppb"
  | "temperature_c"
  | "voltage_v";

const METRICS: { key: MetricKey; label: string; color: string }[] = [
  { key: "temperature_c", label: "Temperature, °C", color: "#34d399" },
  { key: "humidity_pct", label: "Humidity, %", color: "#60a5fa" },
  { key: "voltage_v", label: "Voltage, V", color: "#f97316" },
  { key: "co_ppm", label: "CO, ppm", color: "#facc15" },
  { key: "light_lux", label: "Light, lux", color: "#a855f7" },
  { key: "no2_ppb", label: "NO₂, ppb", color: "#f97316" },
];

export const Dashboard: React.FC = () => {
  const [selectedMetrics, setSelectedMetrics] = useState<MetricKey[]>([
    "temperature_c",
  ]);
  const [locations, setLocations] = useState<ILocation[]>([]);
  const [sensorsList, setSensorsList] = useState<ISensors[]>([]);
  const [sensorId, setSensorId] = useState<number>(-1);
  const [sensorInfo, setSensorInfo] = useState<ISensorInfo[] | undefined>();
  const [locationId, setLocationId] = useState<number>(-1);
  const [timeFrom, setTimeFrom] = useState<string>("");
  const [timeTo, setTimeTo] = useState<string>("");
  const [viewData, setViewData] = useState<ISensorInfo[] | undefined>();

  useEffect(() => {
    fetchLocations().then(setLocations);
  }, []);

  useEffect(() => {
    if (locationId != -1) {
      fetchSensorsList(locationId).then(setSensorsList);
    }
  }, [locationId]);

  useEffect(() => {
    if (sensorId != -1) {
      fetchSensor(sensorId).then(setSensorInfo);
```

```
  }
}, [sensorId]);

const toggleMetric = (metric: MetricKey) => {
  setSelectedMetrics((prev) =>
    prev.includes(metric)
      ? prev.filter((m) => m !== metric)
      : [...prev, metric]
  );
};

const handleApplyFilters = async () => {
  try {
    const data = await getData(sensorId, timeFrom, timeTo);
    setViewData(data);
  } catch (err) {
    console.error(err);
  }
};

return (
  <div className={s.root}>
    <div className={s.container}>
      <header className={s.header}>
        <div>
          <h1 className={s.title}>Dashboard</h1>
          <p className={s.subtitle}>A quick overview of your measurements.</p>
        </div>
      </header>

      <section className={s.chartCard}>
        <div className={s.chartHeader}>
          <div>
            <h2 className={s.cardTitle}>Measurements</h2>
            <p className={s.cardText}>
              One flexible chart. Toggle metrics and change filters to explore
              data.
            </p>
          </div>

          <div className={s.chartFilters}>
            <div className={s.filterGroup}>
              <label>Location</label>
              <select
                value={locationId}
                onChange={(e) => setLocationId(Number(e.target.value))}
              >
                <option value={-1}>-----||------</option>
                {locations.map((location, id) => {
                  return (
                    <option key={id} value={location.id}>
                      {location.name}
                    </option>
                  );
                })}
              </select>
            </div>

            <div className={s.filterGroup}>
              <label>Sensor</label>
              <select
                value={sensorId}
                onChange={(e) => setSensorId(Number(e.target.value))}
              >
                <option value={-1}>-----||------</option>
```

```
        {sensorsList.map((sensors, id) => {
         return (
          <option key={id} value={sensors.id}>
           {sensors.code}
          </option>
         );
        })}
       </select>
      </div>

      <div className={s.filterGroup}>
       <label>From</label>

       <select
        value={timeFrom}
        onChange={(e) => setTimeFrom(e.target.value)}
       >
        <option value={-1}> YYYY-MM-DD HH:MM:SS"</option>
        {sensorInfo?.map((sensor, id) => {
         return (
          <option key={id} value={sensor.ts}>
           {sensor.ts}
          </option>
         );
        })}
       </select>
      </div>

      <div className={s.filterGroup}>
       <label>To</label>
       <select
        value={timeTo}
        onChange={(e) => setTimeTo(e.target.value)}
       >
        <option value={-1}> YYYY-MM-DD HH:MM:SS"</option>
        {sensorInfo
         ?.map((sensor, id) => {
          return (
           <option key={id} value={sensor.ts}>
            {sensor.ts}
           </option>
          );
         })
         .reverse()}
       </select>
      </div>

      <button
       className={s.btnApply}
       onClick={handleApplyFilters}
       disabled={!(locationId && sensorId && timeFrom && timeTo)}
      >
       Apply
      </button>
     </div>
    </div>

    <div className={s.chartMetrics}>
     {METRICS.map((m) => (
      <button
       key={m.key}
       className={
        selectedMetrics.includes(m.key)
         ? `${s.metricToggle} ${s.metricToggleActive}`
         : s.metricToggle
```

```
              }
              onClick={() => toggleMetric(m.key)}
            >
              <span
                className={s.metricColorDot}
                style={{ backgroundColor: m.color }}
              />
              {m.label}
            </button>
          ))}
        </div>

        <div className={s.chartWrapper}>
          <ResponsiveContainer width="100%" height="100%">
            <LineChart
              data={viewData}
              margin={{ top: 10, right: 30, left: 0, bottom: 0 }}
            >
              <CartesianGrid stroke="#1f2937" vertical={false} />
              <XAxis
                dataKey="ts"
                tick={{ fill: "#9ca3af", fontSize: 12 }}
                tickLine={false}
                interval="preserveStartEnd"
                minTickGap={40}
              />
              <YAxis
                tick={{ fill: "#9ca3af", fontSize: 12 }}
                tickLine={false}
              />
              <Tooltip
                contentStyle={{
                  backgroundColor: "#020617",
                  border: "1px solid #1f2937",
                  borderRadius: 8,
                  color: "#e5e7eb",
                  fontSize: 12,
                }}
              />
              <Legend wrapperStyle={{ color: "#9ca3af" }} />
              {METRICS.filter((m) => selectedMetrics.includes(m.key)).map(
                (m) => (
                  <Line
                    key={m.key}
                    type="step"
                    dataKey={m.key}
                    name={m.label}
                    stroke={m.color}
                    strokeWidth={2}
                    dot={false}
                    activeDot={{ r: 4 }}
                  />
                )
              )}
            </LineChart>
          </ResponsiveContainer>
        </div>
      </section>
    </div>
  </div>
 );
};

export default Dashboard;
```

src/components/style.module.css

```
.root {
  min-height: 100vh;
  background: radial-gradient(circle at top left, #1e293b 0, #020617 45%, #000 100%);
  color: #e5e7eb;
  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", sans-serif;
}

.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 32px 24px 40px;
}

.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 24px;
}

.title {
  font-size: 28px;
  font-weight: 700;
  color: #f9fafb;
}

.subtitle {
  margin-top: 4px;
  font-size: 14px;
  color: #9ca3af;
}

.topRow {
  display: grid;
  grid-template-columns: repeat(2, minmax(0, 1fr));
  gap: 20px;
  margin-bottom: 20px;
}

.card,
.chartCard,
.bottomCard {
  background: #020617;
  border-radius: 18px;
  border: 1px solid #111827;
  padding: 18px 22px 20px;
  box-shadow: 0 18px 45px rgba(0, 0, 0, 0.7);
}

.cardTitle {
  font-size: 16px;
  font-weight: 600;
  color: #e5e7eb;
  margin-bottom: 6px;
}

.cardText {
  font-size: 13px;
  color: #9ca3af;
}

.cardList {
  margin: 8px 0 0;
  padding-left: 18px;
```

```
  font-size: 13px;
  color: #d1d5db;
}

.statusBadges {
  display: flex;
  gap: 8px;
  margin-top: 14px;
}

.badge {
  padding: 4px 10px;
  border-radius: 999px;
  font-size: 11px;
  font-weight: 500;
}

.badgeSuccess {
  background: #16a34a1a;
  color: #4ade80;
  border: 1px solid #16a34a;
}

.badgeOutline {
  background: transparent;
  color: #60a5fa;
  border: 1px solid #1d4ed8;
}

.chartCard {
  margin-bottom: 20px;
}

.chartHeader {
  display: flex;
  justify-content: space-between;
  gap: 18px;
  align-items: flex-start;
  margin-bottom: 12px;
}

.chartFilters {
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
  align-items: flex-end;
}

.filterGroup {
  display: flex;
  flex-direction: column;
  gap: 4px;
}

.filterGroup label {
  font-size: 11px;
  color: #9ca3af;
}

.filterGroup select,
.filterGroup input {
  background: #020617;
  border-radius: 10px;
  border: 1px solid #1f2937;
  padding: 6px 9px;
```

```
  font-size: 12px;
  color: #e5e7eb;
  outline: none;
}


.btnApply {
  padding: 7px 14px;
  border-radius: 999px;
  border: none;
  background: linear-gradient(135deg, #3b82f6, #22c55e);
  color: white;
  font-size: 12px;
  font-weight: 600;
  cursor: pointer;
}

.btnApply:hover {
  opacity: 0.9;
}

.btnApply:disabled{
  padding: 7px 14px;
  border-radius: 999px;
  border: none;
  background: linear-gradient(135deg, #020617, #1f2937);
  color: white;
  font-size: 12px;
  font-weight: 600;
  cursor: pointer;
  border: solid 1px black;
  opacity: 1;
}

.chartMetrics {
  display: flex;
  flex-wrap: wrap;
  gap: 8px;
  margin: 8px 0 12px;
}

.metricToggle {
  display: inline-flex;
  align-items: center;
  gap: 6px;
  padding: 5px 10px;
  border-radius: 999px;
  border: 1px solid #1f2937;
  background: #020617;
  font-size: 11px;
  color: #9ca3af;
  cursor: pointer;
}

.metricToggleActive {
  border-color: #3b82f6;
  color: #e5e7eb;
  background: #0b1120;
}

.metricColorDot {
  width: 8px;
  height: 8px;
  border-radius: 999px;
}
```

```
.chartWrapper {
  margin-top: 2px;
  height: 380px;
}

.bottomCard {
  margin-top: 10px;
}

@media (max-width: 900px) {
  .topRow {
    grid-template-columns: 1fr;
  }

  .chartHeader {
    flex-direction: column;
    align-items: flex-start;
  }

  .chartFilters {
    width: 100%;
    justify-content: flex-start;
  }
}
```

4. Додати запити на сервер на створені ендпойнти для отримання інформації про локації, сенсори та вимірювальні данні.

src/api/locations/locations.tsx

```
import type { ILocation } from "../../interfaces/locations";
import type { IRes } from "../../interfaces/response";

export const fetchLocations = async (): Promise<ILocation[]> => {
  const url = import.meta.env.VITE_API_URL;

  const res = await fetch(`${url}/api/locations/all`, {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
    },
  });

  if (!res.ok) {
    throw new Error(
      `Failed to fetch locations: ${res.status} ${res.statusText}`
    );
  }
  const body: IRes<ILocation[]> = await res.json();
  return body.data;
};
```

src/api/sensors/sensors.tsx

```
import type { IRes } from "../../interfaces/response";
import type { ISensorInfo, ISensors } from "../../interfaces/sensors";

export const fetchSensorsList = async (
  location_id: number
): Promise<ISensors[]> => {
  const url = import.meta.env.VITE_API_URL;


  const res = await fetch(`${url}/api/sensors/all/location/${location_id}`, {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
```

```
    },
  });

  if (!res.ok) {
    throw new Error(
      `Failed to fetch locations: ${res.status} ${res.statusText}`
    );
  }
  const body: IRes<ISensors[]> = await res.json();
  return body.data;
};

export const fetchSensor = async (
  sensor_id: number
): Promise<ISensorInfo[]> => {
  const url = import.meta.env.VITE_API_URL;

  const res = await fetch(`${url}/api/sensors/sensor/${sensor_id}`, {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
    },
  });

  if (!res.ok) {
    throw new Error(
      `Failed to fetch locations: ${res.status} ${res.statusText}`
    );
  }
  const body: IRes<ISensorInfo[]> = await res.json();
  return body.data;
};

export const getData = async (
  sensor_id: number,
  time_from: string,
  time_to: string
): Promise<ISensorInfo[]> => {
  const url = import.meta.env.VITE_API_URL;

  const res = await fetch(
    `${url}/api/sensors/sensor?sensor_id=${sensor_id}&time_from=${time_from}&time_to=${time_to}`,
    {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
    }
  );

  if (!res.ok) {
    throw new Error(
      `Failed to fetch locations: ${res.status} ${res.statusText}`
    );
  }
  const body: IRes<ISensorInfo[]> = await res.json();
  return body.data;
};
```

Та відповідні інтерфейси для них:

src/interfaces/sensors.tsx

```
export interface ISensors {
  code: string;
  fw_version: string;
  id: number;
  location_id: number;
```

```
  serial: string;
  type: string;
}

export interface ISensorInfo {
  co_ppm: number;
  humidity_pct: number;
  light_lux: number;
  no2_ppb: number;
  sensor_id: number;
  temperature_c: number;
  ts: string;
  voltage_v: number;
}
```

src/interfaces/locations.tsx

```
export interface ILocation {
  description: string;
  id: number;
  name: string;
}
```

src/interfaces/response.tsx

```
export interface IRes<T> {
  data: T;
}
```

4. Продемонструвати роботу веб застосунку продемонструвавши вкладку network в DevTools. Та відображення вимірювальної інфомації.

5. Закомітити зміни та запушити на GitHub, сформувати PR описати його(що було виконано) та дочекатись схвалення викладача на merge. Надіслати на пошту звіт і посилання на PR.

8.3 Зміст звіту

1. Найменування і мета роботи;
2. Код по кожному пункту порядку виконання роботи;
3. Результати роботи по кожному пункту виконання роботи;
4. Висновки.

8.4 Контрольні запитання

1. Чим відрізняється path (/products/1) від query string (?page=2&limit=20)?
2. У яких випадках логічно використовувати path-параметр, а в яких – query-параметр?
3. Як кодуються спеціальні символи у query-рядку (пробіли, українські букви тощо)?
4. Чому важливо перевіряти та валідовувати значення query-параметрів (тип, діапазон, обов’язковість) на бекенді? Наведи можливі наслідки відсутності такої валідації.

Додаток 1

```
import sqlite3
from pathlib import Path

USERS_DB = Path("backend/app/db/users.db")

MEAS_DB = Path("backend/app/db/measurement.sqlite")

def main():
    conn = sqlite3.connect(USERS_DB)
    cur = conn.cursor()
```

```
    cur.execute("ATTACH DATABASE ? AS meas", (str(MEAS_DB),))

    tables = cur.execute(
        """
        SELECT name, sql
        FROM meas.sqlite_master
        WHERE type = 'table'
          AND name NOT LIKE 'sqlite_%';
        """
    ).fetchall()

    for name, create_sql in tables:
        print(f"Обробляю таблицю: {name!r}")

        exists = cur.execute(
            """
            SELECT 1
            FROM sqlite_master
            WHERE type = 'table' AND name = ?;
            """,
            (name,),
        ).fetchone()

        if not exists:
            if create_sql:
                print(f"  Створюю таблицю {name!r} в users.db")
                cur.execute(create_sql)
        else:
            print(f"  Таблиця {name!r} вже існує в users.db, пропускаю CREATE TABLE")

        try:
            print(f"  Копіюю дані з meas.{name} у {name}")
            cur.execute(f"INSERT INTO {name} SELECT * FROM meas.{name};")
        except sqlite3.Error as e:
            print(f"  ПОМИЛКА при копіюванні таблиці {name}: {e}")

    conn.commit()

    cur.execute("DETACH DATABASE meas")
    conn.close()

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

Додаток 2


Dashboard
A quick overview of your measurements.
Measurements
One flexible chart. Toggle metrics and change filters to explore data.
Location
Server Room
Sensor
S006
From
2025-10-27 14:30:00
To
2025-10-30 14:30:00
Apply
Temperature, °C
Humidity, %
Voltage, V
CO, ppm
Light, lux
NO₂, ppb
80
60
40
20
0
2025-10-27 14:30:00
2025-10-28 09:10:00
2025-10-28 09:50:00
2025-10-28 19:30:00
2025-10-29 05:10:00
2025-10-29 14:50:00
2025-10-30 00:30:00
2025-10-30 14:30:00
Humidity, %
NO₂, ppb
Temperature, °C