JEGYZŐKÖNYV

Web Technológiák 2.

Féléves feladat

Okos Otthon Biztonsági Kérdéseinek Vizsgálata

Készítette: **Kató András**

Neptunkód: **S7KTW0**

Dátum: 2025.05.17.

Tartalomjegyzék

1. Bevezetés

A féléves feladatom egy okos otthon rendszer biztonsági kérdéseinek a vizsgálatával foglalkozik. Az ilyen rendszerek IoT eszközökből és azok kapcsolataiból állnak. Mint minden internetképes eszköz, ezek is rendelkeznek különböző sebezhetőségekkel, illetve biztonsági résekkel, melyek kiszolgáltatottá tehetik a felhasználót a támadókkal szemben. Az intelligens otthonok egyre nagyobb népszerűségre tesznek szert. Ez többek között köszönhető a technológia rohamos fejlődésének, valamint annak az ideálnak, melyben az otthununk egy kényelmes és biztonságos szerepet tölt be az életünkben. Az otthonok bevonásával az internet világába számos előnyre tehetünk szert. Ezek közé tartozik az energiahatékonyság, illetve a kényelem, komfortszint növelése. Az okosotthon rendszerek népszerűsége az elmúlt idő-szakban ugrásszerűen megnőtt, és ezzel arányosan a rosszindulatú támadók száma is. Az ökoszisztémánk tartalmazni fog egy szimulált rendszert, melynek segítségével vizsgálatokat fogunk végezni.

2. A feladat leírása

Az okosotthon vizsgálatához több egységre is szükség van. A rendszer elemzi az otthoni eszközök és szenzorok naplófájljait, hogy kiszűrje a sebezhetőségeket, feltérképezze a lehetséges problémákat. A Java-alapú modul intelligensen összeveti a bejövő logokat, és egy áttekinthető, könnyen feldolgozható summary.json fájlban foglalja össze a legfontosabb részleteket, anomáliákat és javasolt teendőket.

Ezután a Node.js szerver gondoskodik arról, hogy a kapott összefoglaló a böngészőben is azonnal elérhető legyen: egy letisztult webfelületen elérhetőek az adatok, a legutóbbi riasztásokkal, könnyedén nyomon lehet követni az otthon biztonsági állapotát, és interaktív vizualizációkon segítik az eligazodást. Mindez egyetlen weboldalon, felhasználóbarát, átlátható formában.

A főoldalon három nagy kártya pillanatok alatt megmutatja, hány eszköz van összesen, közülük mennyi van offline állapotban, illetve mennyi merült le. Alatta bal oldalt egy keresősáv segítségével szűrhetőek az eszközök név alapján, és egy táblázatban láthatjuk a státuszukat és kockázati pontszámukat. Egy vizuális hálózatábrán a központi egységtől kiindulva szemléltetve láthatjuk, hogy hogyan kapcsolódnak egymáshoz a különböző eszközök, míg egy oszlopdiagramon azt követhetjük nyomon, hogy milyen technológiával csatlakoznak az egyes eszközök, technológiákra bontva. Végül egy magyarázó panel részletesen írja le, mit jelentenek a pontszámok, és milyen események számítanak kockázatosnak.

3. Alkalmazott technológiák

A programom több részből áll, amelyeket négy különböző pontba tudom sorolni működésük és technológiájuk alapján:

- A háttérben egy Eclipse IDE-ben fejlesztett, **Java** nyelvű alkalmazás dolgozik: ez gyűjti össze és elemzi az okosotthon eszközeinek naplófájljait, majd a lényeges adatokat egy summary.json nevű fájlba rendezi.
- Ez a **summary.json** az a híd, ami átvezeti az adatokat a Java-modultól a webes felületig. A fájl tulajdonképpen egy egyszerű, ember- és gépileg is jól olvasható formátum: benne vannak a készülékek neve, állapota és minden olyan részletet, amire a böngészőben futó felületnek szüksége van a megjelenítéshez.
- A szerveroldalt egy **Node.js** környezet szolgáltatja, amit fejlesztés közben a nodemon segítségével indítunk. Így elég menteni a kódot, és a szerver automatikusan újraindul.

A server.js fájl gondoskodik arról, hogy a summary.json mindig elérhető legyen végpontként.

- Végül az **index.html** maga az a felhasználói felület, ahol a böngészőben valós időben mutatja az okosotthon biztonsági állapotát. A beépített JavaScript kóddal beolvassa a summary.json tartalmát, felvázolja a kártyákat, táblázatokat, hálózati ábrákat és diagramokat; mindezt egy letisztult, jelszóval védett oldalon.

4. A program működése

A többkomponensű alkalmazás két jól elkülöníthető rétegre épül: egy Java-alapú feldolgozó modulra és egy Node.js alapú webszerverre. Az első rétegben, Eclipse fejlesztőkörnyezetben készült Java-kód végzi az adatok összegyűjtését és előkészítését, majd az eredményt summary.json formátumban exportálja. Ez a JSON-fájl tartalmaz minden olyan információt, amely a felhasználói felületen megjelenő tartalmak alapját képezi.

Az alkalmazás biztonsági rétegként felhasználónév–jelszó alapú beleptetést alkalmaz, így csak hitelesített felhasználók férhetnek hozzá a tartalmakhoz. A sikeres belépést követően a kliens betölti az index.html oldalt

5. A programkód bemutatása

A programkód bemutatása következik ebben a fejezetben. A nagyobb átláthatóság érdekében képeket is fogok csatolni, így a szöveges leírás mellet párhuzamosan megtekinthető lesz a kód is.

Node.js telepítése, projekt létrehozása

A Node.js-t rendszerszinten telepítjük, és egy-egy projektben a package.json segítségével tudjuk meghívni. A https://nodejs.org/en oldalról letölthető futtatható telepítővel, elég a telepítő útmutatását követni. Ha ez kész van, a terminálban a projekt mappájába navigálva a következő parancsot futtatva tudjuk létrehozni a package.json-t: „npm init -y”.

Miután ez kész van, projektfüggőségként telepíthetünk bármit, például magát az Express könyvtárt vagy a nodemont is: „npm install express”; illetve „npm install --save-dev nodemon”. Ezek után a node_modules mappában landolnak a csomagok, és a package.json-ban, valamint a package-lock.json-ban követve lesz minden verzió.

A szerver elindítása (server.js; package.js; nodemon használata)

```
const express = require('express');
const path = require('path');
const app = express();
const PORT = 3000;

//public mappa elemeinek elerese
app.use(express.static(path.join(__dirname, 'public')));


app.listen(PORT, () => {
  console.log(`Az Okosotthon Áttekintés weblap elérhető: http://localhost:${PORT}`);
});

```

A server.js egy egyszerű Express-alapú szerver, ami a public mappában található fájlokat statikusan szolgálja ki. Amikor elindítjuk parancsorból a nodemon server.js parancsot, a szerver a 3000-es porton jeleníti meg a weblapunkat, és a konzolon kiírja, hogy az áttekintő oldal elérhető a http://localhost:3000 címen.

```
C:\Users\Andris\Desktop\SzakdogaProgram\smartHomeProfile>nodemon server.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Az Okosotthon Áttekintés weblap elérhető: http://localhost:3000
```

A nodemon használatához a project mappájában telepíteni kell először a „npm install --save-dev nodemon” parancsal. A nodemon figyelni fogja a fájlokat, újraindítva a szervert minden mentés után, így igen hasznos tud lenni programfejlesztés alatt. Fontos kiemelni, hogy a package.json fájlban is meg kell adnunk a nodemon függőséget:

```
{
  "name": "smarthomeSecurity",
  "version": "1.0.0",

  "main": "index.js",
  "keywords": [],
  "author": "",
  "license": "ISC",

  "dependencies":
  {
    "express": "^4.21.2"
  },

  "devDependencies": {
    "nodemon": "^3.1.10"
  }
}
```

Ezek után a nodemon server.js paranccsal könnyedén tudjuk futtatni a szerverünket és minden módosítás után újra fogja indítani a Node.js szerverünket.

A summary.json tartalma

Olvashatóra formázás ☑


```
[
  {
    "name": "Mi Wireless Switch",
    "connection_type": "ZigBee",
    "appearances": 24,
    "firstSeen": "2025-01-17T00:00:00",
    "lastSeen": "2025-01-17T23:00:00",
    "lastStatus": "Online",
    "firmwareVersion": "1.9.6"
  },
  {
    "name": "Mi Motion Sensor",
    "connection_type": "ZigBee",
    "appearances": 24,
    "firstSeen": "2025-01-17T00:00:00",
    "drainedAt": "2025-01-17T21:00:00",
    "lastStatus": "Offline",
    "firmwareVersion": "2.25.00"
  },
  {
    "name": "Xiaomi Smart Plug 2",
    "connection_type": "Wi-Fi IEEE 802.11, Bluetooth 5.0",
    "appearances": 24,
    "firstSeen": "2025-01-17T00:00:00",
    "lastSeen": "2025-01-17T23:00:00",
    "lastStatus": "Online",
    "firmwareVersion": "2.1.17",
    "ipAddress": "192.168.0.6",
    "macBlueAddress": "D4:3A:BC:12:34:5A"
  },
  {
    "name": "Xiaomi LED Desk Lamp 1S",
    "connection_type": "Wi-Fi IEEE 802.11",
    "appearances": 24,
    "firstSeen": "2025-01-17T00:00:00",
    "lastSeen": "2025-01-17T23:00:00",
    "lastStatus": "Online",
    "firmwareVersion": "11.0.0",
    "ipAddress": "192.168.0.4",
    "macBlueAddress": "D4:3A:BC:12:34:58"
  },
  {
    "name": "Xiaomi Smart Doorbell 3",
    "connection_type": "Wi-Fi IEEE 802",
    "appearances": 24,
    "firstSeen": "2025-01-17T00:00:00",
    "lastSeen": "2025-01-17T23:00:00",
    "lastStatus": "Online",
    "firmwareVersion": "3.0.16",
    "ipAddress": "192.168.0.3",
    "macBlueAddress": "D4:3A:BC:12:34:57"
  },
```

A summary.json az összegzett adatok, melyek megmutatják az eszközök kinyert adatait: benne van minden eszköz neve, típusa és az, hogy épp online vagy offline állapotban van-e. Egy nagy naplófájl, ami röviden összefoglalja, hogy az egyes szenzorok és kiegészítők mikor bukkantak fel először és legutóbb a hálózatban, illetve ha valamelyik lemerült, azt is mutatja.

A fájl formátuma egyszerű, átlátható JSON: minden eszköz egy tárgy (object), ahol a „name” kulcsnál az eszköz nevét látjuk (például „Mi Motion Sensor” vagy „Xiaomi Smart Plug 2”), a „connection_type” alatt pedig azt, hogy ZigBee, Wi-Fi vagy Bluetooth kapcsolaton keresztül kommunikál. Így a felület könnyen csoportosíthatja és szűrheti az eszközöket kapcsolat szerint.

Az „appearances” mező megmutatja, hányszor jelentkezett az eszköz az adott időszakban, a „firstSeen” és „lastSeen” mezők pedig percekre pontosan rögzítik, mikor indult el és mikor hagyta el utoljára a hálózatot.

Ha valamelyik eszköz lemerült, a „drainedAt” pontos idejét is feltünteti a fájl, ezzel segítve a gyorsabb karbantartást és az akkumulátor-csere ütemezését. A firmware-verziók (firmwareVersion) és az IP- vagy MAC-címek (ipAddress; macBlueAddress) pedig lehetőséget adnak arra, hogy a követés és frissítés még precízebb legyen.

Az index.html bemutatása

```
<!DOCTYPE html>
<html lang="hu">
    <head>
        <meta charset="UTF-8" />
        <title>Okosotthon Áttekintés</title>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/vis/4.21.0/vis.min.js"></script>
        <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
        <link href="https://cdnjs.cloudflare.com/ajax/libs/vis/4.21.0/vis-network.min.css" rel="stylesheet" />
```

A fájl elején a head részben történik a metaadatok (UTF-8 karakterkódolás), a dokumentum címe (Okosotthon Áttekintés), valamint a külső könyvtárak betöltése: a **vis.js** a hálózati gráfokhoz, a **Chart.js** az oszlopdiagramhoz, és a vis-stíluslap a hálózati ábra formázásához, azonban ennek használata nem feltétlenül szükséges.

A style blokkban található szabályok határozzák meg az oldal kinézetét: a világos háttér- és kontrasztos színpaletta, a rácsos elrendezés (grid), valamint a kártyák, táblázatok és bejelentkező ablakok árnyékai és lekerekített sarkai adják az egységes, barátságos megjelenést. A bejelentkező felugró ablak (modal) takarja el a fő tartalmat, amíg nem lépünk be. A HTML-ben egyszerű bejelentkezési mezők (felhasználónév; jelszó) és egy belépés gomb található

benne, a hozzájuk tartozó JS ellenőrzi a felhasználónév és jelszó párost, és ha stimmel, eltünteti a modált.

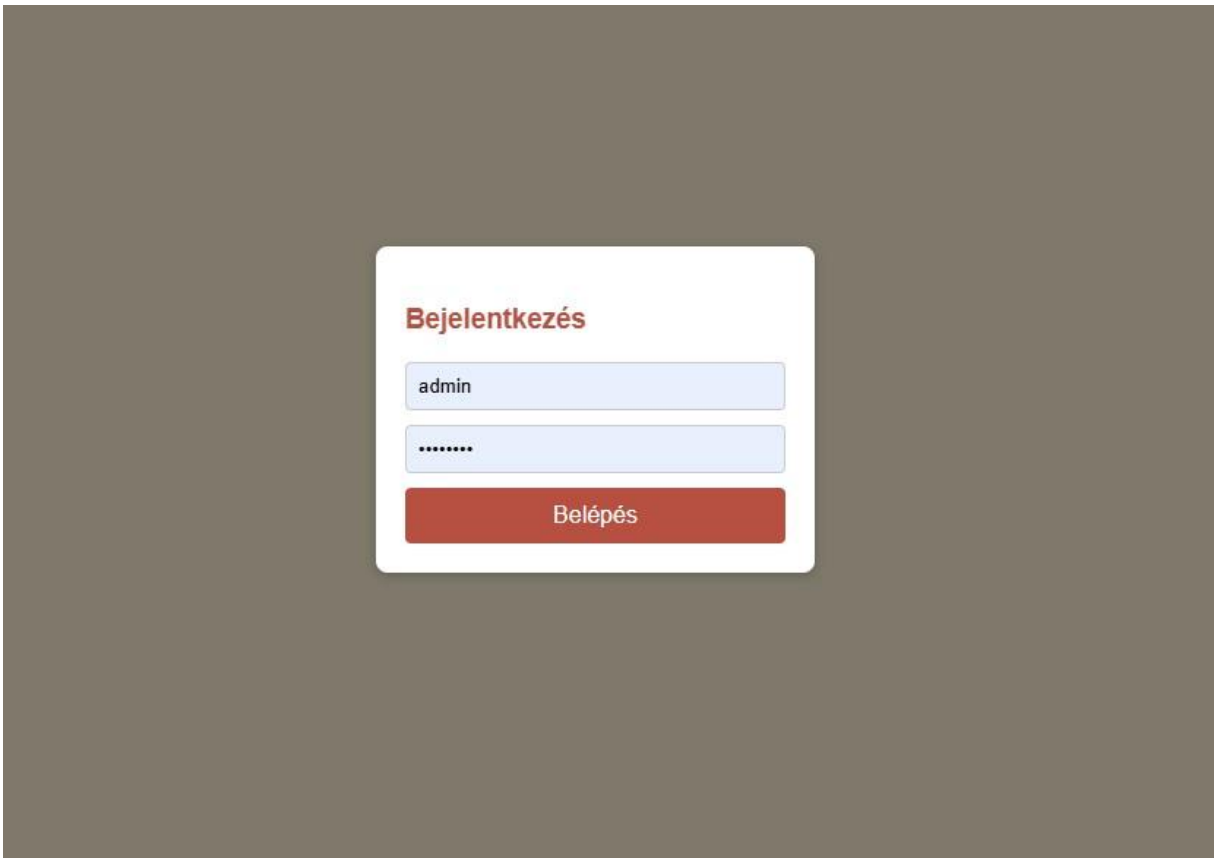
```
<style>
    * {box-sizing: border-box;}

    body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #FFF2D7;
    color: #B85042;
    overflow: hidden;
    }

    #login-modal {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: rgba(0, 0, 0, 0.5);
    display: flex;
    align-items: center;
    justify-content: center;
    z-index: 1000;
    }

    #login-modal .modal-content {
    background: #fff;
    padding: 20px;
    border-radius: 8px;
    width: 300px;
    box-shadow: 0 2px 8px rgba(0, 0, 0, 0.2);
    }

    #login-modal input {
    width: 100%;
    margin-bottom: 10px;
    padding: 8px;
    border: 1px solid #ccc;
    border-radius: 4px;
```



A **KPI (Key Performance Indicator) kártyák** három, egymás mellé rendezett kártya, melyek egy pillantás alatt bemutatják a legfontosabb számadatokat: összes eszköz, offline eszközök és lemerült eszközök száma. A fehér háttér és a finom árnyék kiemeli őket a világos borításból, a lekerekített sarkok és a lebegő animáció (hover) pedig interaktivitást és esztétikát kölcsönöz nekik.

Az **eszközlista és kereső** Az oldalsávon egy egyszerű szövegmező segítségével bármikor kereshetünk eszköznevekre, és a lentebb található táblázat azonnal szűri a találatokat. Maga a táblázat három oszlopban sorolja fel az eszköz nevét, aktuális státuszát (online, offline vagy lemerült), valamint a kiszámolt kockázati pontszámot. A pontszám szerint rábökve növekvő vagy csökkenő sorrendbe rendezhető a lista.

Az **interaktív hálózati ábra (vis.js)** egy physics-el rendelkező, mozgatásra is képes gráf mutatja, hogyan kapcsolódnak egymáshoz az egyes eszközök. A csomópontok színe vizuális állapotjelző: zöld, ha minden rendben, piros, ha offline vagy lemerült a készülék. A fizika-alapú elrendezés és a rugalmas csomópontok lehetővé teszik, hogy saját igény szerint rendezzük át a hálózatot.

```
<body>

    <div id="login-modal">
        <div class="modal-content">
            <h3>Bejelentkezés</h3>
            <div id="login-error"></div>
            <input type="text" id="login-user" placeholder="Felhasználónév" />
            <input type="password" id="login-pass" placeholder="Jelszó" />
            <button id="login-btn">Belépés</button>
        </div>
    </div>


    <div id="app">


        <h1>Okosotthon Áttekintés</h1>

        <div class="controls">
        <button id="refresh">Frissítés</button>
        </div>

        <div class="dashboard">


            <div class="kpi-cards">
                <div class="card">
                    <h2>Összes eszköz</h2>
                    <p id="kpi-total">0</p>
                </div>
                <div class="card">
                    <h2>Offline eszközök</h2>
                    <p id="kpi-offline">0</p>
                </div>
                <div class="card">
                    <h2>Lemerült eszközök</h2>
                    <p id="kpi-drained">0</p>
                </div>
            </div>


            <div class="sidebar">
                <h3>Eszköz keresése</h3>
                <input type="text" id="filter-input" placeholder="Név szűrése..." />
                <table>
                    <thead>
                        <tr>
                            <th>Név</th>
                            <th>Státusz</th>
                            <th id="th-risk">Kockázat (pont)</th>
                        </tr>
                    </thead>
                    <tbody id="device-table"></tbody>
                </table>
            </div>
```

A **kapcsolódási technológia diagram** (Chart.js) egy egyszerű oszlopdiagram, mely szemlélteti, hány eszköz használ Wi-Fi-t, ZigBee-t, Bluetooth 4.x-et vagy 5.0-t. A diagram fölé vitt kurzor tooltipként listázza az adott kategóriába tartozó eszközöket, így nemcsak mennyiségi, de minőségi rálátást is ad a hálózat használatra.

És végül a **magyarázó doboz** táblázat és diagram alatti panel rövid, de lényegre törő leírást ad a kockázati pontok számításáról (offline +20 pont, Bluetooth 4.x +10 pont) és a gráf

színkódjairól (piros = lemerült). Ez a statikus magyarázat segít, hogy bárki gyorsan megértse, mi alapján történik az értékelés, és mi mit jelent a felületen, anélkül hogy mélyebben ismernie kellene a háttérben futó logikáját a programnak.

```
<script>
  const loginModal = document.getElementById('login-modal');
  const loginBtn = document.getElementById('login-btn');
  const errDiv = document.getElementById('login-error');
  const appDiv = document.getElementById('app');

  const showApp = () => {
    loginModal.style.display = 'none';
    document.body.style.overflow = 'auto';
    appDiv.style.display = 'block';
    initApp();
  };

  loginBtn.addEventListener('click', () => {
    const u = document.getElementById('login-user').value;
    const p = document.getElementById('login-pass').value;
    if (u === 'admin' && p === '12345678') showApp();
    else errDiv.textContent = 'Hibás felhasználónév vagy jelszó';
  });

  function initApp() {
    const explanationBox = document.querySelector('.explanation');
    const updateTimestamp = () => {
      const now = new Date();
      const ts = now.toLocaleString('hu-HU', { dateStyle:'short', timeStyle:'medium' });
      if (!explanationBox.querySelector('em')) {
        explanationBox.innerHTML = `<p><em>Utolsó frissítés: ${ts}</em></p>` + explanationBox.innerHTML;
      }
    };
    document.getElementById('refresh').addEventListener('click', () => { updateTimestamp(); window.location.reload(); });
    updateTimestamp();
```

A Bejelentkezés megvalósításához lekérjük a főbb elemeket: a bejelentkező modált (loginModal), a belépés gombot (loginBtn), a hibaüzenet megjelenítőjét (errDiv) és a fő alkalmazás konténerét (appDiv). A belépés gomb eseménykezelője ellenőrzi, hogy a „Felhasználónév” mező értéke 'admin', a „Jelszó” pedig '12345678'–e. Ha stimmel, meghívja a showApp() függvényt, ami elrejti a modált, engedélyezi az oldalsó görgetést és kirajzolja a fő felületet.

Ezután következik az alkalmazás indítása, a showApp() egyszerűen eltünteti a bejelentkező ablakot, és megjeleníti a #app tartalmat, majd meghívja az initApp() függvényt. Ez utóbbi a későbbi összes inicializációs lépést végrehajtja: frissíti az időbélyeget, kezeli a „Frissítés” gombot, és elindítja az adatbetöltést.

Az updateTimestamp() minden híváskor lekéri a jelenlegi dátumot és időt a böngésző beállításainak megfelelő magyar formátumban, majd beilleszti az .explanation panel elejére egy szövegként.

A Frissítés gombhoz egy kattintás esemény van kötve, ami először frissíti az időbélyeget, majd újratölti az oldalt (window.location.reload()), ezzel garantálva, hogy a legfrissebb summary.json kerül betöltésre.

```
fetch('/summary.json')
  .then(res => res.json())
  .then(devices => {

    document.getElementById('kpi-total').textContent =
      devices.length;
    document.getElementById('kpi-offline').textContent =
      devices.filter(d=>d.lastStatus==='Offline').length;
    document.getElementById('kpi-drained').textContent =
      devices.filter(d=>d.drainedAt!=null).length;


    devices.forEach(d=>{
      let score = 0;
      if (d.lastStatus==='Offline') score += 20;
      if (d.connection_type.toLowerCase().includes('bluetooth 4.')) score += 10;
      d.risk = score;
    });


    let sortAsc = false;
    const thRisk = document.getElementById('th-risk');
    thRisk.addEventListener('click', ()=>{
      sortAsc = !sortAsc;
      renderTable(filterInput.value);
      thRisk.classList.toggle('sort-asc',  sortAsc);
      thRisk.classList.toggle('sort-desc', !sortAsc);
    });

    const filterInput = document.getElementById('filter-input');
    const tableBody   = document.getElementById('device-table');
    function renderTable(filter='') {

      const shown = devices
        .filter(d=>d.name.toLowerCase().includes(filter.toLowerCase()))
        .sort((a,b)=>
          sortAsc
            ? a.risk - b.risk
            : b.risk - a.risk
        );
      tableBody.innerHTML = shown.map((d,i)=>`
        <tr data-idx="${devices.indexOf(d)}">
          <td>${d.name}</td>
          <td>${d.lastStatus}${d.drainedAt?' (lemerült)':''}</td>
          <td>${d.risk}</td>
        </tr>
      `).join('');
    }
    renderTable();
    filterInput.addEventListener('input', e=>renderTable(e.target.value));

    thRisk.classList.add('sort-desc');
```

A fetch('/summary.json') hívás után JSON-ként dolgozzuk fel a válaszadathalmazt, majd kiszámoljuk az alapvető mutatókat: összes eszköz (devices.length), offline állapotúak száma, illetve az lemerült eszközök száma. Ezeket fogjuk megjeleníteni a három KPI-kártyán.

Minden eszközhöz végig iterálva létrehozunk egy risk, vagyis kockázat mezőt: ha a készülék offline, +20 pontot kap; ha Bluetooth 4.x kapcsolaton kommunikál, további +10 pont jár. Ezzel a lépéssel minden tételhez egy egyszerű, de könnyen értelmezhető kockázati érték társul. A renderTable(filter) függvény a felhasználó által beírt szűrőszöveg (filterInput.value) alapján leszűri az eszközöket név szerint, majd a sortAsc változó állapota szerint növekvő vagy

csökkenő kockázat szerint rendezi. Az így kapott sorokból építi újjá a body tartalmát. A „Kockázat” oszlop fejlécére kattintva váltható a rendezési irány.

```
    renderTable();
    filterInput.addEventListener('input', e=>renderTable(e.target.value));

    thRisk.classList.add('sort-desc');

    tableBody.addEventListener('click', e=>{
      const tr = e.target.closest('tr');
      if (!tr) return;
      const idx = +tr.dataset.idx;
      network.selectNodes([idx]);
      network.focus(idx, {animation:{duration:300}});
    });

    const nodes=[], edges=[];
    let hubId=null;
    devices.forEach((dev,i)=>{
      const color= dev.drainedAt
                   ?'red'
                   :(dev.lastStatus==='Offline'?'red':'green');
      nodes.push({id:i,label:dev.name,color,font:{multi:true}});
      if(dev.name==='Smart Home Hub 2') hubId=i;
    });
    if(hubId!==null) nodes.forEach((_,i)=>{
      if(i!==hubId) edges.push({from:hubId,to:i});
    });
    const network = new vis.Network(
document.getElementById('network'),
{ nodes: new vis.DataSet(nodes), edges: new vis.DataSet(edges) },
{
layout: { improvedLayout: true },

physics: {
 enabled:        true,
 repulsion:      { nodeDistance: 200 },
 barnesHut:      { springLength: 200, springConstant: 0.05, damping: 0.09 },
 stabilization:  { enabled: true, iterations: 1000, fit: false }
},

interaction: { dragView: true, dragNodes: true },
nodes: { shape: 'dot', size: 16 }
}
);


network.once('stabilizationIterationsDone', () => {
network.fit({ animation: false });

network.setOptions({ physics: true });
});

    const stats = {
      'Wi-Fi': {count:0,names:[]},
      'ZigBee': {count:0,names:[]},
      'Bluetooth 4.x': {count:0,names:[]},
      'Bluetooth 5.0': {count:0,names:[]}
    };
    devices.forEach(d => {
const ct = d.connection_type.toLowerCase();
```

```
            if (/\bwi[-\u2010\u2011]?fi\b/.test(ct)) {
            stats['Wi-Fi'].count++;
            stats['Wi-Fi'].names.push(d.name);
            }

            if (/\bzigbee\b/.test(ct)) {
            stats['ZigBee'].count++;
            stats['ZigBee'].names.push(d.name);
            }

            if (/bluetooth\s*4\b/.test(ct)) {
            stats['Bluetooth 4.x'].count++;
            stats['Bluetooth 4.x'].names.push(d.name);
            }

            if (/bluetooth\s*5\b/.test(ct)) {
            stats['Bluetooth 5.0'].count++;
            stats['Bluetooth 5.0'].names.push(d.name);
            }
            });


                const ctx=document.getElementById('connChart').getContext('2d');
                new Chart(ctx,{
                  type:'bar',
                  data:{
                    labels:Object.keys(stats),
                    datasets:[{
                      data:Object.values(stats).map(s=>s.count),
                      backgroundColor:['green','green','red','green']
                    }]
                  },
                  options:{
                    responsive:true,
                    plugins:{
                      title:{
                        display:true,
                        text:'Kapcsolódási technológia'
                      },
            tooltip: {
             callbacks: {
            title: () => 'Eszközök',
            label: ctx => {

              return stats[ctx.label].names.map(name => `• ${name}`);
            }
             }
            },
                      legend:{display:false}
                    },
                    scales:{
                      y:{ beginAtZero:true, ticks:{precision:0} }
                    }
                  }
                });
              })
              .catch(console.error);

        }
    </script>
</body>
```

A nodes és edges tömböt feltöltjük: minden eszköz egy csomópont, piros színnel jelölve az offline vagy lemerült állapotot, a „Smart Home Hub 2” pedig központi hubként definiálja az összeköttetéseket. A vis.Network objektum kirajzolja az interaktív gráfot, ahol a felhasználó mozgatni tudja a csomópontokat, és rá is tud közelíteni.

Összesítjük a különböző kapcsolatokat (Wi-Fi, ZigBee, Bluetooth 4.x és 5.0) egy stats objektumban, majd a Chart.js segítségével egy oszlopdiagramot készítünk. A tooltipekben az adott kategóriába tartozó eszközök nevét listázza.

6. Összefoglalás

Ez a projekt egy okosotthon biztonsági áttekintő felületet valósít meg. A Java-modul feldolgozza és elemzi a különböző eszközök naplófájljait, majd egy könnyen értelmezhető summary.json fájlba gyűjti össze az aktuális állapotot. Ezt követően egy Node.js alapú szerver (nodemonnal indítva) szolgálja ki az index.html oldalt, amin keresztül jelszóval védetten egy letisztult áttekintő felületen keresztül nézhetjük meg a rendszer legfontosabb mutatóit.

A böngészőben három KPI-kártyán látjuk az összes, az offline és a lemerült eszközök számát, egy oldalsávban név szerinti keresővel és rendezhető táblázatban böngészhetjük a komponenseket, a vis.js hálózati gráf pedig interaktív módon mutatja a kapcsolódásokat és a státuszokat. Végül a Chart.js oszlopdiagramja a különböző kommunikációs technológiák (Wi-Fi, ZigBee, Bluetooth) eloszlását jeleníti meg, míg egy magyarázó doboz gyorsan áttekinthetővé teszi a pontszámítás szabályait és az ábra működését.