Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Általános Informatikai Intézeti Tanszék

Elosztott hálózat-felügyeleti rendszerek

Adatgyűjtő ágens fejlesztése

Szakdolgozat

Készítette: Szabó Alex Gyula
Neptun-kód: AS97C7
Cím: 3580 Tiszaújváros, Barcsay Jenő tér 1. 2/3.

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar

Általános Informatikai Intézeti Tanszék
3515 Miskolc-Egyetemváros

Szak: **Mérnökinformatikus**

Szakirány: Korszerű WEB technológiák

Szakdolgozat azonosító:
IAL/AS97C7/BSc/2020
Intézmény azonosító: FI 87515

S Z A K D O L G O Z A T F E L A D A T

Szabó Alex Gyula
BSc mérnökinformatikus jelölt részére

A tervezés tárgyköre: **Számítógép-hálózatok**

A feladat címe: **Elosztott hálózat-felügyeleti rendszerek Adatgyűjtő ágens fejlesztése**

A feladat részletezése:

- *Tanulmányozza a számítógép-hálózatok elosztott felügyeletének problémakörét.*
- *Mutasson be néhány, a gyakorlatban elterjedtebb felügyeleti rendszert, az alkalmazott struktúrákat és protokollokat.*
- *Készítsen tervet egy elosztott SNMP hálózat-felügyeleti rendszer kialakítására, mely alkalmas önálló adatgyűjtésre és ezen adatoknak egy központi alkalmazás felé való továbbítására.*
- *Implementálja ezen adatgyűjtő alkalmazás egyes részleteit, végezzen teszteket a rendszer hatékonyságáról és tegyen javaslatot fejlesztési lehetőségre.*

Tervezésvezető: Dr. Kovács Szilveszter	Tanszék, beosztás: ME-GEIAL, egyetemi docens
Konzulens: Almseidin Mohammad Abdallah Suleiman, PhD	Cég, beosztás: Princess Sumaya University for Technology, assistant professor
A szakdolgozat kiadásának időpontja:	2020. február 17.
A szakdolgozat beadásának határideje:	2020. május 15.

Miskolc, 2020. február 17.

Dr. Kovács László
tanszékvezető, egyetemi docens

1. A szakmai gyakorlat helye: _____

2. A szakmai gyakorlat vezetőjének neve: _____

3. A szakdolgozat módosítása: szükséges (a módosítást külön lap tartalmazza)
   nem szükséges (a megfelelő rész aláhúzandó)

   Miskolc, _____

   tervezésvezető aláírása

4. A tervezést ellenőriztem: (1) _____

   (2) _____

   (3) _____

   (4) _____

   dátum, tervezésvezető aláírása

5. A szakdolgozat beadható

   nem adható be

   Miskolc, _____

   konzulens aláírása tervezésvezető aláírása

6. A szakdolgozat ….. szövegoldalt,
   ….. db rajzot,
   ….. db CD mellékletet
   ….. egyéb mellékletet tartalmaz.

7. A szakdolgozat bírálatra: bocsátható

   nem bocsátható

   A bíráló neve, címe: _____

   Miskolc, _____

   tanszékvezető aláírása

8. Osztályzat: a bíráló javaslata: _____

   a tanszék javaslata: _____

   a Záróvizsga Bizottság döntése: _____

   Miskolc, _____

   a Záróvizsga Bizottság elnökének aláírása

EREDETISÉGI NYILATKOZAT

Alulírott *Szabó Alex Gyula*; Neptun-kód: *AS97C7*, a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős *mérnökinformatikus BSc* szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy az

„*Elosztott hálózat-felügyeleti rendszerek – Adatgyűjtő ágens fejlesztése*”

című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, 2020. június 30. ……..…

Hallgató

Tartalomjegyzék

1. Bevezetés

Az emberi kapcsolattartás, kommunikáció iránti vágy alapvető tulajdonságunk, melynek lehetőségei, formái folyamatosan változtak és fejlődtek. A technológiai újítások lehetővé tették, hogy az egykor csak személyes kapcsolattartást évtizedek alatt a robbanásszerűen fejlődő társadalmakkal lépést tartva alakítsuk, azok elvárásai szerint. Napjainkra a kommunikációnk határait oly mértékben kiszélesítettük, hogy a Föld bármely pontján képesek vagyunk kapcsolatban maradni akár szóban, akár írásban. Ennek az átalakulásnak elengedhetetlen lépése volt az adathálózatok létrejötte, ezek egymással történő összekapcsolása, valamint a kapcsolatok megbízhatóságának rohamos javulása. Az adathálózatok kezdetben mindössze néhány, katonai vagy tudományos célokra felhasznált, akkoriban nagy számítási kapacitásúnak tekinthető gépeket kapcsoltak össze. Nőtt az igény a személyi számítógépek és azok összekötésének lehetősége iránt. Ezen igény nyomására alakult ki szinte az egész világot lefedő, összekapcsolt hálózat, az Internet.

Manapság az Internetre csatlakoztatott eszközök száma megközelíti a 30 milliárdot, mely 2025-re becslések szerint eléri a 75 milliárdot. Kialakulóban van az IoT (Internet of Things) világa, hiszen lassan minden elképzelhető használati eszköz „okossá” válik. Bárki vásárolhat hálózaton vezérelt mosógépet, önmagától rendelést leadó hűtőgépet vagy akár a forgalom többi résztvevőjével összeköttetésben álló autót is, csak hogy néhány példát említsek. Mindennek természetesen velejáró következménye az is, hogy ezeket az exponenciálisan terjeszkedő kapcsolatokat egyre többen próbálják feltörni információlopás, vagy gyanútlan áldozatok eszközeinek illegális tevékenységekre történő felhasználásának céljából. Emiatt ezeket a hálózatokat szigorú felügyelettel és védelemmel kell ellátni, amelynek ugyanolyan ütemű fejlesztésére van szükség, mint az általuk védett rendszereknek.

Az alapvető megoldandó probléma a felügyelet, amely az összetettség növekedése miatt egyre nehezebb feladat. Enélkül a hálózati hibák, leállások sokkal tovább maradnának észrevétlenek, ami a mindennapi életünkhöz létfontosságú rendszerek esetén nem elfogadható. A megfigyelés növeli a rendelkezésre állást, csökkenti a költségeket, valamint gyorsítja a hibajavítás folyamatát. A piacon számos hálózat-felügyeleti rendszer elérhető különböző komplexitással és képességekkel, legyen szó hardveres vagy szoftveres védelemről.

1.1 Feladat ismertetése

Dolgozatom célja a számítógép-hálózatok elosztott felügyeletének problémakörét tanulmányozni, a létező felügyeleti rendszerek legelterjedtebb megvalósítását bemutatni, valamint ismertetni az ebben használt struktúrákat és protokollokat. A megszerzett ismereteket felhasználva szeretnék terveket készíteni egy elosztott hálózat-felügyeleti alkalmazás kialakítására, mely alkalmas önálló adatgyűjtésre és ezen adatok központi egység felé való továbbítására, vagy a hálózat hibája esetén azok tárolására és újraküldésére. Ezentúl képes a hálózatban lévő, megfigyelhető eszközök automatikus felismerésére, kézi konfiguráció nélkül. Szándékom a megtervezett alkalmazás implementálása, ügyelve annak szerkezetére, letisztultságára és kódminőségére, illetve az elkészült szoftver működésének tesztelése, hatékonyságának mérése.

Feladatom témáját azért választottam, mert az egyetemi éveim alatt megszerettem a számítógép-hálózatok témakörét, amelyben szándékom mélyebb tudást szerezni. Emellett természetesen kiemelten érdekel a szoftverfejlesztés is. Ezen okok miatt született meg a fent említett feladat ötlete, hiszen együttesen igényli a hálózatok ismeretét és a szoftverfejlesztésben való jártasságot is.

2. A hálózat-felügyelet fogalma, feladata, elemei [1]

Az egyre több felhasználót kiszolgáló számítógépes rendszerekre egyre nagyobb feladat hárul, ami ahhoz vezet, hogy növekvő problémát okoz, ha a hálózatban valamelyik eszköz meghibásodik. Hiba bekövetkezése esetén gyakran annak megoldása, illetve felderítése lényegesen több időt vehet igénybe, mint a megelőzése. Mivel a mai korszerű hálózatok számottevően több hálózati eszközből épülnek fel, mint a kezdeti hálózatok, így ezeknek elengedhetetlen részei a felügyeleti rendszerek.

A felügyeleti rendszerek célja, hogy a megfelelő eszközök és módszerek segítségével csökkentsék vagy akár megelőzzék a hibákat, továbbá az ezek által okozott károkat. Ezt úgy érik el, hogy az adott hálózatot folyamatosan figyelik, lassú vagy meghibásodott komponenseket keresve, melyekről értesítik a rendszergazdákat. Az értesítés mellett képesek megfelelő részletességű és pontosságú jelentéseket biztosítani, amelyek alapján a rendszergazdák könnyebben detektálhatják a hiba forrását, valamint javíthatják a problémát.

A hálózatfelügyelet tulajdonképpen azoknak az erőforrásoknak a koordinálását és felügyeletét jelenti, amelyek a kommunikációban és a kommunikációs folyamatban részt vesznek, úgy, hogy a hálózat folyamatos és hatékony működését biztosítsa. Ez a funkcionalitás mindig valamilyen modell szerint kerül megvalósításra, melybe beletartozik minden, valamilyen módon megfigyelhető eszköz vagy objektum. A tényleges felügyelet általában egy felügyelő állomáson történik, amelyen egy magas szintű alkalmazás gyűjti, tárolja és elemzi a hálózat különböző pontjairól beérkező adatokat, egészen a berendezések állapotától az adatforgalmat jellemző paraméterekig.

Az alapján, hogy a felügyelő állomás honnan kapja az adatokat, megkülönböztetünk „ágensalapú” (*agent-based*) és „ágens nélküli” (*agentless*) rendszereket – ezek a későbbiekben kifejtésre kerülnek. A monitorozó program adatgyűjtése során az adatforgalom közvetlen megfigyelése szükséges. A hálózat működését jellemző, összegyűjtött adatokból jelentések, statisztikák készíthetők, melyeket elemezve a program képes határértékeket figyelni. Amennyiben egy előredefiniált határértéket átlép a hálózat bármely eszköze vagy része, riasztás kerül kiküldésre.

Mint az informatika bármely területén, a hálózatfelügyelő rendszerek esetében is beszélhetünk különböző elvárásokról, követelményekről:

- hálózat elérhetőségének figyelése,
- válaszidők figyelése,
- biztonsági funkciók nyújtása,
- hálózati forgalom irányítása,
- különböző helyreállítási lehetőségek.

2.1 A hálózat-felügyelet elemei [2]

Megfigyelés

A hálózatfelügyelet alapja és legfőbb feladata a rendszerben megtalálható eszközök monitorozása. Célja, hogy ezzel segítse a rendelkezésre állás idejének növelését, a hibák minél hamarabb történő felismerését és javítását. Az eszközök működési jellemzői közül szükséges legalább az elérhetőség, a késleltetés, a terheltség és a kapcsolatok minőségének megfigyelése. Különböző technológiákat használhatunk fel erre, például ICMP (Internet Control Message Protocol), SNMP (Simple Network Management Protocol).

Megjelenítés

A hatékony felügyelet érdekében elvárt, hogy a felügyelt hálózat felépítése az abban lévő eszközök, kapcsolatok alaprajzszerűen dokumentálva és valamilyen jól áttekinthető formában megjelenítve legyenek. Ezáltal elősegíti a hibák felkutatását és javítását.

Naplózás

Feladata, hogy a hálózatban bekövetkező rendellenességeket nyilvántartsa könnyen elérhető és olvasható formában.

Riasztás

Rendeltetése, a rendszergazdák által előre konfigurált határértékek átlépésekor az adminisztrátorok felé történő értesítés.

2.2 A hálózat-felügyelet modellje [3] [4]

Ahhoz, hogy a hálózat-felügyeleti rendszerek egységesen és hatékonyan működhessenek, szükség van szabványokra, protokollokra. Ilyen szabványokat több szervezet is megalkotott, melyek a következők:

- International Organization for Standardization (ISO)
- International Telecommunication Union (ITU, régebben CCITT)
- Internet Engineering Task Force (IETF)

Ezen szervezetek közül az ISO szabványa a legelterjedtebb, amely az OSI (*Open Systems Interconnection*) részeként ad jól definiált architektúrát a hálózatmenedzsmentre.
A hálózatmenedzsment alapját az alábbi három modell alkotja:

1. Az *ISO 7894-4* szabvány, amely a hálózatmenedzsmentet funkciók szerint osztályozza
2. OSI referenciamodell, amely a hálózati funkciókat rétegekre osztja
3. Menedzser állomás/agent modell, amely biztosítja a hálózati egységek irányítását, ellenőrzését és monitorozását.

2.2.1 Funkciók szerinti osztályozás

Az International Organization for Standardization (ISO) hálózatmenedzsment modellje (FCAPS) öt funkcionális területet különböztet meg:

- Hibamenedzsment (Fault management)
- Konfigurációmenedzsment (Configuration management)
- Teljesítménymenedzsment (Performance management)
- Biztonságmenedzsment (Security management)
- Nyilvántartásmenedzsment (Accounting management)

2.1 ábra: FCAPS modell elemei

Hibamenedzsment

Célja, hogy detektálja, naplózza és a lehetőségekhez mérten automatikusan javítsa a hibákat, továbbá értesítse róluk a felhasználókat. Mivel a problémák a rendszer leállását okozhatják, vagy elfogadhatatlanul lelassíthatják, ezért a hibamenedzsment talán a legfontosabb és legszélesebb körben implementált eleme az ISO hálózatfelügyelet funkciói közül.

A legtöbb hálózati eszköz képes a felügyeleti állomások számára riasztást küldeni, amikor valami hiba keletkezik a hálózati rendszerben. Egy effektív hibamenedzsment rendszer több alrendszerből áll. A hibadetektálás akkor történik, amikor bármelyik eszköz SNMP trap üzenetet küld vagy átlép egy úgynevezett *remote monitoring* (RMON) határt, azaz egy előre konfigurált megfigyelési küszöböt.

Konfigurációmenedzsment

Feladata a hálózat és a rendszerbeállítások figyelése azért, hogy a hardver- és szoftverváltozások hatása a hálózatra követhető és menedzselhető legyen.

A telepített hálózati eszközök számának növekedésével egyre fontosabb az, hogy egy eszköz helyét pontosan meghatározhassuk. Ennek megkönnyítése érdekében elnevezési konvenciókat kell vagy érdemes bevezetni, egészen az eszköznevektől a különböző interfészekig. Ezek a konvenciók alapozhatnak földrajzi elhelyezkedésre, épületnévre, emeletre, irodára stb.

Bármely beállítás megadásakor meg kell bizonyosodnunk arról, hogy a hálózat integritását ne befolyásolja. Egy hibásan konfigurált hálózati eszköznek katasztrofális hatásai lehetnek a hálózat kapcsolataira és teljesítményére.

Teljesítménymenedzsment

Feladata a különböző teljesítménymutatók gyűjtése az interfészek, eszközök és protokollok szintjein, SNMP-t felhasználva. A legtöbb felügyeleti rendszer képes ezeknek tárolására és megjelenítésére is. A begyűjtött statisztikák alapján meghatározható a teljesítmény szintje. Eszközszinten ezek közé tartozik a CPU kihasználtság, a puffer- és memóriaallokáció. Továbbá ezek segítségével optimalizálhatjuk a felsőbb szintű protokollok teljesítményét. Interfészek esetén ezek a mutatók lehetnek az átviteli kapacitás, a válaszidő, keretméret stb. A teljesítmény határértékeit a hálózatot felhasználó szervezet szolgáltatási szint megállapodásában (*SLA – service level agreement*) rögzíthetjük.

Biztonságmenedzsment

Célja, hogy a hálózati erőforrásokhoz való hozzáférést a szervezeti előírások szerint irányítsa azért, hogy a hálózat véletlenül vagy szándékosan ne rongálódhasson meg. A biztonságmenedzsment alrendszer képes például a felhasználók bejelentkezéseit figyelni és megtagadni a belépést azoktól, akik nem megfelelő kóddal rendelkeznek. A jó biztonság implementációja megfelelően erős biztonsági előírásokkal és procedúrákkal kezdődik.

Fontos, hogy egy platformfüggő, alapvető konfigurációs szabványt biztosítsunk minden router és switch számára, amelyek követik a legbeváltabb biztonsági módszereket, például:

- Access Control Lists (ACL)
- Lokális, eszközszintű felhasználónevek és jelszavak
- Terminal Access Controller Access-Control System (TACACS)

A hálózatok biztonsági szintjét a TCSEC (*Trusted Computer System Evaluation Criteria*) által definiált szintek szerint osztályozhatjuk. A szintek A-tól D-ig terjednek, ahol „A” jelenti a legmagasabb biztonsági szintet, „D” pedig a legalacsonyabbat. „A”-ba a kritikus, „B”-be a kiemelt, „C”-be a normál, míg „D”-be pedig a minimális védelmet igénylő rendszerek tartoznak.

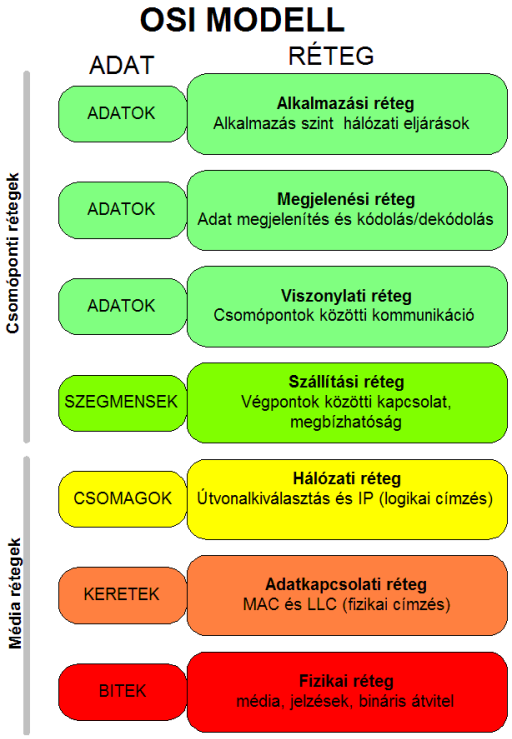
A biztonságmenedzsment része az authentikációs folyamat is, melynek feladata a felhasználók azonosítása felhasználónévvel és jelszóval, *challenge-response* mechanizmussal.

Nyilvántartásmenedzsment

Lényege, hogy a hálózat felhasználási paramétereit mérje, így elérve azt, hogy különböző felhasználók vagy felhasználók csoportjai megfelelően szabályozhatók legyenek. A teljesítménymenedzsmenthez hasonlóan a legelső feladat az összes fontos hálózati erőforrás használtságának mérése.

2.2.2 OSI referenciamodell [5]

Az Open Systems Interconnection referenciamodell egy rétegekbe szervezett rendszer absztrakt leírása, amely a számítógépek kommunikációjához szükséges hálózati protokollt határozza meg. Célja, hogy a különböző protokollok által nyújtott funkciókat egymásra épülő rétegekbe sorolja. Minden réteg csak és kizárólag az alsóbb rétegek által nyújtott funkciókra támaszkodhat és az általa megvalósított funkciókat pedig csak a felette lévő réteg számára nyújthatja. A rétegek hierarchikus rendszere meghatározza a két számítógép közötti kommunikáció feltételeit.



2.2 ábra: Az OSI modell

Fizikai réteg

A bitek kommunikációs csatornára való kibocsátásáért felelős. Biztosítania kell, hogy az adó által küldött jeleket a vevő is azonosként értelmezze. A gyakorlatban feladata az, hogy az 1-es jel mindenhol 1 legyen, míg a 0 mindenhol 0 maradjon. Lényeges, hogy a feszültség mennyi ideig tart egy bit továbbításához, az átvitel megvalósítható-e mindkét irányba, illetve az összeköttetés miként bomlik le, ha már nincsen szükség rá.

Adatkapcsolati réteg

Alapvető feladata a hibamentes átvitel biztosítása a szomszéd gépek között, vagyis a hibás, zavart, tetszőlegesen kezdetleges átviteli vonalat hibamentessé transzformálja az összeköttetés fennállása alatt. Az adatokat adatkeretekké (*data frame*) szabdalja, továbbítja, a nyugtázó keretet fogadja (*acknowledgement frame*), hibajavítást és forgalomszabályozást végez.

Hálózati réteg

A kommunikációs alhálózatok működését irányítja. Legfontosabb feladata az útvonalválasztás forrás és célállomás között, valamint az egymástól eltérő hálózatok összekapcsolásának lehetővé tétele. Az útvonalak meghatározása történhet statikus és dinamikus eljárással. A nyújtott szolgáltatásminőség (késleltetés, átviteli idő, sebességingadozás stb.) is ezen réteg feladatai közé tartozik. Az utolsó olyan réteg, amely ismeri a hálózat topológiáját.

Szállítási réteg

A végpontok közötti hibamentes adatátvitelt végzi. Már nem ismeri a topológiát, csak a két végpontban van rá szükség. Feladata az összeköttetések felépítése, bontása, csomagok sorrendbe állítása. Ennek gyakorlati megvalósítása során adatokat fogad a viszonyrétegtől és szükség esetén feldarabolja azokat kisebb egységekre, majd továbbítja az egyes darabokat a hálózati rétegnek és garantálja azt is, hogy a célállomásra minden kis egység hibátlanul megérkezzen.

Viszonyréteg

Lehetővé teszi, hogy két számítógép felhasználói kapcsolatot, viszonyt (*session*) hozzon létre egymással. Jellegzetes feladata a logikai kapcsolat felépítése és bontása, párbeszéd szervezése. Szolgáltatásai többek között a párbeszéd-irányítás (*dialog control*), a vezérjel kezelés (*token management*) és a szinkronizáció (*synchronizaion*), ellenőrzési pontok beépítésével.

Megjelenítési réteg

Szemben az alacsonyabb szintű rétegekkel, a megjelenítési réteg nem a bitek mozgatásával foglalkozik, hanem az átvitt információ szintaktikájával és szemantikájával. Az egyetlen olyan réteg, amely megváltoztathatja az üzenet tartalmát. Tömörít, rejtjelez (adatvédelem és adatbiztonság miatt), kódcserét (pl.: ASCII - EBCDIC) végez el.

Alkalmazási réteg

Széles körben igényelt szolgáltatásokat tartalmaz. Olyan protokollok változatos sokaságát fogja össze, amelyekre a felhasználóknak igen sokszor szükségük van, például az internet jelenlegi alapvető protokollja a HTTP, vagy a fájlok gépek közötti másolását megvalósító FTP.

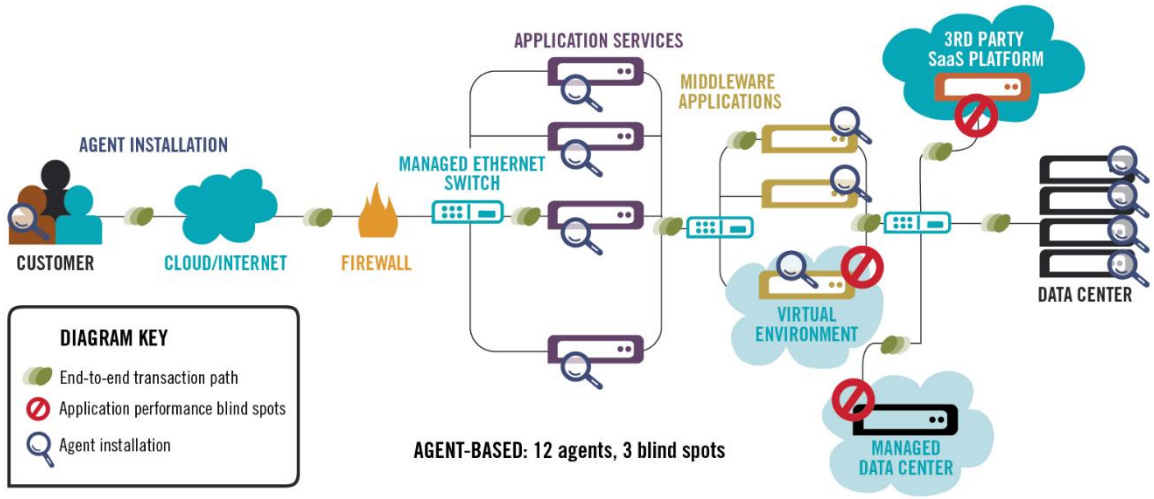
2.3 Ágensalapú és ágens nélküli hálózat-felügyelet [6] [7]

Míg régebben szinte csak ágenseket használó felügyeleti rendszereket alkalmaztak, manapság egyre jobban terjednek az ágens nélküli rendszerek. A kettő közötti különbség még mindig komoly vitákat vált ki az informatikus társadalomban. A választást nagyban befolyásolja a szervezet infrastruktúrája, céljai és erőforrásai. Az ágens nélküli felügyeletnek legnagyobb előnye a könnyű kitelepítés, az alacsony költség és a rugalmasság. Az ágensalapú felügyelet ezzel szemben több lehetőséget nyújt és általánosságban elmondható, hogy biztonságosabb és stabilabb.

2.3.1 Működésük

A hálózatfelügyeleti eszközök valós idejű információt nyújtanak a hálózat teljesítményéről, ezt különböző metrikák gyűjtésével teszik lehetővé, amelyeket kétféleképpen szerezhetnek.

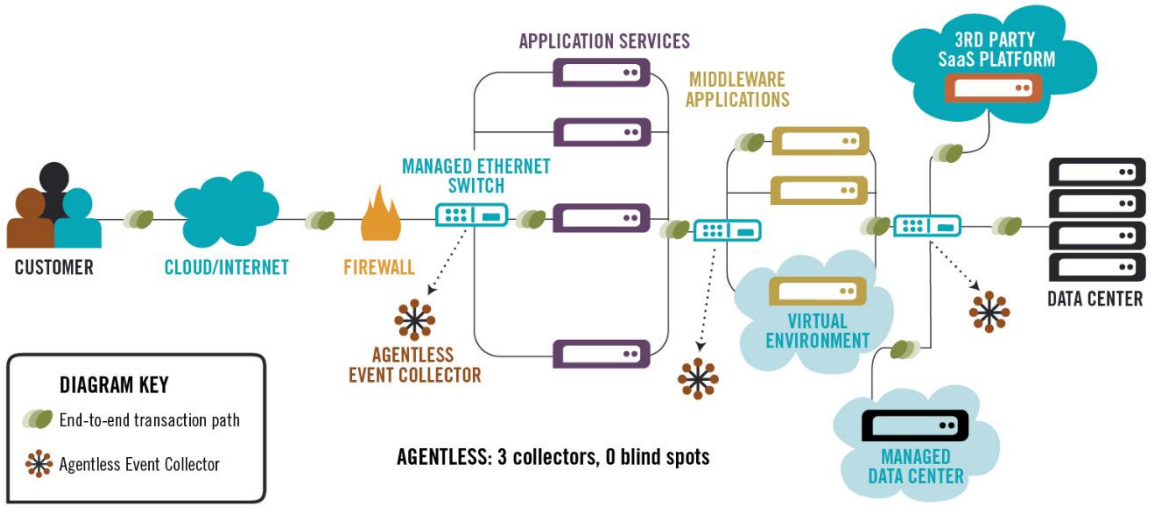
Az ágensalapú megközelítés egy előre feltelepített ágensre (szoftver) alapoz, hogy adatokat szolgáltasson a hardverekről és a rajtuk futó alkalmazásokról egy központi gép számára. Ennek követelménye, hogy minden megfigyelni kívánt eszközön telepítésre kerüljön az ágens program.



2.3 ábra: Ágensalapú felügyeleti modell

Az ágens nélküli megközelítés olyan protokollokat használ, melyek előretelepítettek a szervereken, a használt alkalmazások, továbbá hardverek alapból ismerik és kezelik őket. Ezáltal ezek a rendszerek nem igényelnek semmilyen extra komponenseket vagy alkalmazásokat. Ilyen protokollok például az SNMP, WMI (*Windows Management Instrumentation*), NetFlow stb.

A legtöbb eszköz és szerver rendelkezik azzal a képességgel, hogy egy adott protokollt támogasson. Az ágens nélküli rendszerekben elegendő a megfelelő információt továbbítani a megadott protokoll segítségével, ezzel ellentétben az ágensalapú rendszerek a saját fejlesztésű protokollok használatát igénylik, kikerülve ezzel az eszközök által alapból ismerteket.



2.4 ábra: Ágens nélküli felügyeleti modell

2.3.2 Ágens nélküli felügyelet előnyei

Általában a fő ellenérv az ágensalapú rendszerek ellen az, hogy menedzselésük komplex, implementációjuk időigényes. Ezzel szemben az ágens nélküli felügyelet előnyei:

- Telepítésük egyszerű és gyors

  A felügyelet azonnal elkezdődhet anélkül, hogy minden eszközre feltelepítenénk az ágenst. Ez kifejezetten előnyére válhat kisebb szervezeteknek, amelyek limitált erőforrással rendelkeznek.

- Beszerzése és fenntartása kevésbé költségigényes

  A könnyebb telepítéssel már időt lehet spórolni, továbbá kevesebb karbantartás igénye van, hiszen nem kell minden ágenst frissíteni, amikor az eszközeinket frissítjük.

- Nem szabványos technológiák megfigyelésére is alkalmas

  A legtöbb felügyeleti megoldás könnyen képes monitorozni általános hálózati komponenseket (szerver, router, switch). Az ágens nélküli megoldások képesek nem szabványos eszközök gyors felismerésére és megfigyelésére is.

3. SNMP – Simple Network Management Protocol

Az SNMP a *Simple Network Management Protocol*, vagyis az egyszerű hálózat menedzsment protokoll rövidítése. A TCP/IP család része, az IETF hozta létre. AZ SNMP protokoll egy egyszerű "kérdezz-felelek" protokollnak tekinthető, ahol az NMS-en (*Network Management System*) futó alkalmazások folyamatosan vagy egy előre meghatározott időközönként lekérdezik a felügyeleti eszközökhöz rendelhető változókat, amelyek valamilyen választ fognak adni további feldolgozás céljából. Lényeges, hogy egy elosztott felügyeletű protokollról van szó, amely a hálózatra kötött eszközök vezérlését, adatainak lekérdezését szolgálja.

3.1 Verziói

SNMP Verzió 1

Az első SNMP verzió, amelyet az RFC 1065, RFC 1066, RFC 1067 ír le. Több szállítási rétegbeli protokollal is használható, pl.: UDP, TCP, DDP, IPX. Fő hibája a nagyon gyenge biztonság. Az üzenetcsere nem titkosított, jelszó helyett pedig „*community*” változókat használ, ami olvasható szövegként kerül továbbításra, így könnyen támadható megoldás. Lehetőséget ad egyéni authentikáció használatára, de a legtöbb implementációban az üzenetek biztonsága függ az átviteli csatornától. Ezek ellenére nagyon széles körben elterjedt, és a hibáit áthidaló megoldásokkal jól használható.

SNMP Verzió 2

Leírja az RFC 1441 és RFC 1452. Javítottak a biztonságon, teljesítményen, menedzser-menedzser kommunikációt hoztak létre. A GETBULK üzenet létrehozásával több adatot egyszerre lehet lekérdezni, így javítva a v1 hatékonyságán. A bonyolult, és vitatott biztonsági megoldások miatt nem terjedt el, helyette az RFC 1901 és RFC 1908 által leírt SNMP v2c vagy nem hivatalos nevén az SNMP v1.5 vált ismertté. Ebben megtalálhatók az SNMPv2 javításai, de a bonyolult biztonsági megoldás helyett maradt a v1 egyszerű „*community”* változón alapuló megoldásánál. Bár a v1.5 elméletileg csak tervezet, nem pedig elfogadott szabvány, mégis ez a gyakorlatban elterjedt megoldás. Az SNMPv1 32-bites adatváltozóival ellentétben (ami például egy 10 gigabit-es vagy attól nagyobb interfész számára nem elegendő, hiszen kevesebb, mint 1 perc alatt betelik) az SNMPv2 már 64-bites változókat képes kezelni.

SNMPv1 és SNMPv2 kompatibilitás

A második verzió eltérő fejlécet és adategységet (PDU) alkalmaz az első verzióhoz képest, illetve két olyan funkciót is használ, amelyek az elődjében nem találhatóak meg. Ezen inkompatibilitás feloldása érdekében két lehetőséget fejlesztettek ki.

Az egyike ezeknek az úgynevezett „*proxy ágensek*” alkalmazása. Működési elvük szerint SNMPv2 ágensek, amelyek képesek SNMPv1 eszközök üzeneteit továbbítani megfelelő verzióval. A másik lehetőség a kétnyelvű hálózat-menedzsment rendszer alkalmazása, amelyek támogatják mindkét verziót. Az eszközökkel való kommunikáció előtt a rendszer egy adatbázisból lekérdezi, hogy a cél eszköz mely verziók üzeneteit tudja értelmezni.

SNMP Verzió 3

Az RFC 3411–RFC 3418 által leírt, jelenleg hivatalos SNMP szabvány. A régebbieket az IETF elavultnak nyilvánította. A gyakorlatban az eszközök több SNMP verziót is támogatnak, általában a v1, v2c és v3 szabványokat is. Az SNMPv3 főként két területre helyezi a hangsúlyt az újítások szempontjából, a biztonságra és az adminisztrációra. Ettől a verziótól kezdve erősebb authentikációt használ, illetve lehetőséget ad az adatok titkosítására is. Működésében szinte egyezik az SNMPv2-vel, annak funkcióit javítja, kiterjeszti.

3.2 Felépítése [10][11]

3.2.1 SNMP motor

Az SNMP motor nyújtja az üzenetek küldéséhez és fogadásához szükséges szolgáltatásokat, valamint authentikálást és titkosítást végez. Minden SNMP entitáshoz egy SNMP motor tartozik, amely a következő részeket tartalmazza:

- diszpécser
- üzenet-feldolgozó alrendszer
- biztonsági alrendszer
- hozzáférésvezérlés-alrendszer

snmpEngineID

Minden SNMP motor rendelkezik egy egyedi azonosítóval, amely az SNMP motorok és entitások közötti egy az egyhez megfeleltetés miatt nem csak a motort, de az entitást is egyértelműen azonosítja.

Diszpécser

A diszpécser (*dispatcher*) által képes az SNMP motor egyidejűleg az üzenetek több verziót támogatni. Üzenetek kiküldése vagy fogadása során meghatározza azok verzióját, majd a megfelelő üzenet-feldolgozó modellt választja ki. Absztrakt interfészt nyújt a protokoll adategységeinek szállítására.

Üzenet-feldolgozó alrendszer

Az üzenet-feldolgozó alrendszer (*message processing subsystem*) felelős a küldendő üzenetek összeállításáért, illetve a beérkezett üzenetek tartalmának feldolgozásáért. Több üzenet-feldolgozó modellt is tartalmazhat, amelyek a különböző SNMP verziók üzeneteinek formátumát definiálják.

Biztonsági alrendszer

A biztonsági alrendszer (*security subsystem*) olyan szolgáltatásokat biztosít, mint az authentikáció és titkosítás. Több biztonsági modellt tartalmazhat, amelyek leírják az alrendszerben felhasznált protokollokat és a veszélyeket, amik ellen a rendszernek védelmet kell nyújtania.

Hozzáférésvezérlés-alrendszer

A hozzáférésvezérlés-alrendszer (*access control subsystem*) authorizációs szolgáltatásokat nyújt egy vagy több hozzáférés-vezérlés modell szerint. Ezek a modellek a hozzáférési döntések függvényeit definiálják, amelyek alapján az authorizációt végzi a rendszer.

3.2.2 SNMP csomagok

Az SNMP üzenetek protokoll-adatcsomagokat (*protocol data unit – PDU*) tartalmaznak, amelyek a fogadó SNMP motor által végrehajtandó operációkat írják le. Ezen adatcsomagok mindegyike a következő osztályok valamelyikébe sorolható:

Olvasás – Olyan operációk tartoznak bele, amelyek valamilyen menedzsment információ kinyerésére szolgálnak. Adatcsomagjai: *GetRequest, GetNextRequest, GetBulkRequest*

Írás – Az írás osztály olyan műveleteket tartalmaz, amelyekkel a menedzsment információk módosíthatók. Adatcsomagja: *SetRequest*

Válasz – Egy megelőző olvasási műveletre adott válaszműveletek lehetséges fajtái tartoznak bele. Adatcsomagjai: *Response, Report*

Jelzés – Azon műveletek osztálya, amelyek jelzést küldenek egy fogadó alkalmazás számára. Adatcsomagjai: *Trapv2, InformRequest*

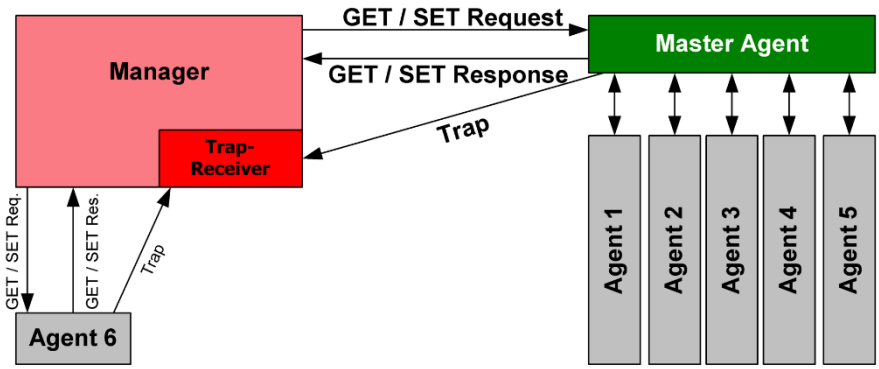
Belső – Olyan operációk, amelyek a rendszeren belül különböző SNMP motorok közötti kommunikációra használhatók. Adatcsomagja: *Report*

SNMP csomagformátuma:

<table>
<tr><td colspan="2">Üzenethossz Byte</td><td rowspan="3">SNMP-csomag Header</td></tr>
<tr><td colspan="2">Verziószám</td></tr>
<tr><td colspan="2">Community Name</td></tr>
<tr><td>Csomagtípus (Get, GetNext, …)</td><td rowspan="5">PDU-fejléc</td><td rowspan="11">PDU (Protocol Data Unit)</td></tr>
<tr><td>PDU hossz Byte</td></tr>
<tr><td>RequestID</td></tr>
<tr><td>Hibakód</td></tr>
<tr><td>Hiba-Index</td></tr>
<tr><td>PDU törzs hossza Byte</td><td rowspan="6">PDU-Body</td></tr>
<tr><td>Variable Binding 1</td></tr>
<tr><td>Variable Binding 2</td></tr>
<tr><td>…</td></tr>
<tr><td>Variable Binding n</td></tr>
</table>

3.3 Működése [8] [9]

Az SNMP a hálózati modellek alkalmazási rétegében működik. Minden üzenete UDP-n keresztül továbbítódik. Az ágensek a 161-es porton keresztül kapják a kéréseket, a menedzser állomás pedig a 162-es porton keresztül kapja az értesítéseket. Az SNMP jellemző használati eseteiben egy vagy több adminisztratív számítógép (menedzser) kapja azt a feladatot, hogy felügyeljen vagy menedzseljen eszközöket számítógépes hálózaton.



3.1 ábra: Az SNMP működési modellje

Minden menedzselt eszköz egy szoftvert futtat (ágens), ami SNMP üzeneteket felhasználva információt küld a menedzsernek. Az SNMP-felügyelt hálózatok három fő komponensből állnak:

- Menedzselt eszközök
- Ágens
- Network Management Station (NMS) – szoftver, ami a menedzseren fut

3.3.1 Management information base (MIB)

Az SNMP ágensek a menedzsment adatokat a felügyelt eszközökön változók formájában használják. A protokoll a konfiguráció változtatását ezeken a változókon keresztül engedi meg, amelyeket hierarchiába rendez. Az SNMP ténylegesen nem határozza meg, hogy egy felügyelt rendszer milyen változókat szolgáltasson, ehelyett bővíthetőre tervezték, tehát megengedi az alkalmazások számára, hogy saját hierarchiát határozzanak meg. Ezeket a hierarchiákat nevezzük MIB-eknek.

A MIB struktúráját és szabályait a felügyeleti adatok struktúrája (*SMI – Structure of Management Information*) adja meg. Nem határozza meg a menedzselhető eszközöket, de minden olyan adatot tartalmaz, amely az eszközök menedzselése miatt lényeges. Ilyen adat például az objektum név, adattípus, leírás, hozzáférés és azonosító. Minden MIB egyedi azonosítóval rendelkezik (*object identifier*), amely alapján fa struktúrába rendezhetők a létező implementációk, melyek lehetnek általánosak, vagy gyártó-specifikusak.

A következőkben bemutatom a hálózat-menedzsment rendszerek egyik legismertebb implementációját, a Nagios-t.

4. Nagios [12] [13]

4.1 Mi az a Nagios?

A Nagios a legelterjedtebb, nyílt forráskódú hálózat-felügyeleti szoftver, amely a hálózatokat, valamint az általuk kialakított infrastruktúrát és az ezekben megtalálható rendszereket felügyeli. Monitorozó és riasztási feladatokat ellátó szolgáltatásokat ajánl szerverekhez, hálózati eszközökhöz, valamint alkalmazásokhoz. Eredetileg Linux-on való futásra tervezték, de azóta elérhető bármely más Unix disztribúción, illetve Microsoft Windows alatt is. Fejlesztése is folyamatos, hiszen szélesebb körben használják, mint a hozzá hasonló megoldásokat (Zabbix, Groundwork, stb.). Ebből adódóan a legtöbb általános igényre már született megoldás, az esetek többségében nem szükséges saját kiegészítés elkészítése. Feladata tulajdonképpen a beállított szolgáltatások ellenőrzése, ennek során a megfigyelt értékek számszerűsítése és ezen értékek megadott szintekhez való viszonyításával a szolgáltatás aktuális állapotának felmérése. Ha az ellenőrzött érték eltér a megadott szinttől, a rendszer előre konfigurált feladatokat futtat, amely lehet értesítés küldése megfelelő helyre, vagy egy korrigáló program elindítása is.

A Nagios két fő részből áll:

- Nagios Core

  Ez a Nagios magja, amely a konkrét logikát tartalmazza, tehát az ellenőrzéseket, értesítéseket, logolást végzi az aktuális konfiguráció alapján.

- Nagios Web

  A Nagios Core-hoz tartozó scriptekből áll, amelyek a webes felületért felelősek. Ebben a rétegben generálódnak a weboldalak, ahol az aktuális állapotokat, illetve eseményeket tekinthetjük meg, valamint további ellenőrzéseket futtathatunk. A felület az alapértelmezett megjelenítőn kívül kiszolgálható más modullal is.

4.2 Működése

A rendszer központjában a Nagios processz áll, amely köré vannak felvéve a host-ok, hálózatra kapcsolt eszközök, amelyekhez szolgáltatások rendelhetők. Célszerű a valós

hálózati topológia alapján konfigurálni az alkalmazást, hiszen ha ezt követjük, úgy probléma esetén annak felkutatása és elhárítása is gyorsabban megtörténhet. A Nagios leköveti ezt a topológiát, tehát például ha egy router megy tönkre és a valós hálózatról leszakad a rákapcsolt többi eszköz is, akkor a felügyelet számára az ezeken futó szolgáltatások automatikusan elérhetetlenné válnak. Így az alkalmazás visszajelzései alapján látható, hogy feltehetőleg hálózati hiba történt, nem pedig a szolgáltatásokkal van gond. A hálózatban lévő eszközök ilyenfajta függőségének beállítását a „parent-host" konfigurációs érték megadásával tudjuk megtenni.

A szolgáltatások nem csak számítógépekhez rendelhetők, host lehet bármilyen eszköz, amely képes IP kommunikációra (nyomtató, IP telefon, kamera stb.). A hozzárendelés lehet magától értetődő (levelező szolgáltatás – levelező szerver), azonban akár logikai is, hiszen egy szolgáltatás nem feltétlenül egy konkrét hálózati kiszolgáló által nyújtott szolgáltatást takarhat. Bármi lehet a Nagios-ban szolgáltatás, ami számokkal kifejezhető (pl. rendszerek hőmérsékletének változása). Lehet szoftveres, vagy valamilyen fizikai, hardvert érintő változás. Ezek megfigyelését plugin-ek végzik, amelyek nagyrésze az általánosan felhasznált kiegészítőkkel együtt fel is települ a rendszer installálásakor. Igény szerint saját plugin is készíthető, speciális igények kiszolgálására, melyhez a követelmények a Nagios weboldalán elérhetők.

4.2.1 Állapotok

Az ellenőrzések eredményeként a végpontok és szolgáltatások különböző állapotokba kerülhetnek. Végpontok esetén ezek az *Up, Down, Unreachable* lehetőségek. Az „Up" állapot azt jelöli, hogy az eszköz fut és elérhető a hálózaton, míg a „Down" ennek ellenkezője. Az „Unreachable" állapotot akkor veheti fel, ha a hozzá megadott „parent-host" sem érhető már el, ezért nem tud megbizonyosodni a gép állapotáról a rendszer.

Szolgáltatások esetén a felvehető állapotok az *OK, Warning, Critical és Unkown*. Az „OK" magától értetődően a hibamentes állapot. A „Warning" azt jelzi, hogy a szolgáltatás jellemzői közül egy vagy több átlépte a jelzési határértékét, míg a „Critical" a kritikus határértékek túllépését jelenti. Az ismeretlen állapotot akkor veszik fel, ha az ellenőrző programok a Nagios számára értelmezhetetlen válaszüzenetet adnak. A „Warning" és „Critical" állapotokhoz beállíthatók a „Warning limit" és „Critical limit" értékek.

4.3 Beállítások

4.3.1 Szolgáltatás konfiguráció

A szolgáltatások beállításai közé tartozik a korlátokon túl az ellenőrzés periódusa (*check period*), valamint rendszeressége (*check interval*). Ezekkel azt adhatjuk meg, hogy milyen időszakban (pl. a nap melyik óráiban) legyen egy-egy szolgáltatás monitorozva, illetve mekkora időközönként. Ha a Nagios változást észlel az előző értékekhez képest, akkor az ellenőrzést a beállított ismétlési idő (*retry interval*) elteltével újra végrehajtja, annyiszor ismételve, amennyi a maximum ellenőrzések száma (*max check attempts*) beállításban limitálva van. Ha minden megismételt érték eltérő, akkor változtatja a szolgáltatás állapotát.

Ezeken túl értesítési beállítások is megadhatók. Az ellenőrzés időszakától függetlenül konfigurálható az értesítés időszaka (*notification period*). A notifikáció célja és formája úgynevezett kontaktcsoport (*contactgroup*) hozzárendeléssel adható meg, figyelve arra, hogy ki és milyen szintű állapotokról kapjon értesítést. Az említett ellenőrzési és figyelmeztetési időszakokat *Timeperiods* néven hívja a Nagios. Megadott időszakok közül lehet választani, heti bontásban adhatók meg, egy napra akár több is.

4.3.2 Kontakt és kontaktcsoport

A kontaktok (*contacts*) létrehozásakor meg lehet adni, hogy az adott felhasználót milyen változásokról, milyen módon és időszakokban tájékoztassa a rendszer. Lehetőség van a felhasználó nevének, e-mail címének és telefonszámának megadására is. Alapértelmezetten a rendszer az e-mail-ben történő tájékoztatást használja, de beállítható SMS küldés is. Az alkalmazás értesítésnek hívja, de probléma esetén egy program futtatása is opció, paraméterekkel kiegészítve. Képes állapotváltozásokat fájlba logolni, programot indítani argumentumként a gép IP címével, így automatizálni a problémamegoldást.

A már említett kontaktcsoportoknak kiemelt szerepe, hogy a szolgáltatások értesítési beállításait egyszerűsítse, redundánsabbá tegye. Ajánlott célként kontaktcsoportot (*contactgroup*) megadni, nem pedig egyedi kontaktokat. Könnyebbé teszik a felhasználók menedzselését is, hiszen személyi változás esetén a megfelelő személyt elég a csoporthoz rendelni. Egy kontakt több csoport tagja is lehet, illetve egy szolgáltatáshoz vagy géphez több csoport is hozzárendelhető.

4.3.3 Adatforrások

A Nagios az általa felhasznált adatokat többféle módon és helyről képes begyűjteni. Az ellenőrizni kívánt szolgáltatások és gépek alapján választhatjuk ki a legmegfelelőbb forrást.

Közvetlen port ellenőrzés

Erre az adatgyűjtési módra akkor van lehetőségünk, amikor a Nagios-t futtató számítógép ugyanabban az alhálózatban van, mint az ellenőrzött gépek, és azok IP porton elérhető szolgáltatásait szeretnénk monitorozni. Ekkor a rendszer adott plugin-jai közül a megfelelő kiválasztásával és a szükséges paraméterekkel való futtatásával el is indítható a felügyelet. Publikus szolgáltatások működésének vizsgálatára remekül működik, pl. HTTP, SMTP.

SNMP

Összetettebb, nem publikus szolgáltatás, hanem valamilyen állapot megfigyelésére alkalmas. Például switch portok kimenő forgalmának átlaga adott időn belül.

Távoli gépek ellenőrzése

Az előző adatforrások segítségével csak külsőleg monitorozható adatokat képes gyűjteni a rendszer, azonban egy szolgáltatás működésének leállása sokszor előre jelezhető, ha a kiszolgálójának egyéb teljesítménymutatóit is figyelni tudjuk (memória- és CPU kihasználtság, tárkapacitás használata, stb.). A Nagios erre több lehetőséget is ad.

Az egyik ilyen a megfigyelni kívánt állomásra Linux / Unix esetén az **NRPE** (*Nagios Remote Plugin Executor)*, Windows esetén pedig az **NSClient++** telepítése. Ezek működése megegyezik, mindkettő a Nagios által adott plugin-ok futtatására szolgál. A Nagios szerver és a monitorozni kívánt eszköz közötti biztonságos hálózati kommunikációt biztosítják. Használatukkal helyben futtathatók az említett teljesítménymutatókat lekérdező kiegészítők.

A másik, kevésbé használt lehetőség pedig az, hogy a megfigyelt eszköz önmaga futtatja rendszeres időközönként az említett plugin-okat, a visszakapott eredményeket pedig elküldi a szerver felé. Ehhez szükség van az **NSCA** (*Nagios Service Check Acceptor*) nevű programra mindkét oldalon (a szerveren és a host-on is). Ennek a módszernek hátránya a bonyolultabb telepítés mellett az, hogy például váratlan meghibásodás esetén több időbe telik, amíg a rendszer értesítést kap a problémáról. Hálózati meghibásodás esetén a kimaradó értesítés hiányában pedig nem tud extra információval szolgálni a felhasználó számára.

4.4 Ellenőrzés

A rendszer installációjakor az egyik legfontosabb döntés, hogy mit és hogyan ellenőrizzünk. Mivel nincs semmilyen szabályozás arra vonatkozóan, hogy a szolgáltatások milyen jellemzőit szükséges figyelni, ezért a telepítést általában hosszas finomhangolás követi. Érdemes például a kiszolgálók hardvereinek állapotát, a hálózati kapcsolat minőségét és meglétét ellenőrizni, továbbá minden paramétert, amely kihatással lehet a szolgáltatás minőségére és működésére. Biztonsági szempontból kifejezetten hasznos lehet a feltelepített patch-ek és újonnan elérhető rendszerfrissítések számának figyelése. Linux disztribúciók használatakor ezt a *check_apt*-vel, míg Windows esetén a *check_windows_updates*-el tehetjük meg.

A Nagios lehetőséget ad különféle logfájlok feldolgozására is, külső programot futtatva bármit nyomom lehet követni. Például DHCP szerver mellett futtatott daemonja képes összegyűjteni a hálózatban működő eszközök MAC címeit, esetleg IP címeit is, ezáltal új gép megjelenéséről tud értesíteni.

Válaszként célszerűen az alábbi három értéket kell eljuttatni a rendszernek:

- ellenőrzés eredménye,
- rövid, szöveges értesítés, amely leírja az állapotot,
- hosszabb, tájékoztatóbb szöveg, amely további információt ad (*performance data*)

4.4.1 Log feldolgozás példák

Linux rendszerek esetén:

auth.log – sikeres és sikertelen bejelentkezések száma.

mail.err – hány levelet küldtek IP címenként nem létező felhasználóknak

apache2/error.log – hány kérést utasított el hibával a webkiszolgáló

syslog – tűzfal által eldobott csomagok száma.

Windows rendszerek esetén:

A Windows eseménynaplójában minden bejegyzéshez tartozik egy úgynevezett *Event ID*, valamint a logok három csoportra vannak osztva: rendszer (*system*), biztonsági (*security*), valamint alkalmazás (*application*) típusokra.

Ezek figyelésére a már korábban említett NSClient++ egyik beépülő moduljának, a **CheckEventLog**-nak az alkalmazása mellett, lehetőség van a **Nagios EventLog Agent for Windows** programot is használni. Az utóbbi **NSCA**-n keresztül csatlakozik a Nagioshoz, tehát aktívan küldi az üzeneteket a szerver felé, nem pedig passzívan vár lekérdezésre. Ha a beállított szűrési feltételeknek megfelelő bejegyzést talál a logban, arról azonnal értesíti a szervert. Mindkét figyelési megoldásnál megadható, hogy visszamenőleg milyen időszakot figyeljen a logokban és bizonyos bejegyzések milyen gyakori előfordulása esetén jelezzen (*MinWarn, MinCrit, MaxWarn, MaxCrit*). Ilyen ellenőrzések lehetnek például:

Ki és mikor állította le a gépet az elmúlt 30 napban?

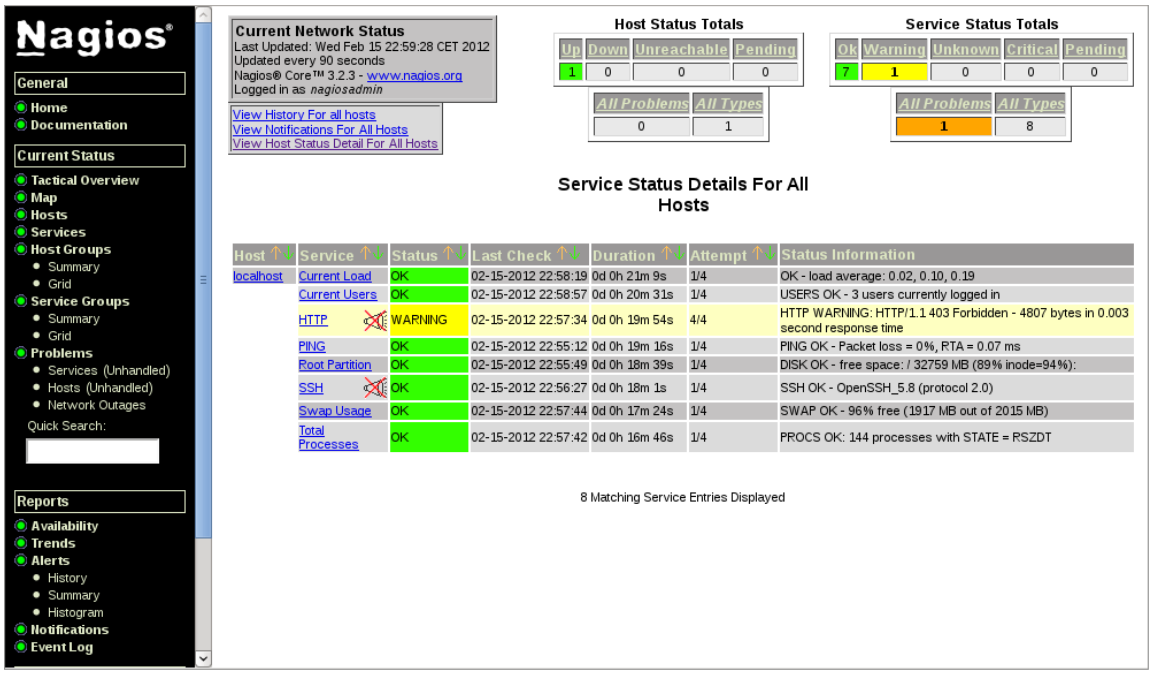
```
check_nrpe -H 127.0.0.1 -p 5666 -c CheckEventLog -a filter=new
file=system MaxWarn=5 MaxCrit=10 filter-generated=\
```

Az elmúlt egy nap sikertelen bejelentkezéseinek ideje:

```
file=security descriptions MaxWarn=5 MaxCrit=10 "filter=generated gt
-1d AND message LIKE 'failed to log on'" "syntax=%generated%,"
```

Az elmúlt két nap azon bejegyzései, amelyek hibára utalnak:

```
check_nrpe -H 192.168.9.85 -p 5666 -c CheckEventLog -a file=system
descriptions MaxWarn=5 MaxCrit=10 "filter=generated gt -2d AND
severity NOT IN ('success', 'informational')" "syntax=%message%"
```



4.1 ábra: A Nagios alapértelmezett felülete

5. Az alkalmazás tervezése

A már korábban említett ágens nélküli és ágensalapú felügyeleti rendszerek több előnnyel és hátránnyal is rendelkeznek. Az ágens nélküli felépítés esetén például nincs szükség a szoftver feltelepítésére minden megfigyelni kívánt eszközön, viszont ha a központi alkalmazás elveszíti a kapcsolatot egy alhálózattal, akkor az abban lévő eszközökről nincs információja. Ezzel szemben az ágensalapú megközelítésnél az ágensek képesek helyben tárolni a gyűjtött adatokat, amelyek később továbbíthatók a központ felé. Ez azonban azt igényli, hogy az ágens minden eszközön telepítve legyen. Ezen felül az ismertebb hálózat-felügyeleti rendszerekben sokszor saját konfigurációt igényel az SNMP használata, a megfigyelni kívánt adatok megadása komplex, amelyre kisebb cégeknél gyakran nincs erőforrás-kapacitás vagy esetleg megfelelő tudás. Kisebb hálózatok esetén nem feltétlenül érdemes a drága és bonyolult telepítéssel járó rendszerek használata.

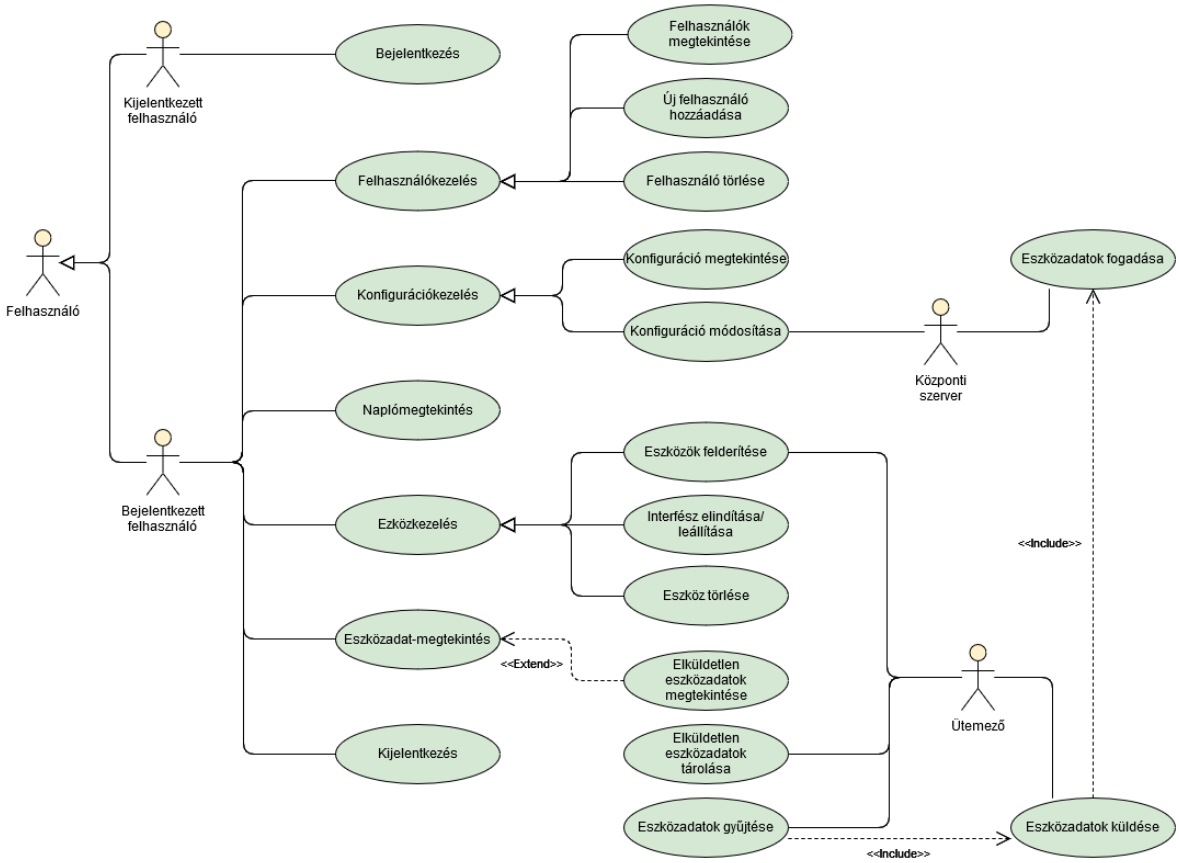
Feladatom egy olyan alkalmazás elkészítése, amely ezen előnyöket együttesen kihasználja, miközben a hátrányokat is mérsékeli. Képes minden eszközről információt gyűjteni, valamint ezeket a központi állomás felé küldeni, amely az adatok feldolgozását végzi. Ugyanakkor annak elérhetetlensége esetén is tovább tud működni, majd a kapcsolat helyreálltakor a mentett adatokat automatikusan továbbítani. Nem szükséges minden eszközre telepíteni, elegendő alhálózatonként egyre, amely lehet dedikált vagy olyan, már az alhálózatban lévő eszköz, amely alkalmas a futtatására. Elindítása nem igényel különösebb szaktudást, az alhálózaton található SNMP képes eszközöket automatikusan felfedezi. A gyűjtendő információk konfigurációját dinamikusan kapja a központi alkalmazástól.

Használatával lényegében kialakul egy olyan rendszer, amely a központi komponens szemszögéből ágensalapú, viszont az alkalmazásom részéről ágens nélkülinek tekinthető. Mindemellett webes felületet nyújtva lehetőséget ad az alkalmazás felhasználóinak az eszközadatok folyamatos megtekintésére, illetve a hálózati elosztókon található interfészek elindítására és leállítására.

5.1 Az alkalmazás funkciói

Egy szoftver tervezésének első lépése általában az igények felmérése, amely által leírható a rendszer elvárt funkcionalitása, a felhasználók által végezhető feladatok. Ezt a leírást sok esetben egy használati eset (*use-case*) diagram kíséri.

Az alkalmazásom „use-case” diagramja a következő:



5.1 ábra: Az alkalmazás use-case diagramja

A diagram háromféle elemtípust tartalmaz. Az aktorok olyan entitásokat jelölnek, amelyek valamilyen kapcsolatban állnak a rendszerrel, pl.: felhasználók, külső vagy belső szoftverelemek, távoli állomások stb. Az ellipszisekben látható használati esetek a rendszer funkcióit reprezentálják, a kapcsolatok pedig a relációkat mutatják be aktor-használati eset, vagy eset-eset között.

5.1.1 Aktorok

Felhasználó

A ki- és bejelentkezett felhasználó általánosítása. A kijelentkezett felhasználónak csak bejelentkezni van lehetősége, míg a bejelentkezett felhasználó tulajdonképpen adminisztrátor, mert az alkalmazásban nem különítek el szerepköröket. Joga van a rendszer összes funkcióját használni.

Ütemező

Olyan belső szoftverentitás, amely megadott időközönként elvégzi az eszközök felderítésének, valamint a felderített eszközökről történő adatgyűjtésnek a feladatát, továbbá vezérli ezen adatok továbbítását és tárolását.

Központi szerver

Az említett, több alhálózatba külön kitelepített ágenseket összefogó rendszert jelöli, amely felé az alkalmazás a begyűjtött adatokat továbbítja, illetve amitől a konfigurációs módosításokat várja.

5.1.2 Használati esetek

Bejelentkezés

Célja, hogy lehetőséget adjon védekezésre a rendszer jogosulatlan felhasználása ellen. A felhasználók bejelentkezési felületen azonosíthatják magukat felhasználónév és jelszó segítségével, értesítve őket az esetleges authentikációs hibákról.

Kijelentkezés

A bejelentkezés fordítottja, mellyel a feladatuk elvégzése után a felhasználók biztonságosan ki tudnak lépni az alkalmazásból, lezárva munkamenetüket. Egy bizonyos időkorlát leteltével a rendszer is automatikusan kilépteti őket.

Felhasználókezelés

Olyan használati eset, amely általánosítja a program felhasználóival végezhető műveleteket. Lehetőséget ad a felhasználók felületen történő megtekintésére, azok törlésére, valamint új felhasználó hozzáadására.

Konfigurációkezelés

Szintén általánosító eset, amely magában foglalja az alkalmazás aktuális konfigurációjának megtekintését, illetve módosításának lehetőségét. A módosítást végezheti felületen egy felhasználó, vagy a központi rendszer is üzenet küldésével.

Naplómegtekintés

Lehetőséget nyújt azon bejegyzések felületen történő megtekintésére, amelyeket a rendszer bizonyos események bekövetkezésekor elment.

Eszközkezelés

Olyan használati eset, amely általánosítja a hálózati eszközökkel kapcsolatos funkcionalitásokat. Lehetőséget ad az alhálózatra kapcsolt eszközök felderítésére, melyet indíthat felhasználó is, de az alkalmazás ütemezője az aktuális konfigurációban megadott időnként automatikusan elvégzi. Továbbá magában foglalja az eszközök törlésének lehetőségét, illetve a switch-ek interfészeinek elindítását vagy leállítását, melyeket a felhasználók tudnak végrehajtani.

Eszközadat-megtekintés

Az alkalmazás által ismert hálózati eszközöktől gyűjtött adatok statisztikai szintű megtekintését teszi lehetővé. Ezt a használati esetet egészíti ki a központi rendszer felé sikertelenül továbbított eszközadatok megtekintése is.

Eszközadatok gyűjtése

Az alkalmazás lényegi feladata, amit a kód ütemezője hajt végre az aktuálisan beállított időközönként. SNMP segítségével lekérdezi a megfelelő adatokat, majd tárolja ezeket.

Eszközadatok küldése

Az említett adatok gyűjtésének velejárója, hogy az alkalmazás minden beérkező adatot továbbít a központi szerver felé.

Eszközadatok fogadása

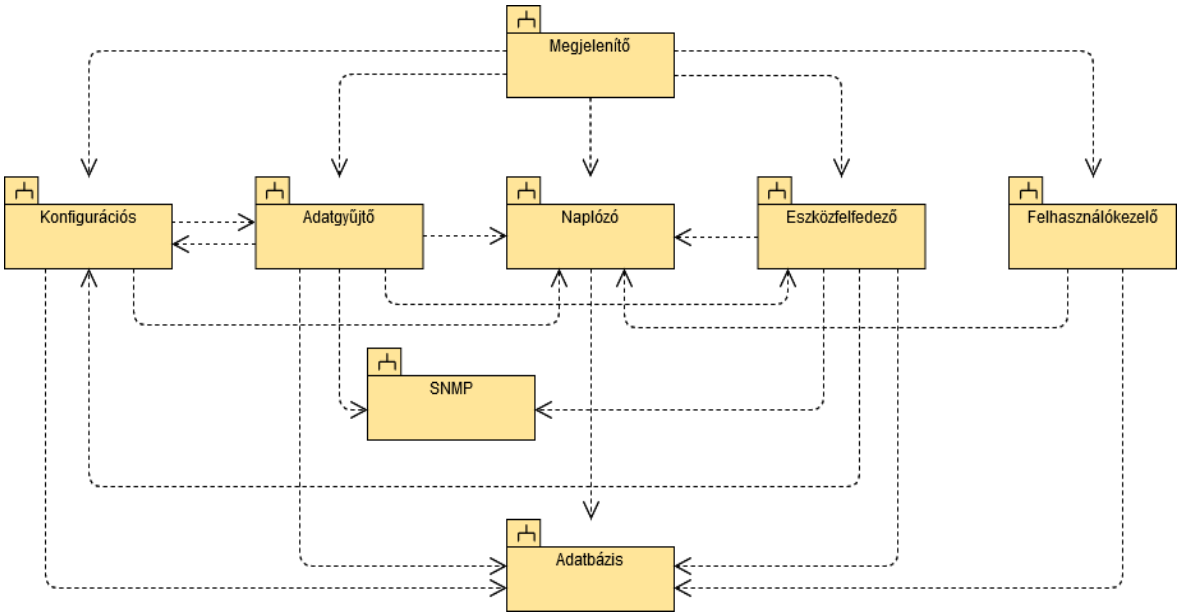
Az adatok küldésének velejárója, hiszen a központi szervernek fogadnia kell ezeket. Habár nem közvetlenül az alkalmazás funkciója, de működéséhez szorosan kötődik, ezért tüntetem fel.

Elküldetlen eszközadatok tárolása

Az ütemező hajtja végre abban az esetben, ha hálózati vagy egyéb hiba miatt nem tudja a begyűjtött adatokat továbbítani a központi szerver felé.

5.2 Az alkalmazás komponensei

A használati esetek kialakítása után megterveztem a megvalósításukhoz szükséges komponenseket, alrendszereket. Ezeket egy egyszerűsített komponens diagram segítségével szemléltetem az alábbi képen:



5.2 ábra: Az alkalmazás komponens diagramja

Adatbázis

Az adatbázis komponens, ahogy a neve is mutatja, gyakorlatilag maga az adatbáziskezelő rendszer, amely az alkalmazásomban egy beágyazott adatbázis kell, hogy legyen.

SNMP

Az alkalmazás összes SNMP kommunikációjáért felelős komponens. Ennek használatával képes az eszközfelfedező és adatgyűjtő alrendszer feltérképezni a hálózatot, valamint begyűjteni az elvárt adatokat.

Eszközfelfedező

Az alkalmazást futtató számítógéppel közös alhálózaton lévő eszközök felderítéséért és azok későbbi menedzseléséért felelős komponens.

Adatgyűjtő

Feladata az alkalmazás által ismert eszközök konfigurációban meghatározott adatainak gyűjtése, valamint ezen adatok továbbítása a központi szerver felé.

Konfigurációs

Az alkalmazás konfigurációjának kezelését végzi, beleértve annak külső rendszertől való fogadását, tárolását, módosítását, valamint az aktuális beállítások többi komponens felé történő kiszolgálását.

Felhasználókezelő

Célja a program felhasználóival kapcsolatos műveletek implementálása. Ezen műveletek közé tartozik az új felhasználó hozzáadása, meglévő törlése, valamint a biztonságos authentikáció menetének biztosítása.

Naplózó

Feladata az alkalmazás többi komponensétől érkező logüzenetek tárolása, valamint a megjelenítő felé való kiszolgálása.

Megjelenítő

A felhasználói webfelület megjelenítéséért, illetve tartalmak lekérdezéséért felelős komponens. A közvetlenül alatta elhelyezkedő réteg minden komponensét felhasználja a megfelelő adatok elérése érdekében.

6. Alkalmazáshoz felhasznált technológiák

6.1 Programozási nyelv

Az alkalmazás elkészítéséhez mindenképpen egy olyan programozási nyelv választására van szükség, amely rendelkezik az SNMP protokollt megvalósító könyvtárral. A lehetőségek között szerepel a *Node.js*, amely tulajdonképpen szerveroldalon futtat JavaScript kódot, ezzel könnyebbé téve programok elkészítését, hiszen a JavaScript általában a programozás belépő nyelvei közé sorolható egyszerűbb struktúrája miatt. Az SNMP-t megvalósító könyvtára a *net-snmp* csomag, amely minden lényeges funkcionalitásra megoldást ad.

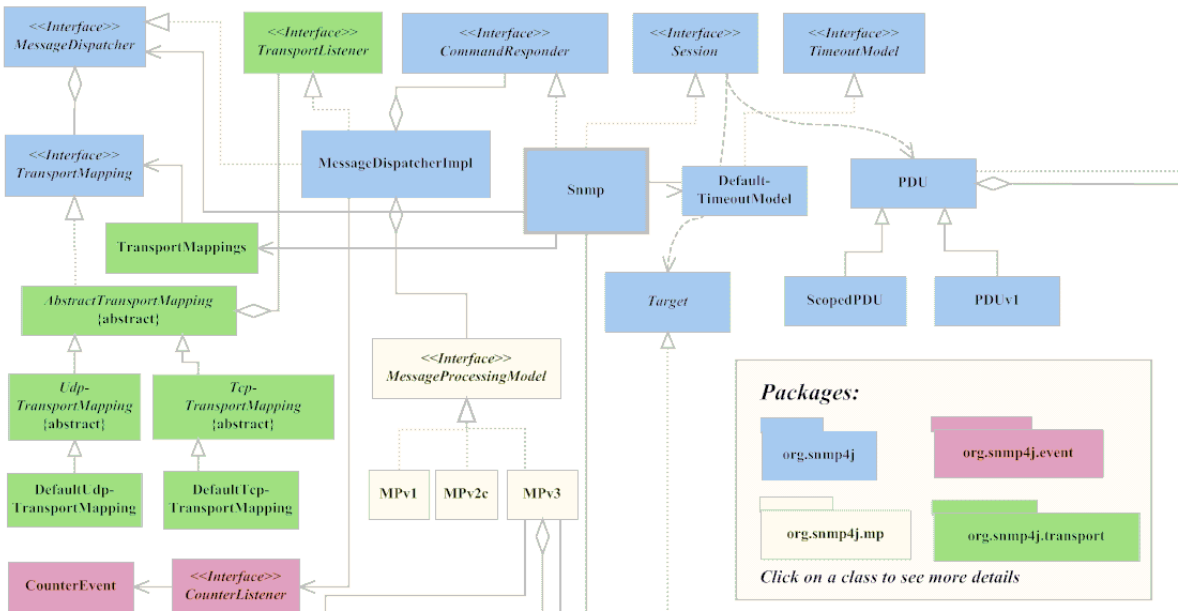
Másik lehetőségként adott a *C* nyelv, mely gyorsaságával, széleskörű felhasználhatóságával vívta ki azt, hogy minden idők legszélesebb körben használt programozási nyelve legyen. Egy gyengén típusos, imperatív nyelvről van szó, amely rendszerprogramozáshoz és felhasználói programok készítéséhez is egyaránt használható. Az SNMP protokoll használatára a *Net-SNMP* könyvtára ad megoldást, amely szintén támogatja a protokoll által nyújtott különböző szolgáltatásokat.

Választásom azonban a *Java* nyelvre esett, hiszen objektumorientált paradigmájával, biztonsági jellemzőivel, folyamatos fejlesztésével, egyre gyorsuló teljesítményével és átlátható struktúrájával kivívta, hogy napjaink egyik legmeghatározóbb programozási nyelve legyen. Nagyvállalati rendszerekben túlnyomó részt ezzel találkozhatunk, hiszen platformfüggetlensége miatt bármilyen architektúrán futtatható fordítás nélkül. Egyetemi tanulmányaim során is ennek a nyelvnek volt a legnagyobb szerepe.

Jelenlegi legfrissebb verziója a Java 14, azonban köszönhetően annak, hogy a fejlesztése gyors ütemben halad, valamint a Java 11-től kezdődően fizetéshez kötött a használata, a legelterjedtebb verziója a Java 8 (Java 1.8). Erősen típusos, interpreteres nyelv, amely az alkalmazáskódot először bájtkóddá fordítja, majd az *interpreter* ezt értelmezi. Használata könnyebb, mint a szintaxisában hasonló nyelveké, C és C++. Legnagyobb könnyítései, hogy nem tartalmaz programozó által kezelhető mutatókat, illetve a memóriakezelést sem bízza ránk. A nyelv készítőinek célja elsősorban az objektumorientáltság, platformfüggetlenség és a biztonság volt. Az utóbbiakat úgy éri el, hogy a kódot nem közvetlenül az operációs rendszer futtatja, hanem az úgynevezett *Java Virtual Machine* (JVM). SNMP funkcionalitást biztosító könyvtára az SNMP4J.

6.1.1 SNMP4J

Az SNMP4J egy ingyenes, jól dokumentált és korszerű SNMP implementáció Java 8 és újabb verziók számára. Támogatja mind a parancsküldést (menedzser), valamint azokra való válaszküldést (ágens). Objektumainak megalkotásában szerepet játszott az SNMP++, amely egy jól ismert API C++-hoz. Tervezésében figyelmet fordítottak a szálkezelésre is, alapvetőleg többszálú környezetekhez készítették. Osztályokat és interfészeket biztosít SNMP üzenetek létrehozására, küldésére és fogadására, valamint események feldolgozására. *Security* csomagja authentikációs lehetőségeket nyújt.



6.1 ábra: Az SNMP4J osztálydiagramjának részlete

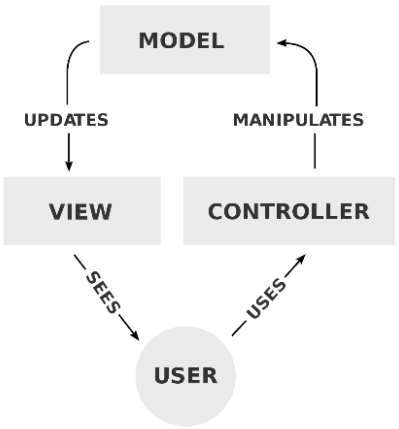
6.1.2 Spring keretrendszer

Mivel a feladatban elkészítendő ágens célja, hogy távolról is könnyen elérhető legyen a felhasználók számára, ezért webalkamazásként implementálom. A webfejlesztés megkönnyítése érdekében használom a *Spring Framework*-öt, amely a szerverfunkciók fejlesztésében segít. Ez a Java nyelvhez jelenleg létező legismertebb és legtöbbek által használt keretrendszer. Főbb felhasználási területe a webes kiszolgálást nyújtó alkalmazások programozása a *Java EE (Java Enterprise Edition)* platformon alapulva, azonban funkciói bármilyen Java program fejlesztésében használhatók.

A Spring egyik fő tulajdonsága a függőségek kezelése. *Bean*-nek nevez minden olyan objektumot, amely valamilyen függőségben (*dependencia*) áll más objektumokkal. Egy

„bean” tulajdonképpen egy osztály egy létrehozott példánya. Ezeket a példányokat tárolja a „bean”-ek tárolójában, amely külön korlátozás nélkül minden osztály számára elérhető. Az objektumok függőségeit ebből a tárolóból szolgálja ki, aminek előnye, hogy elkerülhetjük a felesleges többszöri példányosítást, hiszen a meglévő példányok újra felhasználhatók. A különböző osztályok konstruktoraiban sincs szükség explicit példányosításra, hiszen a Spring a megfelelően felannotált osztályok esetében képes felmérni a függőségeket és a létező bean-ek közül átadni a már létező példányokat (*dependency injection*), vagy akár automatikusan példányosítani. Hátránya, hogy a kezelt objektumok felett elveszítjük a kontrollt, mint programozók. Mindezek mellett természetesen további előnye, hogy a hosszas, bonyolult logikát igénylő *Servlet*-ek helyett a Spring által nyújtott osztályokat vehetjük igénybe. Két kiegészítő modulját is felhasználom, a *Spring MVC*-t és *Spring Security*-t.

A Spring MVC a modell-nézet-kontroller (*model-view-controller*) tervezési mintát valósítja meg, amely elkülöníti az adatok kezelését (modell) a megjelenítéstől (nézet). Ezeket a kontroller réteg köti össze, amely a felhasználók által hívható végpontokat, valamint szolgáltatásokat definiálja.



6.2 ábra: Az MVC tervezési minta

A Spring Security pedig az authentikációs és authorizációs funkciókat biztosítja. Saját biztonsági kontextussal rendelkezik, amely a bejelentkezett felhasználók adatait tárolja. Csak belépett felhasználók számára engedélyezett műveletek elvégzésekor ezek alapján végez vizsgálatot. Szerepkörök is rendelhetők a felhasználókhoz, amelyek korlátozhatják bizonyos metódusok elérését (authorizáció).

6.1.3 Thymeleaf

A tervezésnél említett megjelenítési rétegért az alkalmazásomban a *Thymeleaf* nevű modern, nyílt forráskódú, szerveroldali *template motor* lesz felelős. Azért ezt választottam, mert fejlesztésénél törekedtek a Spring keretrendszerrel való minél könnyebb integrációra. Feladata, hogy a megírt sablonokat a böngészők számára értelmezhető HTML kódra fordítsa. Java programoknál használható, célja a *Java Server Pages (JSP)* technológia teljes leváltása, miközben a logikát egyszerűsíti.

A Thymeleaf sablonjai gyakorlatilag szokásos HTML struktúrák, mind kinézetben és működésben, annyi további lehetőséggel, hogy igény szerint kiegészíthetők Java kóddal. Például lehetséges elemek megjelenését feltételhez kötni, ciklusokat futtatni, változókat kiértékelni, stb., amiket az elemeknél megjelenő attribútumokkal kezel, melyek mindegyikének névtere a „th”, pl.: „th:text”, „th:if”, „th:each”. Biztosítja úgynevezett *fragment*-ek (oldaltöredékek) létrehozását is, amelyek beilleszthetők bármely más sablonba.

6.1.4 Fejlesztői környezet

Manapság két nagyobb fejlesztői környezet létezik Java-hoz, az *Eclipse* és *IntelliJ*. Mindkettő hasonlóan felhasználóbarát és ugyanazok a fejlesztést elősegítő eszközök elérhetők bennük különböző formában, azonban mivel eddigi fejlesztéseim során Eclipse-t használtam, így a mostani feladat elkészítéséhez is ezt preferálom.

Mint minden *IDE (Integrated Development Environment)*, az Eclipse lényege is az, hogy a kód írásához, ellenőrzéséhez és futtatásához egy közös felületet biztosítson, ezzel gyorsítva a fejlesztés ütemét. Első verziója 2001-ben jelent meg, az IBM fejlesztése által. Mára az egész projektet az „Eclipse Foundation” viszi. Fő felhasználási területe a Java alkalmazások készítése, hiszen maga is Java nyelven készült, de rengeteg ingyenesen elérhető kiegészítőjének segítségével használható például C, C++, C#, Ruby, Python nyelvekhez is.

Működését tekintve elmondható, hogy az Eclipse legtöbb funkcióját, egy kis kernelt leszámítva, plugin-ok adják. Magában foglalja az *Eclipse Java Development Tools*-t, amely rendelkezik egy beépített Java értelmezővel, ezzel lehetőséget adva fejlett refaktorálásra és kódanalízisre. Képes interfészeket biztosítani olyan funkciók egyszerű használatára, mint az alkalmazásszerverek menedzselése, verziókezelés, illetve projektmenedzsment.

6.2 Adatbázis

Mivel az alkalmazásom felhasználókat, valamint eszközöket és tőlük beérkező adatokat kezel, ezért szükséges egy adatbázis, amelyben a tárolásuk perzisztens módon megoldható. Annak érdekében, hogy a program könnyedén elindítható legyen bármely rendszeren, nem rendelkezhet a webalkalmazásoknál megszokott, külön telepítendő adatbázissal. Így választásom a ***H2***-re esett, amely egy Java nyelven írt relációs adatbázis-menedzsment rendszer. Számomra fő előnye, hogy a megszokott szerver-kliens mód mellett képes Java alkalmazásokban beágyazott módon futni. A sztenderd SQL egy részhalmazát valósítja meg. Lehetőséget ad memóriában való tárolásra, aminek előnye a sebesség, azonban hátránya, hogy minden újraindításkor elveszti tartalmát. Ebből kifolyólag a másik módot, a fájlban tárolást vettem igénybe, hiszen ez már perzisztens funkcionalitást is nyújt. Minden adatmanipulációs műveletet tranzakciók keretében futtat, valamint tábla szintű zárolásra is képes a konkurencia kezelése érdekében. Támogatja a 2PC protokollt is. A beágyazó alkalmazás futásakor webfelületű adminisztrációs konzolt is biztosít, ahol a szokásos kezelői műveletek nagyrésze elvégezhető. A felületre felhasználónév – jelszó párossal történik a bejelentkezés, amelyet alapértelmezetten SHA-256-os hasheléssel tárol.

6.3 Verziókezelő

Szoftverfejlesztésben általában szükség van valamilyen verziókezelő rendszerre, hiszen több fejlesztő dolgozik ugyanazon a kódon. Olyan esetben is érdemes használni azonban, amikor egy programozó több számítógépről változtatja a kódot, ezáltal elkerülhető az eszközök közötti manuális másolgatás. Az alkalmazás elkészítését én is több számítógépről végeztem, ezért előnyt jelentett a ***Git*** nevű, elosztott verziókezelő rendszer használata. Ennek célja, hogy egy adott tárolóban (*repository*) lévő fájlok változásait kövesse, valamint az ezek közötti gyors váltást, visszaállást biztosítsa. A nem lineáris munkafolyamatokat is remekül támogatja, mert a tárolóban lévő fájlokat elágaztathatjuk, így egyszerre több verzió tud egy időben élni. A különálló fejlesztések végeztével lehetőséget ad ezen elágazások egyesítésére is. Meglévő rendszerekkel és protokollokkal kompatibilis, például a tárolók kezelhetők HTTP, FTP, SSH vagy egyszerű socket-eken keresztül is. A verziók historikusságának biztosítása érdekében úgy tárolja ezek ID-ját, hogy azok függenek az őket megelőző összes korábbi verziótól, tehát visszamenőleg semmi sem változtatható észrevétlenül. Kezelhető parancssoros felületről, vagy manapság számtalan grafikus felület használatával.

6.4 Projektmenedzsment eszköz

A szoftverfejlesztés másik, szinte már elengedhetetlen eleme egy projektmenedzsment eszköz használata. A két legelterjedtebb megoldás az *Apache Maven,* illetve a *Gradle*. Mivel egyetemi tanulmányaim, illetve eddigi munkáim során **Maven**-t használtam, így a mostani feladat elkészítéséhez is ezt választottam. A Maven tulajdonképpen egy automatizációs eszköz, amelyet főképp Java projektekhez használnak. Két fő feladata van: hogyan lesz futtatható program a kódból, illetve a kód függőségeinek kezelése. Egy Maven projekt középpontjában a *POM (Project Object Model)* áll, amely egy XML leírófájl. Ebben kerül definiálásra a teljes projekt, illetve a függőségei. Előre definiált célokat is tartalmaz, amelyek gyakorlatilag futtatható feladatok, például „compile" a kód fordítására, „test" a tesztek futtatására, „package" futtatható JAR fájl készítésére, stb. Lehetőségünk van különböző plugin-ok felhasználásával egyéb célok használatára is, vagy akár saját célokat is írhatunk. A POM-ban felsorolt függőségeket pedig úgynevezett *repository*-kból szerzi be, tehát nem magunknak kell foglalkozni a felhasznált külső könyvtárak letöltésével és projektbe való illesztésével.

6.5 Hálózat szimuláció

A fejlesztés közben történő könnyebb tesztelés érdekében felhasználtam a *GNS3*-at (*Graphical Network Simulator 3*), amely egy 2008-ban megjelent hálózati szoftver emulátor. Lehetőséget ad virtuális és fizikai eszközök együttes használatára komplex hálózatok szimulálásának kivitelezésére. Alapja a *GNS3 VM*, amely az alkalmazás saját virtuális gépe. Minden importált eszközt ezen futtat további virtualizációt használva. Erőssége abban rejlik, hogy az alapértelmezett eszközökön kívül a https://www.gns3.com/marketplace/appliances oldalról letölthető a leggyakrabban használt eszközök képfájlja. Importálásuk után az alkalmazás képes ezeket közel teljes funkcionalitással szimulálni. Akár hivatalos Cisco eszközök hozzáadására is van lehetőségünk, amelyeket a *Dynamips* emulációs szoftver segítségével indít el.

7. Fejlesztés előkészületei

7.1 Fejlesztőkörnyezet kialakítása

Minden alkalmazás elkészítésének első lépése a megfelelő fejlesztőkörnyezet kialakítása, melynek Java programok esetén legfontosabb eleme a *Java Development Kit (JDK)* installálása. Ezt legkönnyebben az Oracle hivatalos oldaláról tölthetjük le, a megfelelő operációs rendszer és Java verzió kiválasztása után, az alábbi linken:

https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

A JDK telepítése elengedhetetlen ahhoz, hogy Java programokat fejlesszünk, hiszen ez tartalmaz minden szükséges könyvtárat.

Ennek installálása után az Eclipse IDE-t szükséges telepíteni, melyet a következő linken érünk el: https://www.eclipse.org/downloads/packages/ – Itt ki kell választanunk a megfelelő programnyelvhez és operációs rendszerhez illő változatot. Mivel az alkalmazásom webfelületet is biztosít, ezért az *Eclipse IDE for Enterprise Java Developers* verziót telepítem. Első indításkor ne felejtsük el átállítani az Eclipse által használt Java könyvtárt, mert alapértelmezetten a sima JRE-t használja. Ezt a *Window* > *Preferences* > *Java* > *Installed JREs* > *Add: Standard VM* lehetőségnél tehetjük meg az előzőekben installált JDK könyvtárának hozzáadásával, majd ennek kiválasztásával.

7.2 Projekt létrehozása

A környezet kialakítása után érdemes a projektstruktúra megalkotása a megfelelő Spring projekt létrehozásával. A Spring projektek egyszerűbb generálására hozták létre a https://start.spring.io/ oldalt, másnéven a *Spring Initializr*-t. Itt beállíthatjuk a projektünk típusát, programnyelvét, metaadatait, illetve a projekthez tartozó dependenciákat. Én az alábbi beállításokkal hoztam létre:

- Project: Maven Project
- Language: Java
- Spring Boot: 2.2.5
- Project Metadata > Packaging: War; Java: 8

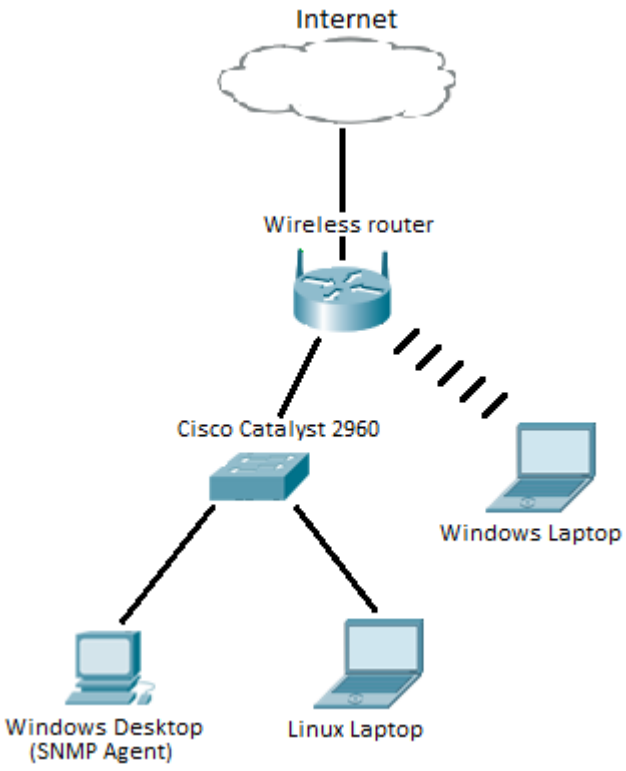
Dependenciák:

- Spring Web – Spring MVC komponenst biztosítja
- Spring Security – Authentikációs modult tartalmazza
- Spring Boot DevTools – fejlesztést könnyítő funkciókat tartalmaz
- Spring Data JPA – Adatbázis könnyebb kezelését segíti
- H2 Database – a H2 adatbázis használatát biztosítja
- Thymeleaf – A Thymeleaf template motor használatát biztosítja

7.3 Teszthálózat kialakítása

Annak érdekében, hogy az alkalmazás a fejlesztés során folyamatosan tesztelhető legyen, szükségem volt egy teszthálózat kialakítására olyan formában, hogy az minél egyszerűbben elindítható és használható legyen. Ehhez használtam fel a már említett GNS3 nevű hálózat szimulátor szoftvert, valamint a fejlesztés végén az alkalmazás éles tesztelése miatt a szimulált hálózatot fizikailag is megalkottam az egyetem által biztosított *Cisco Catalyst 2960 Plus* hálózati switch segítségével.

A kialakított hálózat a következőképpen épült fel:



7.1 ábra: A kialakított teszthálózat

A GNS3-ban az „Internet” egy olyan virtuális eszköz, amely képes a virtuális gépet futtató hoszt hálózati adapterének állapotát lemásolni, illetve annak kapcsolatát áthidalni a szimulált hálózatnak. Ennek az áthidalásnak a beállítását a GNS3-hoz kötelezően feltelepített *VMware Workstation*-ben a „GNS3 VM” képfájl konfigurációi között találhatjuk. Emellett lehetőséget ad más üzemmódokra is: „NAT” (amely a gazdagép IP címét osztja meg), „Host-only” (ami egy privát hálózatra csatlakoztatja a virtuális eszközöket), illetve „Custom” (egyedi beállításokat biztosít).

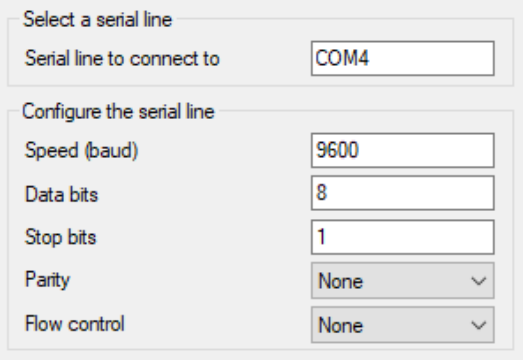
A szimulált és a valós hálózatban is a „Wireless router” nem SNMP képes eszköz, így annak megfigyelésére nem volt lehetőségem. A forgalomirányítást azonban ez végzi, IP címe a 172.16.1.1/24, amelynek a switch beállításánál lesz szerepe. A felhasználói eszközök DHCP-vel kapják IP címüket a router-től, egyiknek sem statikus. A hálózat felépítésének az alkalmazás működését befolyásoló szerepe nincs, gyakorlatilag bármennyi hálózati és hoszt eszközt csatlakoztathatnánk bármilyen topológiában, feltéve, hogy mindegyik az adott helyi hálózatban van.

7.3.1 Switch konfigurálása

Annak érdekében, hogy a switch monitorozható legyen SNMP felhasználásával, bizonyos beállításokat el kell végezni rajta. Ezt a szimulált hálózatban egyszerűen megtehetjük, hiszen a GNS3 minden virtuális eszközhöz szolgáltat konzol elérést. Fizikai eszköz esetén azonban szükségünk van néhány, a csatlakozáshoz megfelelő kábelre, illetve szoftverre. Rendelkeznünk kell egy konzol kábellel, amelynek egyik vége RJ45-ös csatlakozóval van ellátva, hogy a switch konzolportjával kompatibilis legyen, másik végén pedig egy RS232-es soros csatlakozó található. Mivel a mai számítógépek túlnyomó többségének I/O egységei között nem találunk soros portot, így ezt először egy RS232-USB átalakítóval szabványos USB csatlakozásra kell átalakítanunk. Ezen kábeleket is az egyetem biztosította számomra.

A szoftveres csatlakozást egy terminál emulációs program segítségével hozhatjuk létre. Ilyen szoftver például Linux rendszer esetén a *screen*, Windows-on pedig a *PuTTY*. Mivel Windows operációs rendszert futtató eszközről végeztem a beállításokat, ezért az utóbbit használtam.

A PuTTY felületén meg kell adnunk a megfelelő csatlakozási beállításokat. A „Serial” fül alatt a felhasznált portnak a nevét, sebességét, az adatbitek számát, a stopbitek számát, valamint a paritást és átvitelvezérlést kell konfigurálnunk az ábrán látható módon:



7.2 ábra: A PuTTY beállításai

A soros port nevét az eszközkezelőből olvashatjuk ki az USB csatlakoztatása után. A kapcsolat létrejöttét követően az eszköz parancssorát láthatjuk.

A Cisco IOS (*Internetworking Operating System*) operációs rendszer vezérli a Cisco által gyártott összes eszközt. Parancssoros elérést biztosít a konfigurációra, amelyeket különböző jogosultsági szintekhez köt. „User EXEC" módban alapvető, a hálózat működését nem szabályozó beállításokat érhetünk el, míg az „*enable*" paranccsal elért „Privileged EXEC" módban már lehetőségünk van ezek elérésére is. Az általam elvégzett beállítások és a hozzájuk tartozó parancsok a következők voltak:

Eszköznév beállítása

Az egyszerűbb azonosíthatóság miatt megváltoztattam az eszköz nevét.

```
S2>enable
S2#configure terminal
S2(config)#hostname Catalyst2960Plus
```

VLAN felügyeleti IP címének és alapértelmezett átjárójának beállítása

Ezek megadása azért szükséges, hogy az eszköz IP kommunikációval elérhető legyen.
A felügyeleti címet a használt alhálózat legnagyobb elérhető IP címére állítottam.

```
Catalyst2960Plus(config)#int vlan 1
Catalyst2960Plus(config-if)#ip address 172.16.1.254 255.255.255.0
```

Az alapértelmezett átjáró címét pedig az említett router IP címére állítottam.

```
Catalyst2960Plus(config)# ip default-gateway 172.16.1.1
```

SNMP beállítása

A felhasznált eszközön alapértelmezett módon engedélyezett az SNMP, így csak a megfelelő „community" változók beállítására van szükség.

```
Catalyst2960Plus#configure terminal
Catalyst2960Plus(config)#snmp-server community public RO
Catalyst2960Plus(config)#snmp-server community private RW
```

Az egyszerűség kedvéért a csak olvasási joggal (*RO*) rendelkező a „public" nevet kapta, míg az írási és olvasási (*RW*) joghoz a „private" került beállításra.

Beállítások mentése

Az elvégzett beállítások alapból a futási konfigurációba íródnak, amely nem permanens, ezért explicit el kell mentenünk, hogy az eszköz újraindítása után is érvényesek legyenek.

```
Catalyst2960Plus#copy running-config startup-config
Destination filename [startup-config]?
Building configuration...
[OK]
```

7.3.2 Hosztok konfigurálása

A hosztok az SNMP szolgáltató beállításán kívül más konfigurációt nem igényelnek. Ennek menete Windows és Linux operációs rendszerek esetében eltérő.

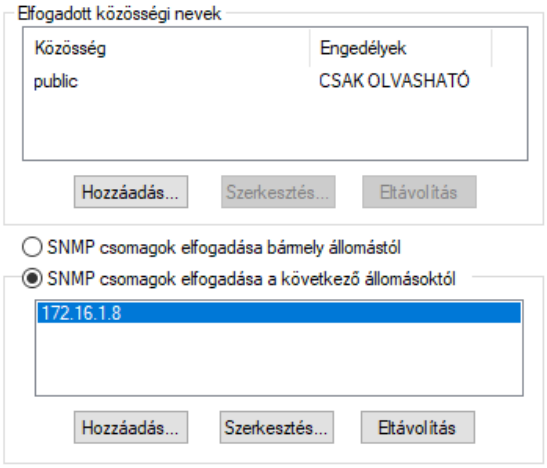
SNMP beállítása Windows-on

Első lépésként a „*Get-Service -Name snmp*"* parancs *PowerShell*-ben futtatásával megtudhatjuk, hogy telepítve van-e rendszerünkön az SNMP szolgáltatás. Amennyiben nincs, a Vezérlőpult *Programok > Programok és szolgáltatások > Windows szolgáltatások be- és kikapcsolása* menüpontjában kikeresve és kijelölve az SNMP-t, feltelepíthetjük. Ha a Windows 10 1803-as vagy annál újabb verzióját használjuk, akkor ebben a listában nem szerepel az SNMP, ezért az alábbi PowerShell parancsot szükséges használnunk:

```
Add-WindowsCapability -Online -Name "SNMP.Client~~~~0.0.1.0"
```

A telepítést követően a „*services.msc"* paranccsal meg kell nyitnunk a „Szolgáltatások" ablakot, amely a rendszer által futtatható szolgáltatásokat sorolja fel. Itt kikeresve az SNMP szolgáltatást, elérhetjük annak konfigurációját, illetve elindíthatjuk vagy leállíthatjuk azt.

A szolgáltatás konfigurációs ablakának „Biztonság” fülén az alábbi képen látható beállításokat végeztem el:



7.3 ábra: Az SNMP szolgáltatás beállításai Windows-on

Itt írási jogot nem állítottam be, mert a Windows a legtöbb funkció esetében nem ad lehetőséget SNMP általi távoli konfigurációra. A 172.16.1.8 pedig az alkalmazásomat futtató állomás címe, ezért csak innen kell a hosztoknak SNMP üzeneteket elfogadnia.

SNMP beállítása Linux disztribúciókon

Az általam használt Linux alapú hoszt az Ubuntu 18.04-es verzióját futtatja, ezért az ezen kiadandó parancsokat írom le, de más disztribúción is hasonlóak a beállítások.

Első lépésként telepítenünk kell az SNMP szolgáltatást az alábbi parancs segítségével:

```
sudo apt-get install snmpd
```

Ennek lefuttatása után a következő paranccsal szerkeszthetjük a szolgáltatás konfigurációs állományát, amelyben a „community” változókat és az elfogadott címeket adhatjuk meg:

```
sudo nano /etc/snmp/snmpd.conf
```

A fájl tartalma nálam a következő:

```
#Listen for connections on all interfaces (both IPv4 *and* IPv6)
agentAddress udp:172.16.1.8:161,udp6:[::1]:161
rocommunity public
rwcommunity private
```

A beállítások megegyeznek a Windows hoszt SNMP konfigurációjával.

8. Az elkészült alkalmazás

A programom alapötlete, hogy egy átlátható, könnyen használható hálózat-felügyeleti megoldást biztosítson olyan környezetben, ahol nem feltétlenül szükséges, vagy anyagilag nem megengedhető a bonyolult, drága szoftverek alkalmazása. Nem csak a technikai implementáció, hanem a felhasználói felület kialakítása során is törekedtem arra, hogy minél értelmezhetőbb és letisztultabb formában prezentáljam a funkciókat, valamint a begyűjtött adatokat. Ügyeltem arra, hogy az alkalmazás teljes kódja a bővíthetőséget szem előtt tartva kerüljön megírásra, illetve törekedtem a Web-MVC minta alkalmazására, valamint a Java 8 új nyelvi elemeinek használatára (Stream API, lambda kifejezések, stb.). A felület teljes egészét is angolul készítettem el, mivel véleményem szerint a célfelhasználóknak hálózati feladataikból következően jártasnak kell lenniük az informatikai szaknyelvben.

8.1 Elkészült komponensek

8.1.1 Adatbázis

Az általam beállított működése szerint a program első indításakor létrehoz egy „data” nevű könyvtárat ott, ahol az alkalmazás futtatható állományát elindítjuk. Ebbe a mappába kerül az adatbázis fájlja „snmp_agent.mv” névvel. A perzisztens működést az „*application.properties*” fájlban adtam meg az alábbi sorral:

```
spring.datasource.url=jdbc:h2:file:./data/snmp_agent
```

A szükséges adatbázistáblákat minden indításkor a „*schema.sql*” fájlt felhasználva automatikusan létrehozza a program, amennyiben még nem léteznek, továbbá a konfigurációs táblába beilleszti az alapértelmezett beállításokat. A táblákat létrehozó SQL utasításokra egy példa a következő:

```
create table if not exists Device_Data(data_id int auto_increment primary key, device_name VARCHAR(50), device_ip VARCHAR(16), data TEXT, timestamp TIMESTAMP, INDEX(device_name));
```

A „*Device_Data*” tábla tárolja az eszközöktől gyűjtött adatokat. Mezői az elsődleges kulcsként használt adat azonosító, az eszköz neve és IP címe, amelytől az adat származik, valamint maga az adat és az időbélyeg, amikor gyűjtésre került.

További táblák a *User*, *Device*, *Agent_Configuration*, *Unsent_Log* és *History_Log*, amelyek SQL utasításai a mellékletben szereplő forráskódban (*schema.sql*) megtalálhatók.

8.1.2 SNMP

Két osztályból áll, melyek az *SnmpOID* és *SnmpUtil*. Az SnmpOID csak konstans adattagokat tartalmaz, amelyek összekötik a konfigurációs kulcsokat a megfelelő MIB objektumazonosítókkal (OID), ezért ezt nem fejtem ki részletesebben. Az SnmpUtil osztály feladata az adatgyűjtés megvalósítása, fő metódusa a következő:

```
collectData(String ip, JSONObject configuration, String deviceType)
```

Az osztály „getter” metódusait felhasználva végzi az adatgyűjtést, amelynek eredményét egy szöveges kulcs-érték párokat tartalmazó „Map”-ben adja vissza. Minden futáskor ellenőrzi, hogy az aktuális konfiguráció alapján melyik adatok gyűjtésére van szükség.

A „getter” metódusok: *getSystemDescriptor, getDeviceName, getCiscoFanState, getCPUUtilization, getMemoryUtilization, getRunningProcesses, getInterfaces, getSystemUptime*, melyek paramétere az IP címe, illetve típusa annak az eszköznek, amelyről a lekérdezés történik. Elnevezésüket úgy alakítottam, hogy magától értetődő legyen a feladatuk. Ezek a metódusok figyelembe veszik az eszközök sajátosságait, különböző módon kérdezik le az adatokat ahol szükséges, vagy üres választ adnak vissza, ha az eszköz nem támogatja az adott adat lekérdezését. Például a processzor kihasználtságot Cisco eszközök esetében egy konkrét OID adja meg, hosztoknál viszont egy MIB tábla sorainak és oszlopainak iterálását kell elvégezni, majd a processzormagok kihasználtságát átlagolni, hasonlóan a memória, illetve interfészek terheltségénél is. A szimpla, egyértékű OID-ok lekérdezését a *sendGETPDU*, a táblákét pedig a *doWalk* metódussal végzik.

```
sendGETPDU(String ip, OID oid)
```

Az SNMP4J-t felhasználva az egyértékű OID-ok lekérdezését valósítja meg. Paramétere az eszköz IP címe és maga a kiolvasandó OID. Az értéket szöveges formátumban adja vissza, vagy null-t, ha az eszköz nem adott választ. Ebben kerül beállításra többek között a „community” változó és a használt SNMP verzió is az alábbiak szerint:

```
CommunityTarget target = new CommunityTarget();
target.setCommunity(new OctetString("public"));
target.setVersion(SnmpConstants.version2c);
```

Mivel olvasó művelet, ezért a „community” változót az eszközökön már konfigurált „public”-ra állítottam, a verziót pedig „2c”-re, mert a Windows nem támogatja a 3-as verziót.

```
doWalk(OID tableOid, String ip)
```

A *doWalk* metódus nagyon hasonló, annyi különbséggel, hogy paraméterként egy tábla OID-ját várja, az értékeket pedig egy kulcs-érték párokat tartalmazó „Map”-ben adja vissza.

A switchek interfészeinek elindítását és leállítását pedig a *shutdownInterfaceByName* és *startupInterfaceByName* metódusok végzik. Paraméterük az eszköz IP címe és típusa mellett az elindítandó vagy leállítandó interfész neve. Az interfésznév alapján a *doWalk* metódussal kikeresik az interfésztábla megfelelő OID-ját, amelyet a *sendSETPDU* metódus használatával 1-re (elindítás) vagy 2-re (leállítás) állítanak.

```
sendSETPDU(String ip, OID oid, Variable value)
```

Hasonló a lekérdező metódushoz, azonban paraméterei kiegészülnek egy értékkel, amelyre az adott OID-ot állítani szeretnénk. A „community” változó ebben „private”-ra van állítva, hiszen írási műveletet végez, valamint az általa küldött PDU típusa „*SET*”.

8.1.3 Eszközfelfedező

Az MVC tervezési mintából adódóan rendelkezik egy *Device* modellosztállyal, egy *DeviceRepository* interfésszel, valamint egy *DeviceDiscoveryService* és *DeviceDiscoveryController* osztállyal.

A *Device* osztály a Spring keretrendszer által biztosított annotációkkal ellátott leképzése az eszközöket tároló adatbázistáblának. Minden példánya egy rekordot reprezentál.

A *DeviceRepository* szintén a keretrendszer által nyújtott *CrudRepository* interfészből leszármaztatott interfész, ellátva a *@Repository* annotációval. Ezen leszármaztatás miatt az alapvető műveletekre (create, read, update, delete) alkalmas további kód írása nélkül. A *@Query* annotációt felhasználva kiegészítettem két metódussal, amelyek IP cím, illetve eszköznév alapján adják vissza a megfelelő adatbázisrekordot.

A *DeviceDiscoveryService* osztály a „service” réteget valósítja meg, amely gyakorlatilag a komponens működési logikáját tartalmazza. Ezeket az osztályokat a *@Service* annotációval kell megjelölnünk. Metódusai a forráskódban megtekinthetők, közülük csak a legfontosabbakat emelem ki, ezek az alábbiak:

```
@PostConstruct
private void discoverDevices()
```

A Java *CompletableFuture* osztályát felhasználva külön szálon meghívja a *deviceDiscoveryTask()* metódust, amennyiben nincs ismert eszköz még a rendszerben, valamint egy változóban tárolja a felderítés futási állapotát. A visszakapott eszközöket a *DeviceRepository* segítségével tárolja. A *@PostConstruct* annotáció miatt az alkalmazás indításakor automatikusan lefut.

```
deviceDiscoveryTask() throws NetworkPrefixTooSmallException
```

Meghatározza az alkalmazást futtató eszköz IP címét, melyből kiszámolja az alhálózat címét és maszkját. A kiszámolt lehetséges címeken, több szálon iterálva összegyűjti és visszaadja az elérhető eszközök nevét, leírását és IP címét az SNMP komponens segítségével. Kivételt dob, ha az alhálózati maszk kisebb, mint 16.

```
startRediscoveryTask()
```

A *TaskScheduler* Java osztály funkcionalitását felhasználva ütemezi az eszközök ismételt felderítését a konfigurációban megadott időközönként. Összehasonlítja a megtalált eszközöket a rendszerben lévőkkel, amiket ennek hatására frissít vagy eltárolja az újakat.

Ezeken felül biztosít olyan funkciókat is, amelyek az SNMP komponens felhasználásával képesek az eszközök interfészeivel való műveletekre is.

A *DeviceController* osztály pedig olyan URL-ekhez rendelt, megjelenítő komponens által használt metódusokat tartalmaz, amelyek a *DeviceDiscoveryService* segítségével eszköztörlésre, illetve azok interfészeinek elindítására és leállítására szolgálnak.

8.1.4 Adatgyűjtő

Szerkezetét hasonló elven alakítottam ki, emiatt a következőkből áll: *DeviceData* modell, *DeviceDataRepository* interfész, *DeviceDataCollectionService* és *DeviceDataCollectionController* osztályok. A modell és a hozzá tartozó „repository” minden komponens esetében hasonló, így a továbbiakban nem részletezem őket. A „service” és „controller” osztályok is ugyanolyan annotációkkal vannak ellátva minden esetben, szerepük is megegyezik, ezért csak a feladatukra és fontosabb metódusaikra térek ki.

A *DeviceDataCollectionService* osztály fő feladata az adatgyűjtés megvalósítása, amelyet a *startCollectingDeviceData()* metódusa végez. Az alkalmazás elindításakor

automatikusan ütemezi egy taszk konfigurált időnként történő ismételt lefutását, amely az eszközök listáján iterálva egyesével ellenőrzi, hogy elérhető-e az adott eszköz. Amennyiben igen, az SNMP komponens segítségével begyűjti a szükséges adatokat, majd tárolja azokat. Ha nem elérhető az eszköz, akkor kitörli a rendszerből. Az adatok tárolása után a *sendLog(LogData log, JSONObject conf_json)* metódus segítségével elküldi azokat a központi szervernek. A küldés sikertelensége esetén külön struktúrában mentésre kerülnek. Sikeres küldés után megkísérli az összes el nem küldött adat továbbítását is. Ezen felül lehetőséget biztosít más komponensek számára ezen ütemezett taszk újraindítására.

Másik fontos metódusa a *getCurrentInAndOutBandwidthByDeviceIdAndInterfaceName (int deviceId, String interfaceName),* amely az interfészek kihasználtsági adatait alakítja át a felhasználók számára értelmezhető számokká. A mért eredményeket átlagolva adja meg az átmenő forgalmat kilobájt/másodpercben.

A *DeviceDataCollectionController* pedig a felfedezési taszk futási állapotának lekérdezésére, valamint annak újrafuttatására ad URL-hez rendelt metódusokat.

8.1.5 Konfigurációs

Felépítése szintén azonos az eddigiekkel: *AgentConfiguration* modell, *AgentConfigurationRepository* interfész, *AgentConfigurationService* és *AgentConfigurationController* osztályok. Az első kettőt itt sem fejtem ki.

Az *AgentConfigurationService* osztály valósítja meg a komponens említett feladatait. Egyik fő metódusa a *listenForConfigurations()*, amely a már korábban is használt *CompletableFuture* segítségével külön szálon, az alkalmazás indításakor automatikusan meghívja az osztály *listen()* metódusát. Ez a Java *ServerSocket* osztályát felhasználva nyit egy portot az aktuális beállítások alapján, amelyen bejövő konfigurációkat fogad. Minden beérkezett üzenet token kulcsát validálja, majd a benne lévő konfiguráció szintaxisát és értékeit is ellenőrzi. Ennek függvényében vagy elmenti azt, vagy a problémát reprezentáló hibaüzenettel figyelmeztet. Sikeres mentés esetén a *restartListeningForConfigurations()* metódus segítségével újranyitja a socketet, hiszen megváltozhat a figyelendő port.
Másik fő metódusa a *saveNewConfiguration()*, amely a paramétereként kapott konfiguráció megfelelő validálása után elmenti azt. Ezt más komponens is használhatja.

Az *AgentConfigurationController* egyetlen, más kontrollerekhez hasonlóan URL-hez rendelt metódust tartalmaz, amely lehetőséget ad a felületről új konfiguráció mentésére.

8.1.6 Felhasználókezelő

Felépítése a következő: *User* és *UserPrincipal* modellek, *UserRepository* interfész, *UserService* és *UserController* osztályok. Az előző komponensekhez képest eltérés, hogy a felhasználókezelő az adatbázisrekordot leképző modellosztály mellett rendelkezik egy, a Spring Security által elvárt, *User* objektumokat becsomagoló *UserPrincipal* osztállyal, amely kiegészíti azt további adatokkal, például az adott felhasználó jogköreivel.

A *UserService* osztály valósítja meg ebben az esetben is a komponens tényleges működését. Implementálja a *UserDetailsService* interfészt, ezzel jelezve a Spring Security-nek, hogy ennek segítségével tudja végrehajtani az authentikációhoz szükséges lekérdező műveleteket. Az alapvető CRUD metódusok mellett megtalálható benne a *createInitialUser()*, amely az alkalmazás indításakor létrehoz egy kezdeti felhasználót, ha üres a felhasználók adatbázistáblája. A mentésre szolgáló metódus a jelszót SHA256-tal hasheli a Spring Security *BCryptPasswordEncoder* osztályának segítségével.

A *UserController* két URL-hez rendelt metódust tartalmaz, amelyek új felhasználó hozzáadására, valamint létező felhasználók azonosító alapján való törlésére használhatók.

8.1.7 Naplózó

Szerkezete közel egyező az előzőkkel: *HistoryLog* modell, *HistoryLogRepository* interfész, *HistoryLogService* osztály. Nem rendelkezik tehát kontroller osztállyal, hiszen nincs felületről elérhető funkciója. Az előző alrendszerek kifejtésénél nem tértem ki a logolásukra, de mindegyik komponens az alkalmazás működését, illetve az abban bekövetkező változásokat a megfelelő helyen és időben egy logüzenet formájában is jelzi a *HistoryLogService* osztály *saveLog(String message)* metódusának felhasználásával.

8.1.8 Megjelenítő

Egyetlen osztályból, a *ViewController*-ből, valamint számtalan HTML fájlból áll. A *ViewController* tartalmazza az URL-ek helyes HTML fájlokhoz való hozzárendelését a Spring MVC *@GetMapping* annotációját felhasználva. Az adatokat a komponensek „service” osztályaitól kéri, majd azokat egy *Model* objektum segítségével továbbítja. Az elkészített HTML fájlokat pedig a már említett Thymeleaf dolgozza fel, amelyekben elérhetők az átadott modellattribútumok. A Thymeleaf emellett lehetőséget ad úgynevezett

„*fragment*”-ek létrehozására, amelyek olyan oldalrészletek, amiket könnyen beszúrhatunk más fájlokba, ezzel gyorsítva az ismétlődő részek kialakítását. Az alkalmazásomban ilyen töredékek a „footer”, „head” és „navigation”, amelyeket minden oldalba beágyaztam.

Egyéb, komponensekhez szorosan nem köthető, említésre méltó fájlok:

DeviceDiscoveryFilter – neve alapján tartozhatna az eszközfelfedező komponensbe, azonban célja a felhasználó átirányítása egy információs oldalra a bejövő kérések figyelésével akkor, ha az eszközök felderítése még fut, vagy a hálózati maszk túl rövid.

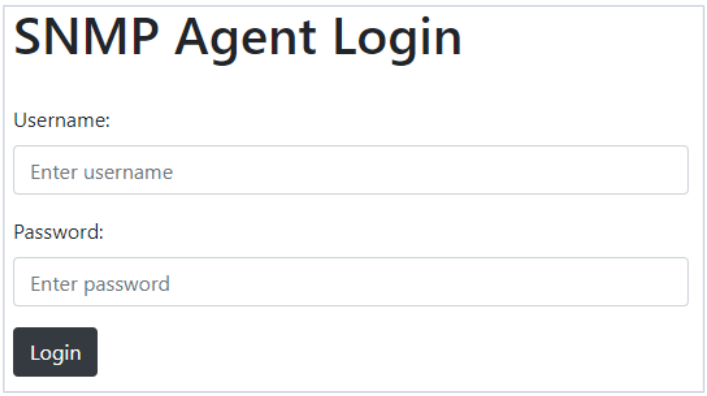
WebSecurityConfig – a Spring Security konfigurációs osztálya. Ebben adtam meg az általa felhasználandó „service”-t, az authentikációs módot, a titkosítás vagy hashelés metódusát, illetve a be- és kijelentkezés elérését, valamint a védett oldalak URL-jét.

8.2 Az alkalmazás felületei

Az alkalmazás elindítása után annak elérési címét a böngészőbe írva két különböző felületre juthatunk. Az egyik mindössze egy, a képernyő közepén megjelenő tájékoztató szöveggel jelzi számunkra, ha az első eszközfelderítés még folyamatban van, vagy olyan alhálózaton indítottuk el a programot, amelynek hálózati maszkja rövidebb, mint 16. Az utóbbi esetében meg kell változtatnunk a maszkot, míg az előbbi esetében a folyamat befejezése után automatikusan a bejelentkezési oldalra kerülünk.

8.2.1 Bejelentkezés

Mivel a funkciók csak authentikált felhasználók számára érhetők el, ezért az alkalmazás bármely felületére navigálva, átirányításra kerülünk a bejelentkező felületre.

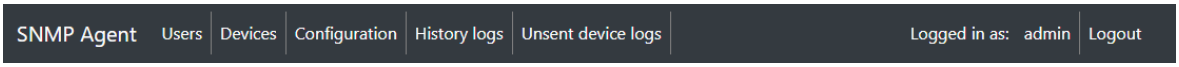


8.1 ábra: Az alkalmazás bejelentkezőfelülete

A bejelentkezéshez felhasználónév és jelszó megadására van szükség, amelyek közül egyik sem maradhat üresen, mert a böngésző a gombra kattintáskor ellenőrzi a mezőket. Hiányos vagy hibás adatok esetén hibaüzenettel figyelmeztet. Kijelentkezés után is ugyanerre az oldalra kerülünk, azonban ilyenkor egy sikeres kijelentkezést visszajelző üzenetet is kapunk.

8.2.2 Navigációs sáv

A navigációs sáv minden oldal tetején megjelenik, a bejelentkezést kivéve. A további felületek bemutatásának könnyítése érdekében ezt külön fejtem ki.

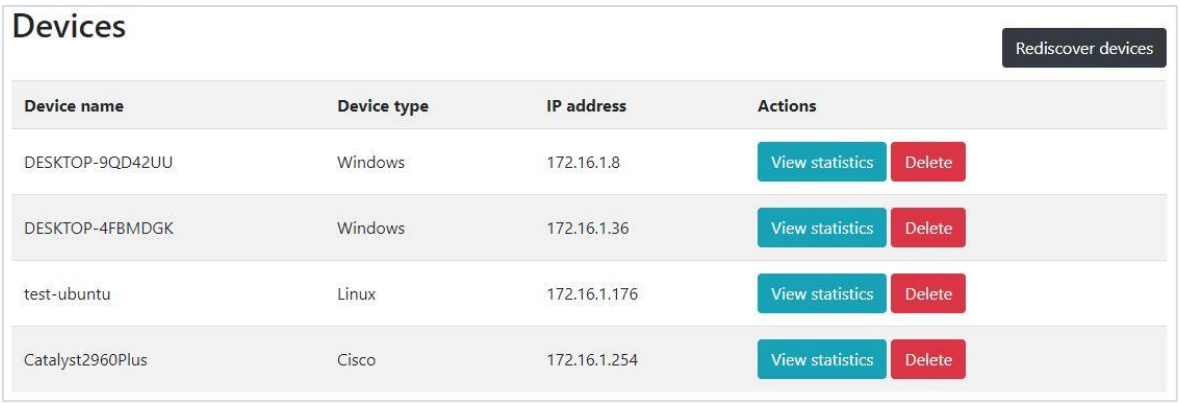


8.2 ábra: Az alkalmazás navigációs sávja

Szerepe a felhasználó alkalmazásban történő navigálásának gyorsítása, ezért szinte minden felület elérhető általa. Ezen felül jelzi az aktuálisan belépett felhasználót, valamint a kijelentkezés gombja is itt található meg. Kialakítása miatt könnyedén bővíthető új menüpontokkal.

8.2.3 Eszközök

Az eszközöket táblázatos formában listázó felület, amely az alkalmazás főoldalaként is szolgál, hiszen bejelentkezés után minden esetben ide érkezik a felhasználó. Az eszközök nevét, típusát, IP címét, valamint a velük végezhető műveleteket mutatja.

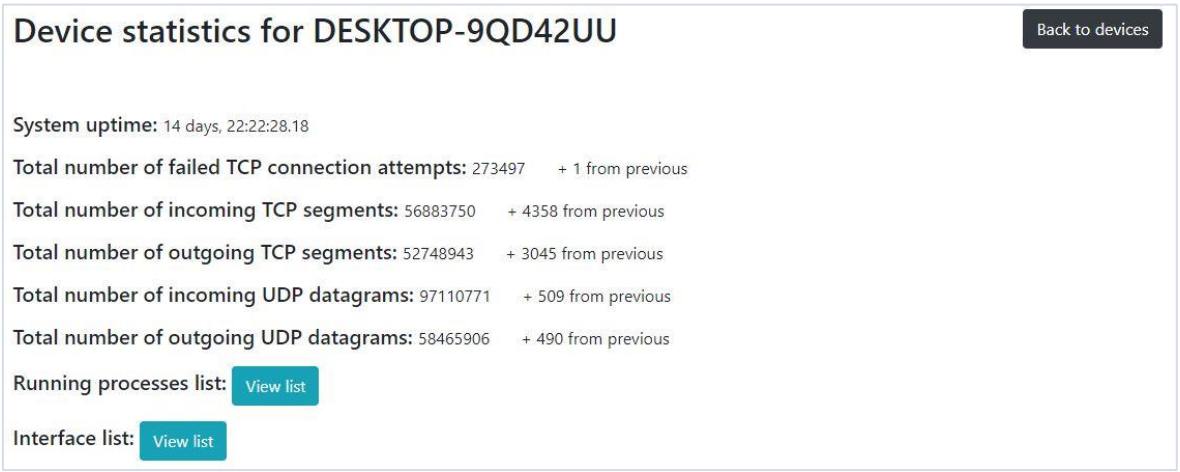


8.3 ábra: Az eszközök listája

Az eszközök típusa a jelenlegi állapot szerint *Windows, Linux, Cisco* vagy *Unknown* lehet. A táblázat jobb felső sarkánál található „*Rediscover devices*” gomb segítségével a teljes eszközfelderítés újrafuttatható úgy, hogy a meglévő eszközöket először törli a rendszer, míg a „*Delete*” gombok csak az adott eszközt törlik. A véletlen kattintások elkerülése miatt, mindkét műveletkor először egy megerősítést kérő ablak ugrik fel. Végrehajtásuk után az oldal automatikusan frissül, ezáltal egyből látható a sikeresség. A „*View statistics*” gombok pedig a megfelelő eszköz statisztikai oldalára navigálnak.

8.2.4 Eszköz statisztikai adatai

Az előző oldalról idenavigálva az adott eszközről gyűjtött adatokat láthatjuk, amelyek automatikusan frissülnek az újonnan beérkezett eredményekkel. Az oldal tetején az eszköz nevét, illetve egy gombot találunk, amely visszavisz az eszközök listájára.

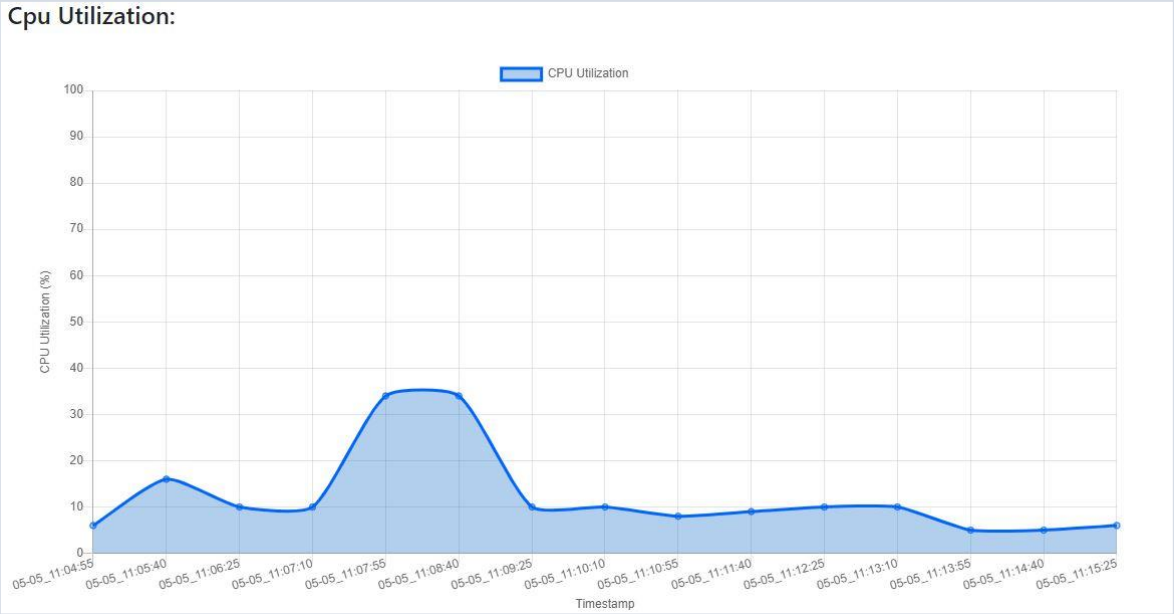


8.4 ábra: A statisztikai oldal felső része

A felső részen kaptak helyet a szöveges adatok, illetve a további oldalakra navigáló gombok. Itt láthatjuk az eszköz elindítása óta eltelt időt, a sikertelen TCP kapcsolatkiépítések számát, az összes bejövő, illetve kimenő TCP szegmens és UDP datagram számát. Ezek mellett olvasható az előző méréssel való összehasonlítás is. Lehetőségünk van megtekinteni az eszköz interfészeinek, illetve futó processzeinek listáját is a megfelelő gombok segítségével.

A felület figyelembe veszi az adott eszköz típusát, aszerint jelenít meg, vagy tűntet el arra nem vonatkozó adatokat. Például Cisco eszközök esetén nem beszélhetünk futó processzekről, azonban elérhető az eszköz hűtőventilátorának állapota.

Az oldal további részén a grafikonnal ábrázolható adatok jelennek meg, melyek a processzor- és memóriakihasználtság, futó processzek száma, valamint az aktív TCP kapcsolatok mennyisége.

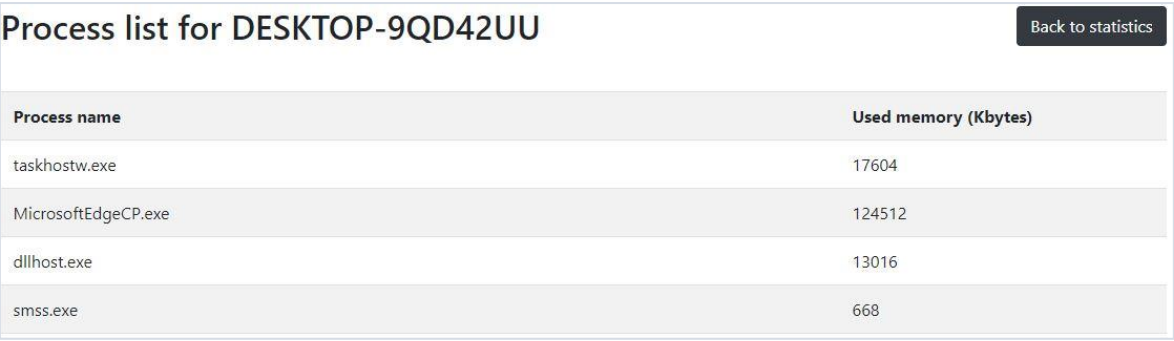


8.4 ábra: A processzorkihasználtság grafikonja

A grafikonok jelenleg minden esetben maximum az utolsó 15 mért értéket mutatják, de ennek növelése vagy csökkentése kis változtatással megoldható a kódban. A kurzorral rámutatva a görbén lévő pontokra, kiíródik a pontos érték. Hasonlóan van a többi említett adat is megjelenítve. Cisco eszközöknél a memóriakihasználtság nem százalékot mutat, hanem a szabad memória méretét.

8.2.5 Futó processzek listája

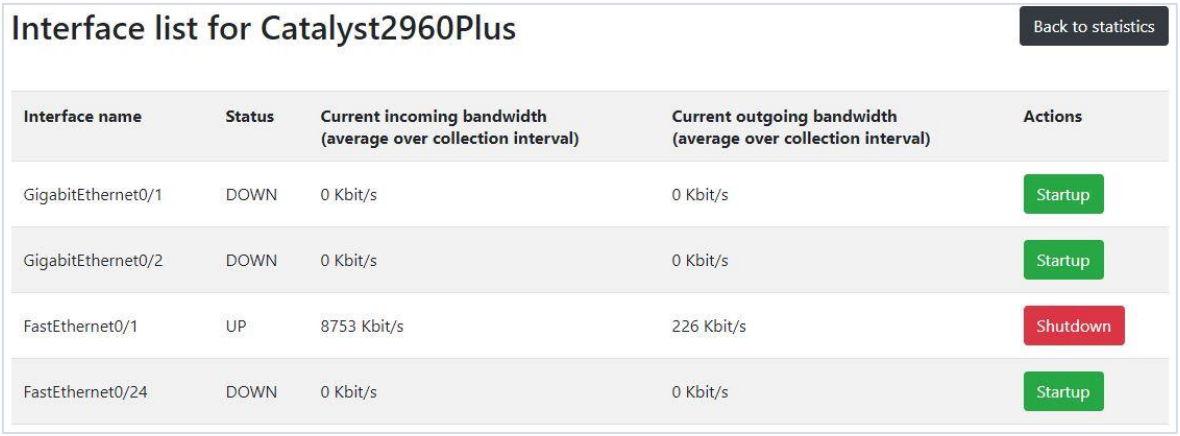
Táblázatos formában felsorolja az adott eszközön futó processzeket, valamint az általuk felhasznált memória méretét kilobájtban.



8.5 ábra: Eszközön futó processzek listájának részlete

8.2.6 Interfészek listája

Hasonló táblázatban felsorolja az adott eszköz hálózati interfészeit. Ezenfelül látható az interfészek aktuális állapota, valamint a bejövő és kimenő forgalmuk kilobit/másodpercben.



8.6 ábra: Eszköz interfészlistájának részlete

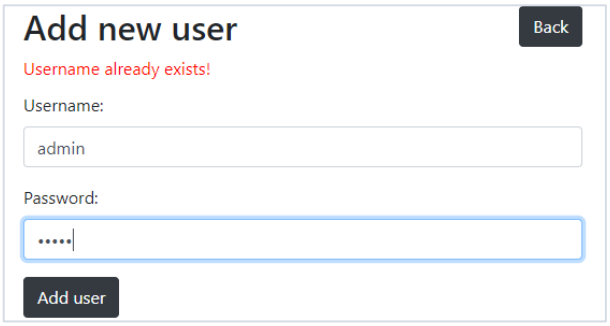
Cisco eszközök esetében a táblázat kiegészül az interfészeket elindító vagy leállító gombokkal is, amelyek a státusz függvényében értelmezhető műveletet mutatják. Az oldal megadott időközönként frissít, így mindig az utolsó mérés értékeit láthatjuk.

8.2.7 Felhasználók

Az „Eszközök” oldalhoz nagyon hasonló felület, amely a program által ismert felhasználók listáját mutatja táblázatban. Itt van lehetőség felhasználók törlésére, valamint innen juthatunk el az új felhasználó felvételére szolgáló oldalra.

8.2.8 Új felhasználó hozzáadása

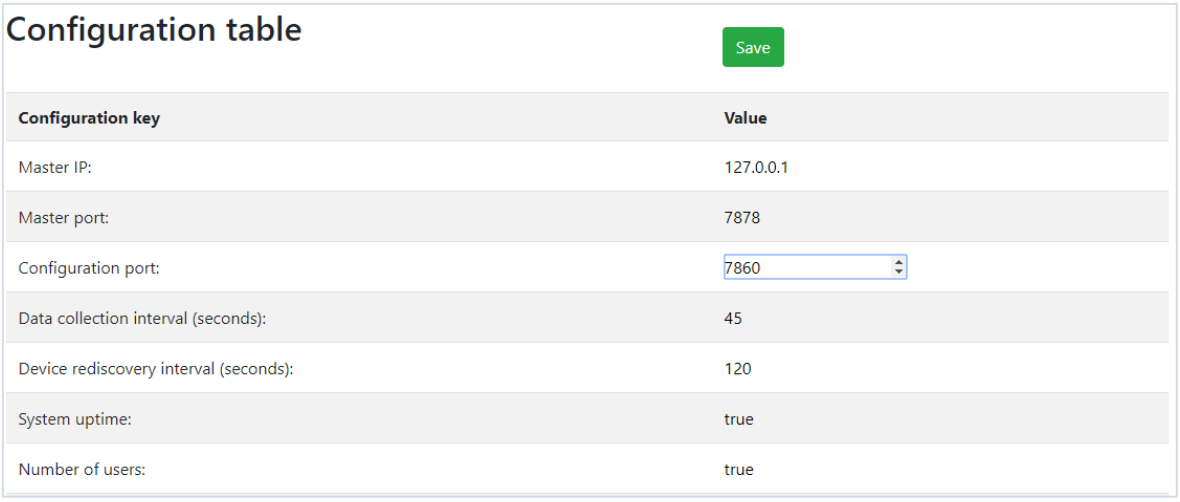
Ezen az oldalon tudunk felhasználónév és jelszó megadásával új felhasználót hozzáadni a rendszerhez. A felület validálja a mezőket, ezért kitöltésük kötelező. Az alkalmazás hibaüzenettel jelzi, ha már létező felhasználót próbálunk felvenni.



8.7 ábra: Új felhasználó hibás hozzáadása

8.2.9 Konfiguráció

A konfigurációs felületen a már megszokott táblázatos formában láthatjuk a program éppen aktuális beállításait, valamint a mezőket átírva módosíthatjuk azokat.



8.8 ábra: Konfigurációs felület részlete

A felület a mezők mindegyikére alkalmaz bizonyos feltételeket. A számmal értelmezhető beállításoknál csak szám megadását engedélyezi, valamint a „Master IP” mezőbe csak IP-cím formátumot fogad el. A további sorokban azt adhatjuk meg, hogy mely adatokat gyűjtse az alkalmazás. Ezek a mezők a „true” vagy „false” értéket engedik meg. A mentés sikeressége esetén, arról egy felugró ablak tájékoztat. Hibaüzenetet pedig akkor kaphatunk, ha a megadott konfiguráció helytelen, vagy nem változtattunk semmin.

8.2.10 Napló

Ezen az oldalon az alkalmazás által eltárolt naplóbejegyzéseket olvashatjuk el táblázat formájában, illetve láthatjuk, hogy azok mikor kerültek bejegyzésre. Minden fontos esemény tárolásra kerül, például felhasználó- és eszköztörlés, sikeres eszközfelderítés és adatgyűjtés, sikertelen adatküldés a központ felé, valamint eszköz elérhetetlenné válása stb.

8.2.11 Elküldetlen eszközadatok

Az eddigiekhez hasonlóan listázza a központi szerver felé sikertelenül továbbított, újraküldésre váró eszközadatokat, illetve azok letárolásának idejét. Magukat az adatokat nyers JSON formátumban mutatja. Sikeres elküldéskor törlődnek a táblából.

8.3 Az alkalmazás tesztelése

Az alkalmazás központi szerverrel való kommunikálásának tesztelése miatt elkészítettem egy egyszerű parancssoros programot, amely szimulálni tudja azt. A különböző, előre megadott tesztkonfigurációk küldésére az alábbi parancsok adnak lehetőséget: „*invalid_config*”, „*invalid_token_config*”, illetve „*valid_config*”. A rosszul formázott, hibás tokent tartalmazó, valamint helyes konfiguráció tesztelését teszik lehetővé. Lényegük, hogy ellenőrizhető legyen az elkészült ágens ezekre adott válasza. Az „*invalid_config*”-ra megjelenő logüzenet: „*Incoming configuration from /127.0.0.1 is missing field: rediscover_interval*”, míg az „*invalid_token_config”*-ra a következő választ kapjuk: „*Incoming configuration from /127.0.0.1 has invalid token!*”. Ezen üzenetekből látszik, hogy a hibás eseteket megfelelően lekezeli az alkalmazás. A beállítások változatlansága természetesen a felületen is ellenőrizhető. A „*valid_config*” parancs megváltoztatja az ágens által figyelt portot is, ezért az alábbi üzenetet adja: „*New agent configuration from /127.0.0.1 saved!, Listening for configurations on port: 7860”.* Láthatjuk, hogy megváltozott a figyelt port, illetve mentésre került az új konfiguráció, amelyet a felület is mutat.

Ezenfelül az adatok továbbításának tesztelése érdekében, a tesztprogram egy *ServerSocket* segítségével hallgat a megfelelő porton, amely az egyszerűség kedvéért az ágens alapértelmezett beállításaiban megadott port. Csak az ágenst elindítva, adatgyűjtéskor az a következőket logolja: „*Data collection run at: …, Log could not be sent to master, stored to DB instead*!”, tehát a központi szerver elérhetetlensége miatt az adatbázisba menti az eszközadatokat. A tesztprogram elindítása után már nem kapunk hibaüzenetet, a felület „Elküldetlen eszközadatok” táblája is kiürül, valamint a tesztprogram parancssorában láthatjuk az oda beérkező üzeneteket. Így meggyőződhetünk az adatküldés helyességéről.

A Cisco eszközök interfészeinek elindítását és leállítását azok konzoljának megtekintésével a legegyszerűbb tesztelni. Például a „*FastEthernet0/3”* port leállítása esetén az alábbi üzenet kell, hogy megjelenjen:

```
May  6 14:55:05.983: %SYS-5-CONFIG_I: Configured from 172.16.1.8 by snmp
May  6 14:55:07.971: %LINK-5-CHANGED: Interface FastEthernet0/3, changed state to administratively down
May  6 14:55:08.978: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/3, changed state to down
```

Az interfészre csatlakoztatott hoszton ezáltal a hálózati kapcsolat megszűnik. Elindításkor pedig a kapcsolatnak helyre kell állnia, valamint a következő üzenetet kell, hogy kapjuk:

```
May  6 14:57:54.983: %SYS-5-CONFIG_I: Configured from 172.16.1.8 by snmp
May  6 14:57:56.510: %LINK-3-UPDOWN: Interface FastEthernet0/3, changed state to up
May  6 14:57:58.210: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/3, changed state to up
```

Magát az adatgyűjtést a hálózaton lévő eszközök használatával és a megfelelő értékek figyelésével tesztelhetjük. Például egy hoszt processzorának vagy memóriájának terhelése meg kell, hogy jelenjen az alkalmazás felületén látható grafikonokon. Hálózati kapcsolatának terhelése az interfészek listájában a mért értékek növekedését okozza.

Az alkalmazás hatékonyságát, teljesítményét főképp az azt futtató számítógép határozza meg. Minél több szálat képes az eszköz kezelni, annál gyorsabb lesz a program műveleteinek lefutása. Mivel a Java 8 adta lehetőségeket használtam, így a párhuzamosan futtatható funkciók automatikusan skálázódnak. Az általam használt hoszt, az átlagos *255.255.255.0*-ás maszkkal rendelkező alhálózat lehetséges címeit (254 db) körülbelül 5 másodperc, míg a megengedett legnagyobb, *255.255.0.0*-ás maszkú alhálózat címeit (65 024 db) 20 perc alatt járja végig eszközfelderítés céljából. Az adatgyűjtés hasonlóan megoldott párhuzamosításával pedig a kialakított teszthálózat eszközeiről szinte időkülönbség nélkül képes lekérdezni az értékeket.

8.4 Fejlesztési lehetőségek

Az alkalmazás implementálása közben számos továbbfejlesztési ötlet fogalmazódott meg bennem. A megírt kód struktúrájának kialakításakor is fő szempontjaim között volt a bővíthetőség és átláthatóság, amely elvek támogatják az új funkcionalitások könnyebb és gyorsabb fejlesztését. A program által jelenleg gyűjtött adatok típusai főképp szemléltetései az SNMP által lekérdezhető információknak, mintsem konkrét problémát reprezentáló értékek. Ebből kifolyólag az egyik legfőbb fejlesztési lehetőségként a gyűjtött adatok körének szélesítését látom. Az alkalmazás gyors átalakíthatósága miatt, akár adott felhasználási környezetben releváns adatokra is testreszabható.

Emellett egy összekapcsoltabb, de mégis decentralizált rendszer érdekében, célszerű lehet az alhálózatokon elhelyezkedő ágensek egymással történő kommunikációjának kialakítása. Például az eszközadatok redundáns, több ágensen eloszló tárolása még hibatűrőbbé tenné a felügyeletet. Végül nagyobb működésbeli fejlesztési lehetőségnek gondolom még új eszköztípusok támogatását, Windows rendszerek esetében akár a WMI használatát.

A webfelületen is számos fejlesztési lehetőség kigondolható. Például a konfiguráció átláthatóbb kijelzése, az eszközök statisztikáinak rendezettebb elosztása, valamint a napló és sikertelenül elküldött adatok oldalainak lapozhatóvá tétele.

Összegzés

Dolgozatom célja a számítógép-hálózatok elosztott felügyeletének tanulmányozása, valamint a megszerzett ismeretek alapján egy ágens megtervezése és implementálása volt.

A számítógép-hálózatok felügyeletének napjainkban egyre nagyobb szerepe van, hiszen a technológia fejlődésével az illegális felhasználási formák száma is jelentősen növekszik. A legtöbb, jelenleg létező monitorozó rendszer, architektúra szempontjából két típusba sorolható. Az ágensalapú szoftverek minden megfigyelni kívánt eszközre igénylik a rendszer ágenseinek telepítését, ezáltal részletes eszközadatok gyűjtésére képesek. Telepítésük és üzemeltetésük azonban bonyolult, valamint költségigényes. Ezzel szemben az ágens nélküli megközelítés előre telepített protokollok segítségével végzi az adatok ellenőrzését, ezáltal elég egy alkalmazás installálása. Költséghatékonyabb, illetve fenntartása sem túl komplex, azonban sok esetben nem képes összetettebb hosztadatokat monitorozni, főként a hálózati forgalmat figyeli. Hálózati problémák esetén nincs információja az eszközökről.

Az említett pozitívumokat és negatívumokat figyelembe véve, a feladatom elkészítése során egy olyan alkalmazást terveztem és implementáltam, amely a lehető legjobban megvalósítja mindkét típus előnyeit, miközben hátrányaikat mérsékeli.

A tervek elkészítését megelőzően, először megismerkedtem az SNMP felügyeleti protokoll felépítésével és működésével, illetve egy létező hálózat-felügyeleti szoftverrel. A kutatás során szerzett, valamint saját szoftverfejlesztési tapasztalataim alapján megterveztem a program funkcionalitásait és a hozzájuk szükséges komponenseket. A kialakított komponensek lefejlesztéséhez megfelelő technológiák kiválasztása után összeállítottam egy teszthálózatot, amelyen a készülő szoftver tesztelhető.

Végezetül bemutattam az elkészült alkalmazás komponenseinek technikai részleteit és felületeit, ügyelve az átlátható és könnyen érthető struktúrára. Kitértem a működés tesztelésére, valamint az elkészítés alatt megfogalmazódott fejlesztési lehetőségekre is.

Az elkészült program képes tehát minden, a saját alhálózatán lévő SNMP-t használó eszközről információt gyűjteni, valamint ezeket a központi állomás felé küldeni, amely az adatok feldolgozását végzi. Annak elérhetetlensége esetén is tovább tud működni, majd a kapcsolat helyreálltakor a mentett adatokat automatikusan továbbítani. Használatával kialakítható egy olyan rendszer, amely a központi szerver szemszögéből ágensalapú, viszont az alkalmazás részéről ágens nélkülinek tekinthető.

Summary

The goal of my thesis was to research the distributed monitoring of computer networks, also design and implement an agent with the gained knowledge.

Computer network surveillance has an increasing role in our current times, because as technology advances, the number of ways to illegally use networks are growing fast as well. Most currently existing monitoring systems can be divided into two main architectural categories. The agent-based softwares require the agents to be installed on all supervised devices, so they can collect detailed information from them. However, their installation and maintenance are complex and costly. On the other hand, the agentless approach collects data by using pre-installed protocols, so the agent can run on just one computer. It is more cost effective and maintenance isn't complicated, but in many cases, it can't gather complex host information. It is mostly monitoring network traffic, but if the network fails, it has no information about the devices.

Taking the mentioned advantages and disadvantages into account, in my thesis I designed and implemented an application, that realizes the positives as much as possible, while also mitigating the negatives.

Before making plans for the program, I firstly got acquainted with the SNMP protocol's structure and operation, as well as with an already existing network monitoring solution. I then designed the functionality and the necessary components of the application, using the acquired knowledge from research, along with my own experiences in software design. After choosing the appropriate technologies to implement the components, I created a network for testing purposes during development.

Finally, I presented the completed components and web interfaces of the application in technical detail, while giving attention to a clear and understandable structure. I mentioned testing of the program's operation, as well as further development ideas.

The finished application can collect device information from all SNMP capable devices on it's sub-network, then send the data to the central server, which processes it. The program can also keep running, if it doesn't have an available connection to the central server. It saves all unsent data and resends it when the connection becomes active. Usage of my application can create a system, that is agent-based from the viewpoint of the central server, while being agentless from the viewpoint of the application.

Irodalomjegyzék

[1] http://www.machinedesign.com/controllers/basics-industrial-network-management-software

[2] https://www.advancedcyber.co.uk/the-definitive-guide-to-network-monitoring#key_terms__protocols

[3] https://www.cisco.com/c/en/us/support/docs/availability/high-availability/15114-NMS-bestpractice.html

[4] https://www.snmpcenter.com/fcaps-network-management/

[5] https://www.tankonyvtar.hu/hu/tartalom/tamop425/0005_24_szamitogepes_halozatok_scorm_03/333_az_osi_modell.html

[6] https://www.helpsystems.com/resources/articles/what-agentless-network-monitoring-software

[7] https://www.eginnovations.com/product/agentless-monitoring

[8] https://searchnetworking.techtarget.com/definition/SNMP

[9] https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol

[10] https://tools.ietf.org/pdf/rfc3584.pdf

[11] https://tools.ietf.org/pdf/rfc3411.pdf

[12] https://linuxadm.hu/cikk/A-Nagios-lehetosegei

[13] https://www.nagios.org/documentation/

Képek, ábrák forrása

2.1 ábra http://www.codrm.eu/conferences/2012/34_Barbu.pdf

2.2 ábra https://hu.wikipedia.org/wiki/OSI-modell#/media/F%C3%A1jl:OSI_mod_2.png

2.3 ábra https://www.inetco.com/app/uploads/2011/09/AGENT_DIAGRAM1.jpg

2.4 ábra https://www.inetco.com/app/uploads/2011/09/AGENTLESS_DIAGRAM.jpg

3.1 ábra https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol#/media/File:SNMP_communication_principles_diagram.PNG

4.1 ábra https://s24255.pcdn.co/wp-content/uploads/2012/02/Fedora-64-bit-@-2012-02-15-220038.png

6.1 ábra http://www.snmp4j.org/UMLClassOverview/UMLClassOverview.htm

6.2 ábra https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#/media/File:MVC-Process.svg

Linkek utoljára ellenőrizve: 2020.05.11.

Köszönetnyilvánítás

Köszönöm Dr. Kovács Szilveszter témavezetőmnek a segítőkész támogatását, hasznos tanácsait, valamint iránymutatásaival a szakmai fejlődésemhez való hozzájárulását.

I would also like to thank my consultant, Almseidin Mohammad Abdallah Suleiman, PhD for his contributions towards the research for my thesis.

A tanulmányban ismertetett kutatómunka az EFOP-3.6.1-16-2016-00011 jelű „Fiatalodó és Megújuló Egyetem – Innovatív Tudásváros – a Miskolci Egyetem intelligens szakosodást szolgáló intézményi fejlesztése” projekt részeként – a Széchenyi 2020 keretében – az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

CD melléklet

A szakdolgozatomhoz mellékelt CD tartalma:

Dolgozat könyvtár:

- Dolgozatot tartalmazó fájl docx szerkeszthető formátumban
- Dolgozatot tartalmazó fájl pdf formátumban
- Kiírást tartalmazó fájl pdf formátumban
- A dolgozat magyar összegzését tartalmazó fájl docx szerkeszthető formátumban
- A dolgozat magyar összegzését tartalmazó fájl pdf formátumban
- A dolgozat angol összegzését tartalmazó fájl docx szerkeszthető formátumban
- A dolgozat angol összegzését tartalmazó fájl pdf formátumban

Forrásfájl könyvtár:

- Alkalmazásom teljes Maven projektje
- Alkalmazásom futtatható *jar* fájlja