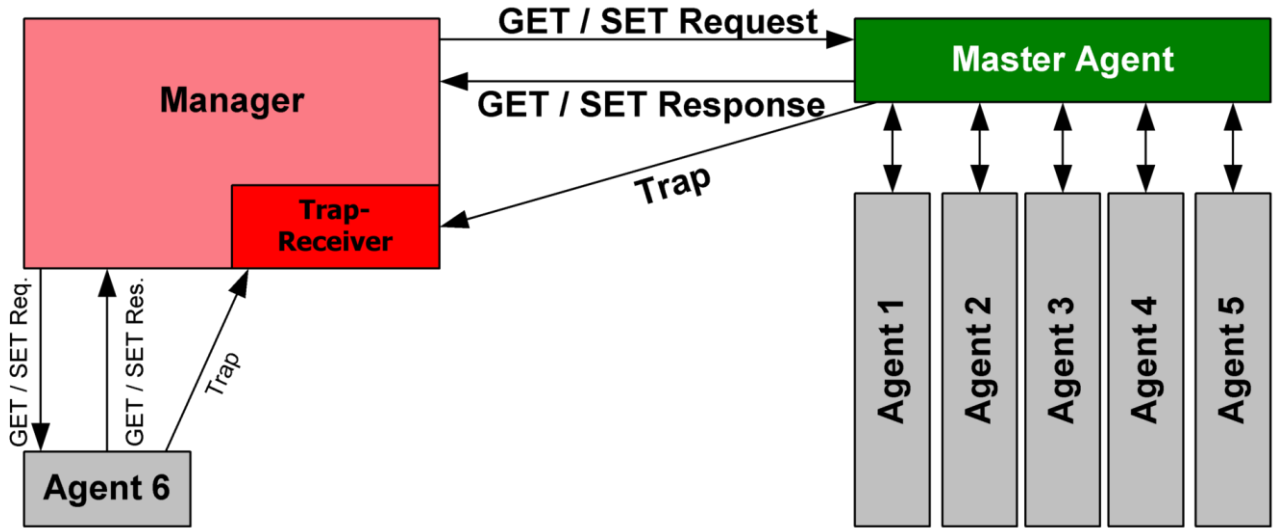
Съдържание

Увод

Още от 80-те години на 20 век хората са имали нужда от инструменти и средства за мониторинг на техните мрежи и устройства. С развитието на технологиите тези нужди са станали все по - ясно изразени и все по - належащи. Дори и при малки мрежи и системи които се състоят от няколко до няколко десетки устройства това може бъде предизвикателство поради ред на брой причини като различният софтуер, vendor, типа на устройствата (компютри, рутери и други) и други. Нуждата от стандартизирано, централизирано и най – вече автоматизирано проследяване от единен портал и единен протокол е била движеща роля през тези години. Една от големите стъпки които са направени в тази посока е създаването на SNMP протокола през 1988 година.



Фиг. 0.1 Принцип на SNMP комуникацията

В следващите глави ще разгледаме в детайли какво представлява SNMP протокола. Това което искам да въведа тук като встъпителна информация е, че този протокол стандартизира взимането на информация за различни параметри на различни видове устройства. Устройства които типично поддържат SNMP са модеми, маршрутизатори, комутатори, сървъри, работни станции, принтери и други. Както се вижда проблемите със “стандартизирано” и “централизирано” проследяване биват отчасти решени с този протокол. Колкото до “автоматизирано” – това се опитваме да решим със текущата дипломна работа. И не само това, опитваме се да решим проблема и с взимането на навременни мерки (изпълняване на действия при определени входни условия).

Глава 1 - Обзор - състояние на проблема по литературни данни

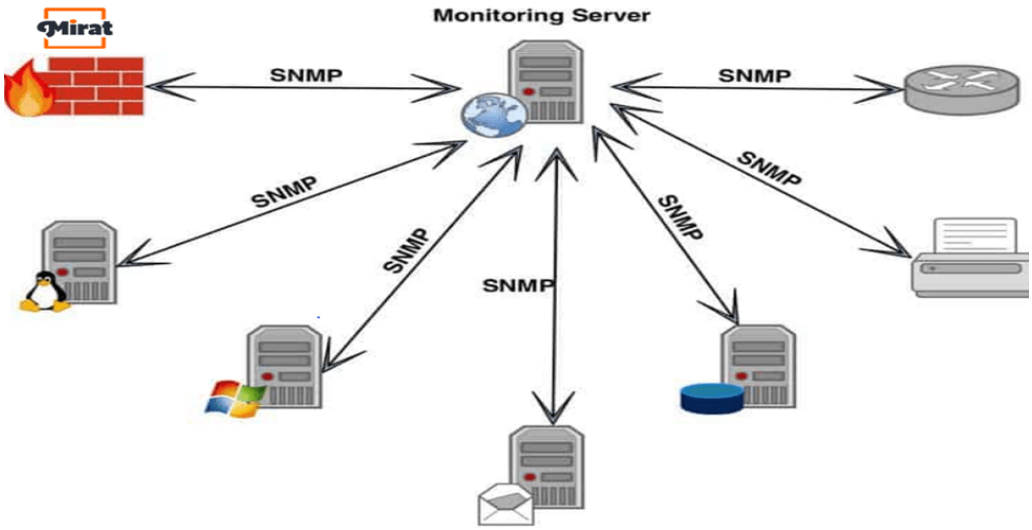
Един от основните проблеми които хората срещат рано или късно в експлоатацията на техните системи е мониторинга на устройствата които използват. Всеки хардуер има срок на експлоатация и потенциални проблеми винаги може да възникнат. Проблеми могат да се появят и от липса на дисково пространство, недостиг на оперативна памет, недостиг на изчислителна мощ и други. Колкото по - гъвкави сме в техният мониторинг и ранното взимане на действия, толкова повече време ще имаме за реакция. Дори и големи и сложни enterprise системи (например VMware Vsphere) се сблъскват с тези проблеми. Когато клиентите имат в някои случаи стотици или хиляди устройства и всякакъв вид хардуер, и разчитат, че системите които работят върху тях ще са 24/7 в готовност, мониторинга е от критична важност. Най – често подобни системи си имат екипи които се занимават изцяло с това и пишат специализиран софтуер който следи стотици параметри и се опитва да покаже всеки потенциален проблем на потребителите в лесен потребителски интерфейс.

Един от основните проблеми с този подход е, че най – често при наличие на проблем се показват нотификации/съобщения в потребителския интерфейс, и хората трябва сами да видят какъв е проблема и трябва да са достатъчно компетентни, че да знаят как точно да го разрешат.

Друг от основните проблеми е, че самите потребители нямат голяма свобода в процеса на самия мониторинг, имат възможност да видят тестовете които системата предоставя и ако има проблем, трябва сами да правят troubleshooting къде точно е проблема защото понякога самата грешка не е много разбираема.

Още един от основните проблеми е, че автоматизацията на това какво трябва да се случи в дадена ситуация не е нещо тривиално, и понякога включва нуждата от дълбоко разбиране на самият софтуер. Ситуацията става още по сложна и когато трябва да се използват публични API-та предоставени от самата система. Също така с всяка нова версия на продукта нещата се променят и хората трябва да са постоянно наясно с всички новости и промени.

SNMP (Simple Network Management Protocol) е протокол, който се използва за обмен на управляваща информация между устройствата. Той е включен в приложния слой на TCP / IP, както е дефинирано от Internet Engineering Task Force (IETF).

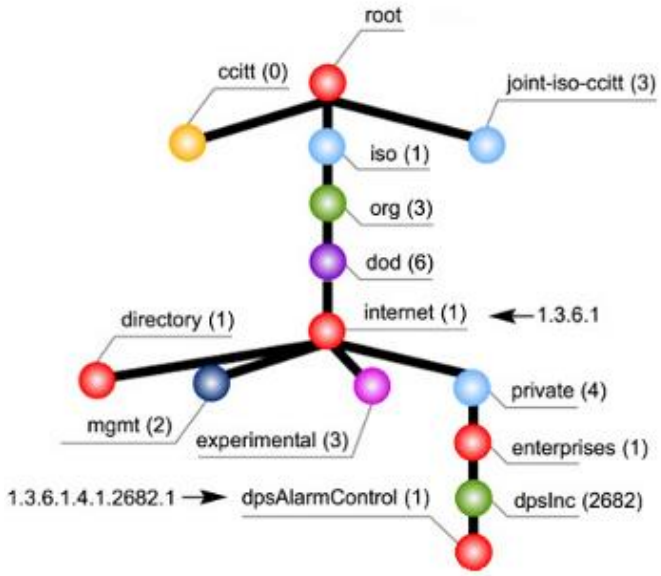


Фиг. 1.1 SNMP overview

Компоненти на SNMP:

MIB (Management Information Base)

База данни, в която са дефинирани обектите, които ще бъдат предоставени на всеки управляван възел от SNMP агента.



Фиг. 1.2 SNMP MIB дървовидна структура

OID (Object Identifier)

Това е уникален идентификатор на статус/информация за дадено устройство. Формата му представлява стринг от числа както в Фиг. 1.3.

```
1.3.6.1.4.1.2682.1.4.5.1.1.99.1.1.6
```

Фиг. 1.3 Пример за OID

Всяко число представлява листо от MIB дървото. Ето и как би изглеждала информацията за OID-то от Фиг. 1.3.

Number	Label	Explanation
1	iso	ISO is the group that established the OID standard.
.3	org	An organization will be specified next.
.6	dod	The US Department of Defense (established the early internet).
.1	internet	Communication will be via Internet/ network.
.4	private	This is a device manufactured by a private entity (not gov't).
.1	enterprise	The device manufacturer is classified as an enterprise.

Number	Label	Explanation
.2682	dpsInc	This device's manufacturer is DPS Telecom Inc.
.1	dpsAlarmControl	This is an Alarm & Control Device built by DPS
.4	dpsRTU	This is a DPS RTU (Remote Terminal Unit)

Number	Label	Explanation
.5	AlarmGrid	We're working with a discrete alarm point (not a control relay or analog)
.1	AlarmEntry	An alarm point will be specified
.1	Port	This is the Port for this alarm point
.99	Address	This is the Address for this alarm point
.1	Display	This is the Dislay for this alarm point
.1	Point	This is the alarm Point number
.6	dpsRTUAState	This is the state of the alarm point (set, clear, etc.)

Фиг. 1.4 Таблица с описание на OID

SNMP агент
Софтуер, който отговаря на SNMP заявки от управляващият софтуер. Обектите на MIB са подредени в групи. Някои от тях се отнасят към съответните слоеве на протоколната архитектура, а други към системата като цяло.

Отдалечен мониторинг RMON
Софтуер, който осъществява с помощта на SNMP агентите отдалечен мониторинг на ресурсите на мрежата.

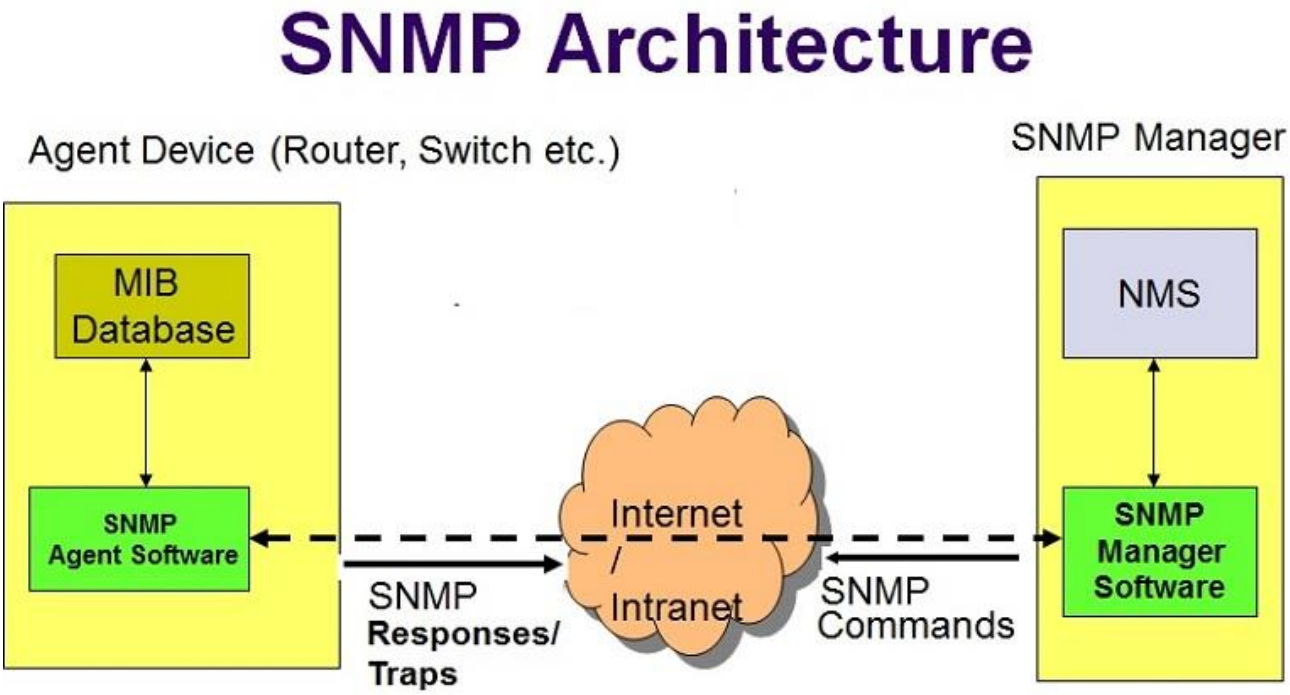
SNMP traps
Traps се наричат непоискани съобщения които биват пращани от агентите когато настъпи някакво важно събитие.

SNMP polling
Polling е когато управляващият софтуер иска от устройствата информация на определен интервал от време.

Основни функции на SNMP
Наблюдение и контрол. SNMP извършва ефективно управление и на най - малките локални мрежи. Протоколът осигурява общ механизъм, чрез който оборудване от различни производители може да бъде наблюдавано и контролирано от общо управляващо устройство. Към повечето от компонентите на една локална мрежа могат да бъдат добавяни SNMP агенти и те да бъдат наблюдавани и контролирани.

Използване на SNMP
Всички съвместими с SNMP устройства съдържат MIB, който доставя съответните атрибути на дадено устройство. Някои атрибути са фиксирани (твърдо кодирани) в MIB, докато други са динамични стойности, изчислени от софтуера на агент, работещ на устройството.

Обобщение
Софтуерът за управление на корпоративната мрежа, като например Tivoli и HP OpenView, използва SNMP команди за четене и записване на данни във всяко устройство MIB. Командите "Get" обикновено извличат стойности на данните, докато командите "Set" обикновено инициират действие на устройството. Например, скриптът за рестартиране на системата често се изпълнява в софтуера за управление, като се дефинира конкретен MIB атрибут и се издава SNMP Set от софтуера за мениджмънт, който пише стойност "рестартиране" в този атрибут.

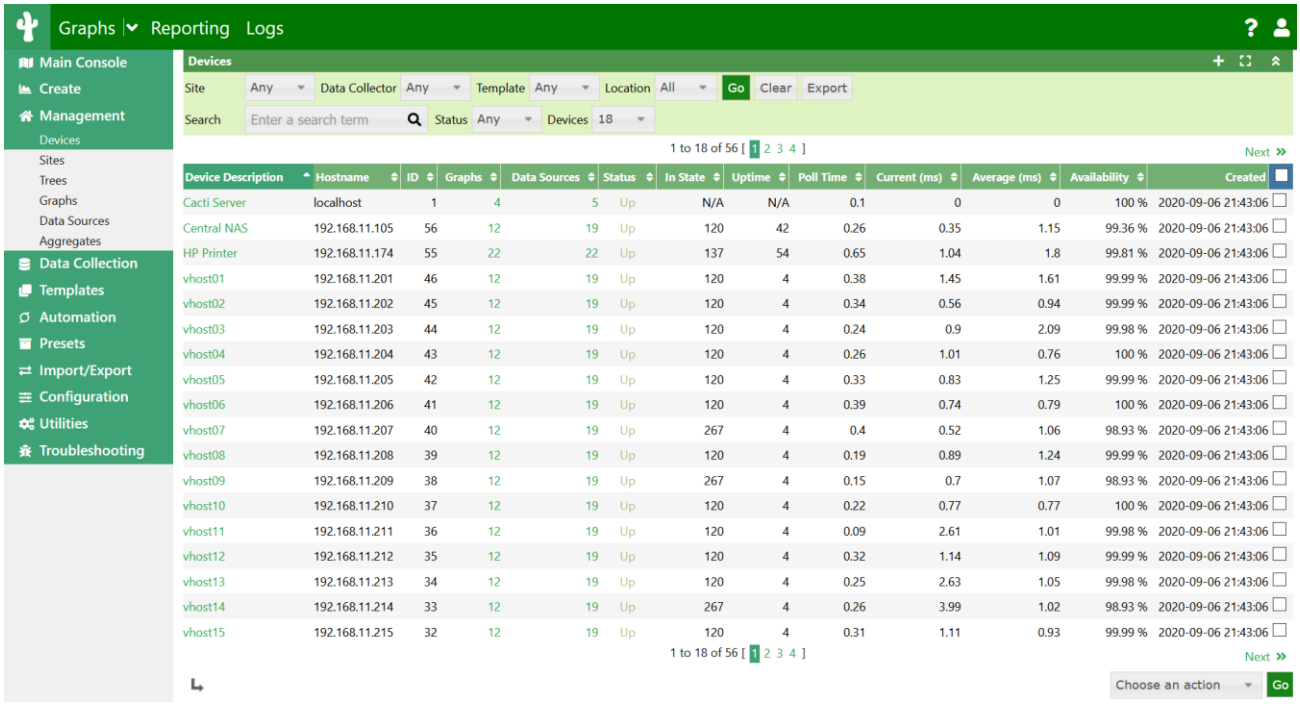


Фиг. 1.5 SNMP - Принцип на действие

Преглед на SNMP системи

В процеса на анализ разгледах няколко системи които използват SNMP протокола и предоставят множество функционалности.

Cacti

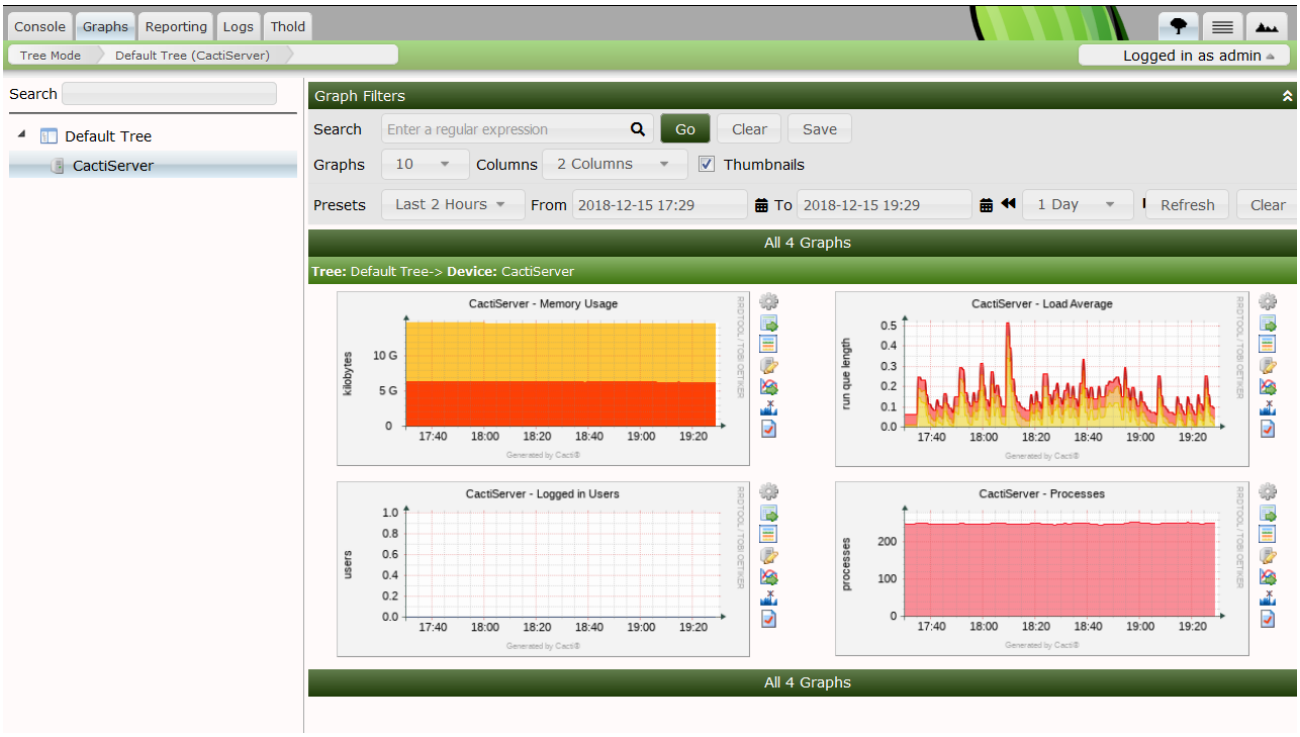


Фиг. 1.6 Cacti - изглед на екрана за устройства

Cacti е система която е насочена към мониторинг на мрежите и управление на проблемите които възникват в мрежовите устройства. Тя съхранява цялата информация в MariaDB или MySQL. Тя е Web базирана като за нейното създаване а използван LAMP стека (Linux, Apache, MySQL and PHP). В момента тя може да бъде инсталирана и на Windows посредством Nginx или IIS за Web Server.

В основата на Cacti са устройствата и шаблоните за тях. При добавяне на ново устройство се асоциира с шаблон и друга мета информация като например Site, Location и други. Спрямо тази информация Cacti ще създаде Graphs and Data Sources подходящи за добавеното устройство. Към днешна дата множеството plugin-и които са разработени за Cacti позволяват системата да бъде ползвана за performance management, fault management, log management, device discovery, router configuration backup, network mapping и други. По документация системата може да работи с до десетки хиляди устройства. Това е постижимо защото Data Collection framework-а може да бъде дистрибутиран и да се използва load balancers което позволява добро хоризонтално скалиране.

Data Source-те са обектите в Cacti които събират данните. Те са доста гъвкави и биха могли да бъдат освен на базата на SNMP протокола, така и скриптове и команди. Например ако искаме да направих Graph за ping в милисекунди, можем да направим скрипт който изпълнява ping към дадено устройство и връща времето в милисекунди. После можем да ползваме този Data Source за създаването на Graph който да покаже тази информация.
Друго интересно е Remote Data Collection. Те са малки програми които имат за цел да са по - близо до устройствата които проследяват. Те са свързани с централната база данни на Cacti посредством HTTPS. Те работят независимо от централната Cacti система и са отговорни за извличането и записването на данни като ако нямат връзка с базата данни, запазват локално всички данни и ги записват когато пак имат връзка с базата данни.



Фиг. 1.7 Cacti - изглед на екрана за графики

Cacti е софтуер който е безплатен и това е позволило с времето много хора да допринесат за неговото развитие.

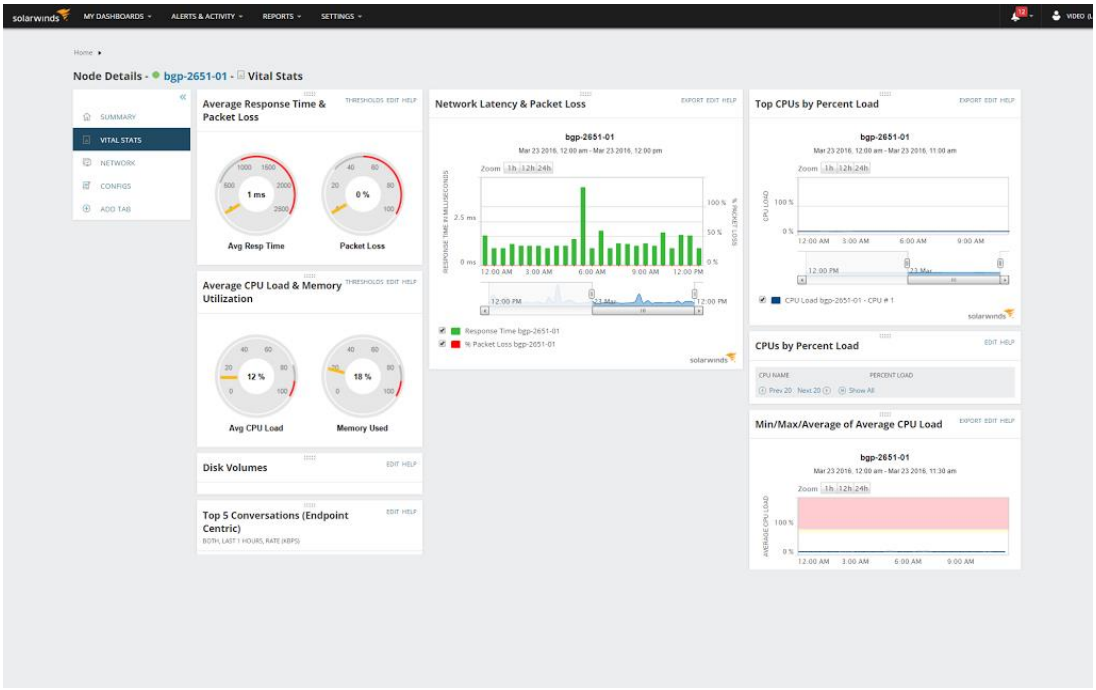
Това което ми харесва в Cacti е, че е безплатна, има голямо community, има всички основни функционалности които са нужни за подобен тип система. Това което мисля, че би могло да се подобри са да се намали сложността за извършването на по – прости операции, по - добър UX и добавянето на повече инструменти за анализ на мрежата и намиране на проблемните точки.

SolarWinds Network Performance Monitor (NPM)

Продуктите разработени под шапката на SolarWinds не се ограничават само до работата с SNMP протокола. Те предоставят множество различни инструменти които се надграждат и предоставят възможност за цялостен мониторинг и мениджмънт на мрежи и мрежови устройства. Ще разгледам само някои от feature-е и инструменти които предоставят.

Автоматично търсене на устройства в мрежата. SNMP polling feature-те които са предоставени в SolarWinds Network Performance Monitor (NPM) също така могат да бъдат ползвани и като SNMP скенер на мрежата. Това е много удобно за големи и динамични мрежи където устройствата могат да идват от различни vendor-и. С помощта на извлечените performance статистики, администраторите могат по - добре да оптимизират и да намират проблеми в бързодействието на устройствата.

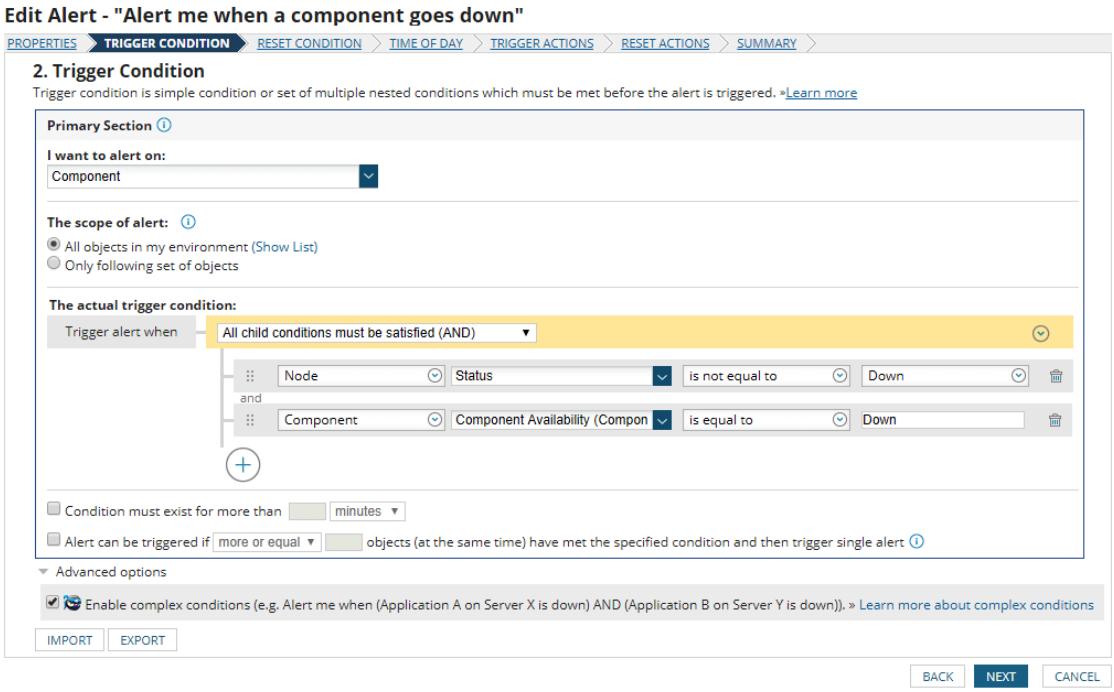
Следене за грешки, статус и бързодействие. Тези инструменти могат да събират информация за критични метрики свързани с бързодействието. Потребителят може да следи за грешки в мрежата, проблеми с устройствата и тяхното бързодействие на всички устройства които се поддържат автоматично, но има и възможност за ръчно предоставяне на OID за устройства които са непознати.



Фиг. 1.8 SolarWinds - Node details vital stats

Ползването на SNMP услугите в системата са лесни за настройка и инсталация. За SNMP v1 и SNMP v2 е необходим само community като входни данни, докато за v3 са необходими допълнителни credentials.

След като намирането и конфигурацията на устройствата приключи, системата предоставя единен портал през който потребителите могат да проверяват SNMP trap receiver-те, мрежовите устройства и цялостен мрежови мониторинг. Също така се предоставя достъп до мрежови нотификации, рапорти и интерактивни графики чрез които потребителите лесно могат да видят и проверят статуса на мрежата и да видят местата които имат проблеми с производителността. Системата предоставя възможност за много детайлни конфигурации които да позволят на потребителя да настрои нотификациите и други настройки спрямо неговите конкретни предпочитания.



Фиг 1.9. SolarWinds - Node trigger conditions details

Системата поддържа всички устройства които пращат syslog съобщения или поддържат SNMP, ICMP, API и WMI. Потребителят може да създава динамични и персонализирани графики и графове които да показват данните в реално време. Може да се използва и HTML от потребителя за още по голяма персонализация.

Фиг. 1.10 SolarWinds - Network summary

Това което ми харесва в тази система е, че разполага с много инструменти за визуализация и за проследяване на мрежата и лесното намиране на проблемните места. Освен това разполага и с много опции и инструменти с които потребителят може да персонализира графики, нотификации и други. Като цяло системата разполага с всичко необходимо. Минуса е, че е платена. Базовата цена към конкретният момент почва от 1335 евро.

Grafana Cloud

Grafana Cloud е надеждна, бърза и скалируема платформа за проследяване на потребителските приложения и инфраструктура. Тя предоставя централизиран изглед на данни както от Grafana Cloud Metrics, така и на такива предоставени от потребителската среда и cloud. Има вградена поддръжка за много от популярните data sources като Prometheus, Elasticsearch и Amazon Cloud Watch. Това което трябва да направи потребителя е да конфигурира data sources в Grafana Cloud.

Основни стъпки за ползване на SNMP с Grafana Cloud

- Потребителят трябва да има съществуващ или да си направи нов акаунт в Grafana Cloud.
- Трябва да се инсталира SNMP на потребителският Grafana Cloud instance.
- Grafana Agent трябва да бъде инсталиран и конфигуриран на сървър за да започне да изпраща метрики към потребителският Grafana Cloud instance. (Grafana Agent-а трябва да може да достъпи мрежовите устройства посредством UDP/TCP портове 161 и 162).

След приключване на интеграцията с SNMP, Grafana Cloud предоставя преконфигуриран dashboard за SNMP метрики. Ето някои от важните метрики които dashboard-а предоставя:

- ifHCInOctets
- ifHCOutOctets
- ifInErrors
- ifMtu
- ifOperStatus
- ifOutErrors
- ifPhysAddress
- ifSpeed
- ifType_info
- snmp_scrape_duration_seconds
- sysUpTime

Фиг. 1.11 Grafana Cloud - dashboard

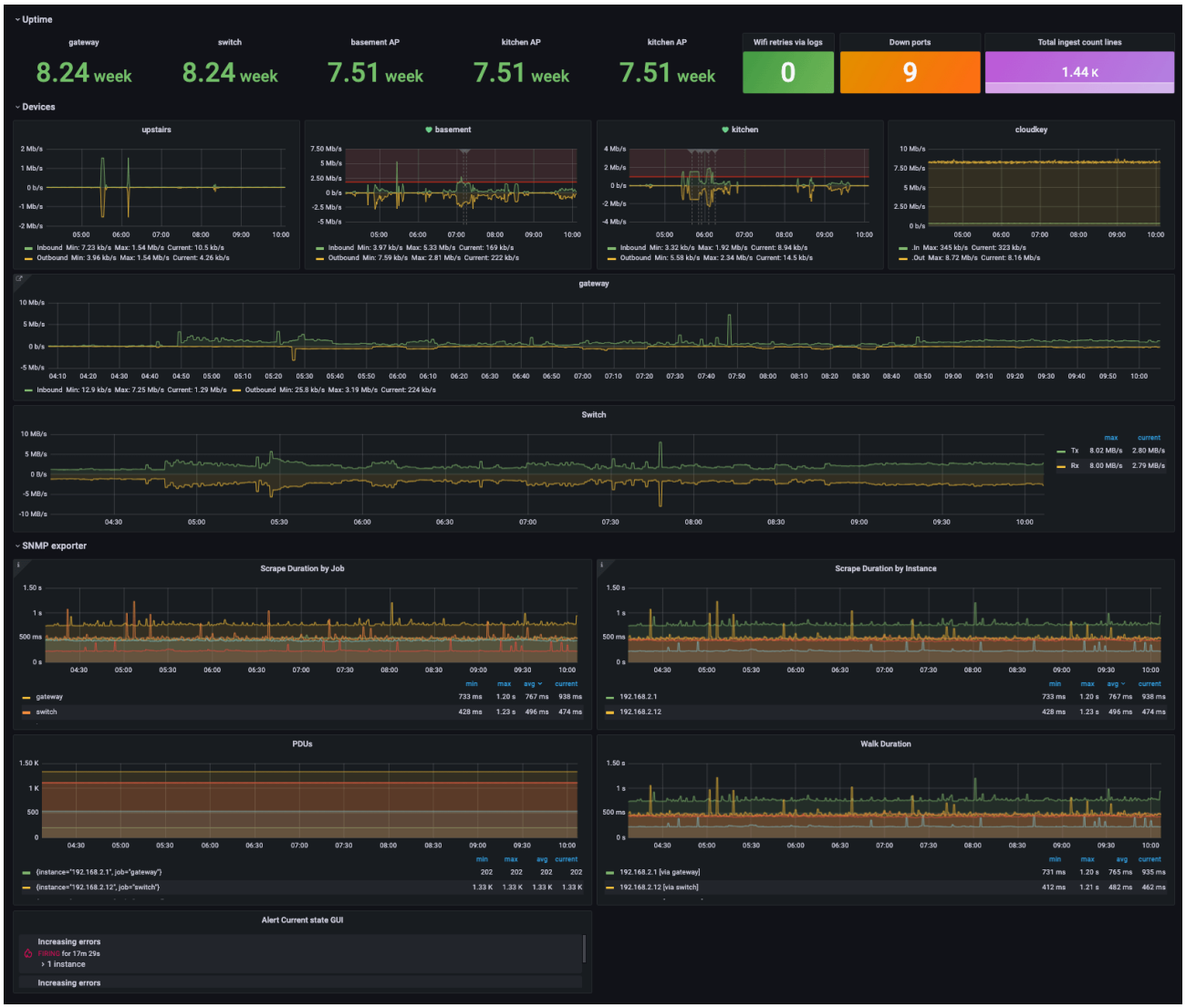
Както споменахме по – рано, част от интеграцията включва конфигурация на Grafana Agent. Част от него е вграденият snmp_exporter. Това става посредством YAML конфигурационен файл.

```
integrations:
  snmp: // enable-ва SNMP exporter интеграцията
    enabled: true
    relabel_configs:
      - action: replace
        source_labels: [job]
        regex: (^.*snmp)\/(.*)
        target_label: job_snmp
        replacement: $1
      - action: replace
        source_labels: [job]
        regex: (^.*snmp)\/(.*)
        target_label: snmp_target
        replacement: $2
    snmp_targets:
      - name: network_switch_1
        address: <host_address_1>
```

```
        module: if_mib
        walk_params: public
      - name: network_switch_2
        address: <host_address_2>
        module: if_mib
        walk_params: public
    walk_params:
      public:
        version: 2
        auth:
          community: public
prometheus_remote_write:
  - url: http://cortex:9009/api/prom/push
```

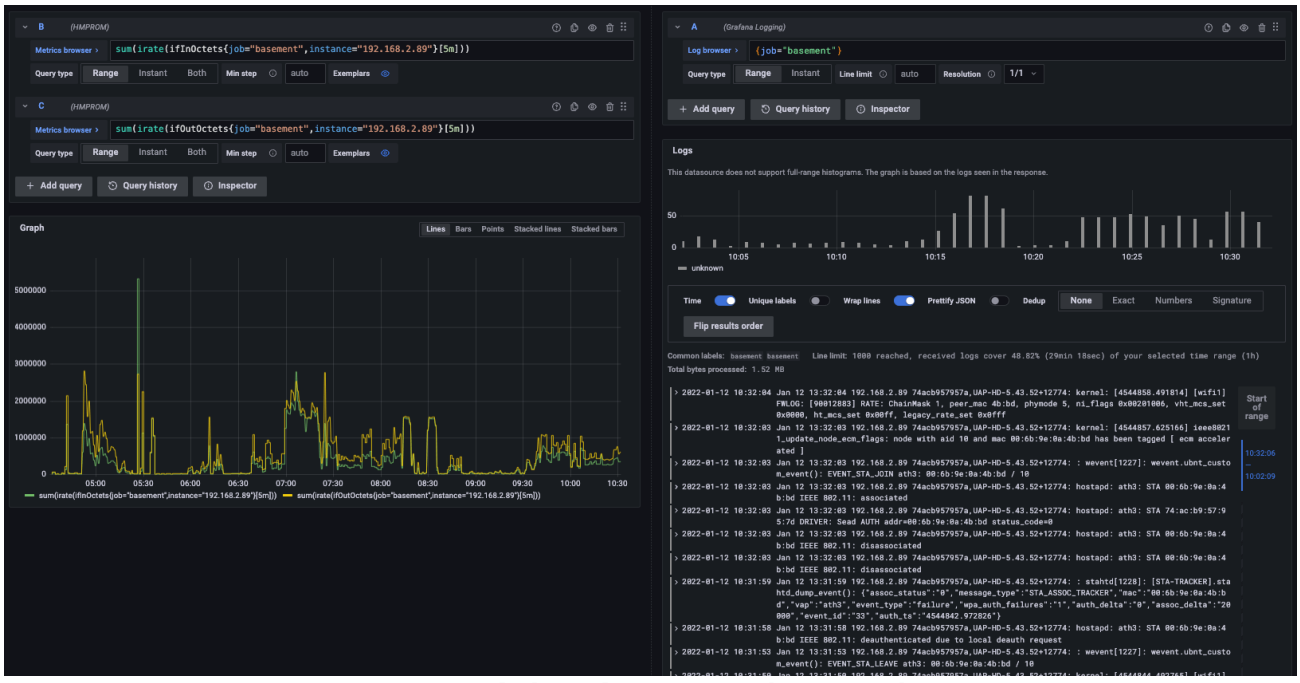
Фиг. 1.12 Grafana Agent – configure SNMP

Ако потребителят използва Prometheus, това може да бъде конфигурирано като част от тази конфигурация. Една от функциите на snmp_exporter е да служи като adapter между данните които биват получени в определен формат, и йерархичният формат на данните в SNMP протокола.



Фиг. 1.13 Networking dashboard

Фиг. 1.14 Traffic and packets dashboard



Фиг. 1.15 Side-by-side metrics to logs correlation

Това което ми харесва в тази платформа е лесната интеграция с други платформи. Освен това безплатната версия е достатъчна за малка мрежа с няколко устройства. Това което не ми харесва са лимитациите свързани с организацията и персонализацията на dashboard-вете. Друго нещо което би могло да се подобри е лимитираният брой на визуализации на различните типове данни.

Цели

Целта на текущата дипломна работа е разработката на софтуер която използва SNMP протокола за извличане на информация на устройствата в мрежата и на база на резултатите и дефинирани от потребителя групи, да изпълнява действия под формата на скриптове. Друга основна цел на дипломната работа е да се направи анализ на реализираният софтуер.

Задачи

Задачите които сме си поставили в процеса на анализ и разработка са следните:

1. Анализ на нуждите на потребителите с цел предоставяне на необходимите функционалности.
2. Разработка на Single Page Application клиентска част която да притежава следните функционалности:
    a. Добавяне на нови метрики (OID-та) които потребителят иска да проследява.
    b. Дефиниране на групи.
    c. Добавяне на действия (скриптове) които системата да изпълнява, ако прочетената стойност попадне в определена група.
    d. Опция за избиране на интервала от време за което потребителят иска да види резултатите.
    e. Визуализация на резултатите в табличен вид.
    f. Визуално представяне на резултатите, групирани по дефинираните от потребителя групи.
    g. Опция за преглед на изпълнените скриптове и резултатите от тях.
    h. Опция за сваляне на резултатите в CSV файл.
3. Разработка на сървърна част която да притежава следните функционалности:
    a. Добавянето на REST APIs които да бъдат ползвани от клиентската част.
    b. Добавяне на модул за следене на добавените от потребителя SNMP endpoints, взимането на резултати, търсене на група в които резултатите биха попаднали и изпълняване на скриптове на база на намерените групи. Да се използва SNMP polling.
    c. Запис на резултатите от изпълнените скриптове.

Глава 2 - Теоретично решение на поставената задача

Целта на текущата дипломна работа не е самото конфигуриране на SNMP агентите, а създаването на система (SNMP Manager Software) която ги използва (и съответно MIB-те до които те имат достъп) за да може да следи определени от потребителя параметри на определен интервал от време, съхранение на извлечените стойности и при съвпадение между стойност и зададена от потребителя стойност, да изпълнява зададен от потребителя скрипт.

Базово описание на работата на SNMP Manager системата и функционалностите които включва.

1. Потребителят въвежда следните входни данни нужни на системата
    a. Host* – IP или DNS адрес на устройството.
    b. Port* – Портът който ще се използва за SNMP заявките от системата към SNMP агентите.
    c. OID* – Идентификаторът на ресурса който искаме да следим.
    d. Community* - това е стринг подобен на потребителско име или парола който се праща с всеки SNMP Get-Request и позволява (или забранява) достъпа до статистиките на устройството. Ако този стринг е правилен, агента на устройството ще върне информацията, а ако е грешен просто ще игнорира заявката и няма да върне резултат.
2. На определен интервал те биват четени и използвани за извличане на информацията с цел мониторинг.
3. Въз основа на прочетените данни, системата на база на потребителската конфигурация, може да изпълнява скриптове предоставени от потребителя.

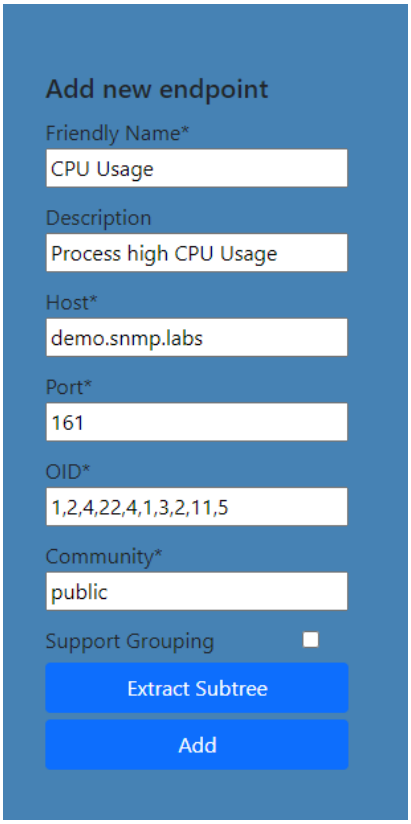
ВАЖНО

За простота по - натам на места ще използвам термина **SNMP endpoint** или само **endpoint** който включва следните данни: **Host**, **Port**, **OID** и **Community**.

Създаване на конфигурация

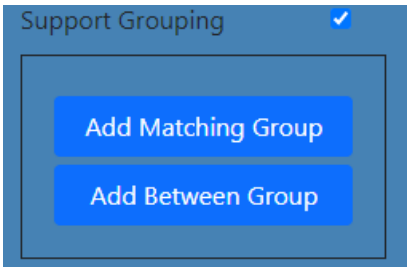
Формата за добавяне на нов endpoint съдържа следните input полета от тип text.

- **Friendly Name*** – Това е името по което потребителя ще може да разпознае кой точно параметър следи. Например: “CPU Temp”.
- **Description** – Това е optional параметър. Функцията му е да може потребителя да опише накратко какво точно представлява този параметър или защо го следим.
- **Host*** – IP или DNS адрес на устройството.
- **Port*** – Портът който ще се използва за SNMP заявките от SNMP агентите.
- **OID*** – Идентификаторът на ресурса който искаме да следим.
- **Community*** - това е стринг подобен на потребителско име или парола който се праща с всеки SNMP Get-Request и позволява (или забранява) достъпа до статистиките на устройството. Ако този стринг е правилен, агента на устройството ще върне информацията, а ако е грешен просто ще игнорира заявката и няма да върне резултат.



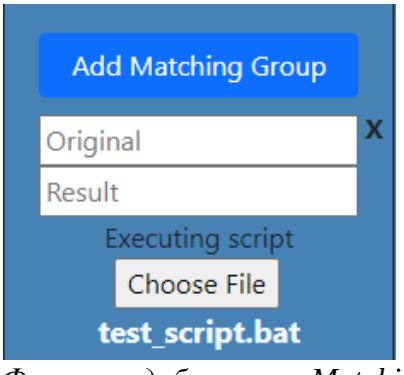
Фиг. 2.1 Форма за добавяне на нов SNMP endpoint

Под самата форма има checkbox наименован “Support Grouping”. Неговата функция е да индикираме, че искаме на базата на стойностите които получаваме за този OID да изпълняваме определен скрипт. При избиране на тази опция се визуализират следните две опции:



Фиг. 2.2 Форма за добавяне на Matching или Between групи

И двете са свързани с добавяне на групи.
Първата опция е наречена “**Matching group**”. При добавяне на нова Matching група потребителят трябва да въведе следната информация:

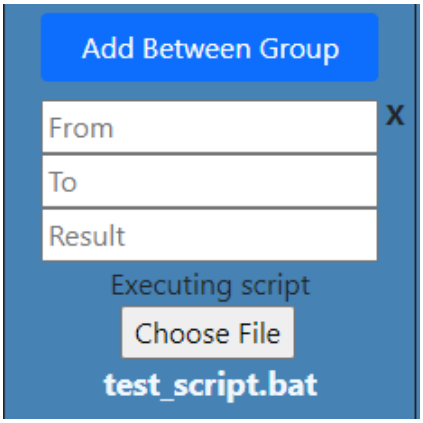


Фиг. 2.3 Форма за добавяне на Matching група

- **Original** – това е стойността за която потребителят иска да следи. Например ако иска да види кога CPU usage-а ще бъде 100%, в Original той може да въведе "**100**".
- **Result** – Това е стойност която той може да използва за придаване на семантика. Например в този случай той може да използва "**Programs not responding**".
- **Executing script (Choose file)** – от тук потребителя може да добави скриптов файл който ще бъде изпълнен ако CPU usage-а е 100%. Например "reboot.bat" или "send_urgent_email.bat" или някакъв друг скрипт който ще извърши желаното действие.

Ползата от този тип група е когато потребителят иска да следи за точно определена стойност.

Втората опция е наречена "**Between group**". При добавяне на нова Between група потребителят трябва да въведе следната информация:



Фиг. 2.4 Форма за добавяне на Between група

- **From** – Това е минималната стойност на която стойноста трябва да отговаря за да можем да идентифицираме, че тя попада в тази група. Например "**0**".
- **To** - Това е максималната стойност на която стойноста трябва да отговаря за да можем да идентифицираме, че тя попада в тази група. Например "**30**".
- **Result** – Това е стойност която потребителят може да използва за придаване на семантика. Например ако използваме същият пример с CPU usage, тогава той може да използва "**Low**" което за него да значи, че между 0% и 30%, натоварването е слабо.
- **Executing script (Choose file)** – от тук потребителя може да добави скриптов файл който ще бъде изпълнен ако CPU usage-а е между 0% и 30%. Например в този случай може да не искаме да извършваме никакво действие и не е нужно да добавяме никакъв скрипт, но ако групата беше за между 60% и 80% можеше да искаме да ползваме скрипт който ни праща email, че натоварването се покачва.

Ползата от този тип група е когато потребителят иска да следи за range от стойности и при всяка от тях иска да изпълни едно и също действие.

Потребителят има възможност да добави множество Matching и Between групи.

ВАЖНО

1. Matching групите са с предимство пред Between групите. Това значи, че ако моментната стойност попадне едновременно в matching и between група, ще се изпълни действието на matching групата заради презумпцията, че тя е по-конкретна и изисква пълно съвпадение.

2. Поредността има значение. Ако моментната стойност попадне едновременно в множество Between, ще изпълним действието на първата която сме дефинирали заради презумпцията, че те са дефинирани по ред на важност. Тоест когато потребителят въвежда групите е важно първо да дефинира по-конкретните, и после по-общите. Например следната конфигурация не е подходяща

10 – 90
30 – 60
70 – 90

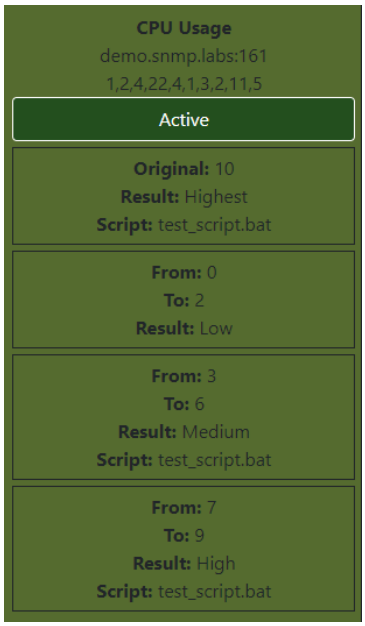
Правилното дефиниране би било

30 – 60
70 – 90
10 – 90

По този начин първо ще хванем по-конкретните групи, и ако стойността не попадне в тях, ще търсим в по-общата.

Преглед на конфигурация

След добавяне на endpoint, потребителят може да го види в левият страничен панел на системата.



Фиг. 2.5 Изглед на добавен SNMP endpoint

Показваме детайлите както за самия endpoint, така и за всички групи и асоциираните с тях скриптове. Друг важен детайл което е видим за потребителя е статуса на този endpoint в нашата система. Възможните статуси са три:

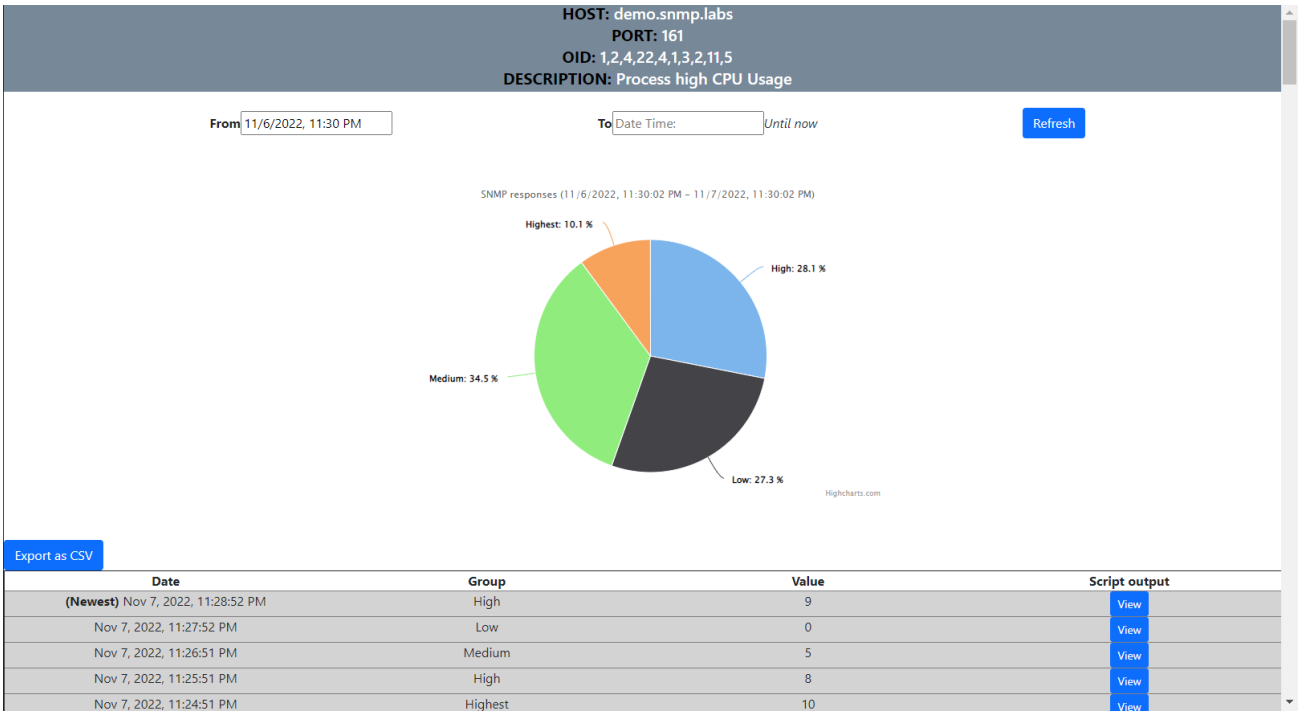
- **Active** – Когато е в този статус, системата ще събира и обработва информацията за този endpoint.
- **Deactivated** – Когато е в този статус, потребителят ще може да види този endpoint в системата, но системата няма да извършва никакви действия свързани с него. Този статус е удобен когато искаме временно да спрем следенето на дадем endpoint и не искаме да събираме потенциално грешни данни (пример: профилактика).
  Потребителят **МОЖЕ** по всяко време да върне статуса на **Active**.
- **Deleted** – Когато endpoint-а е в този статус, потребителят може да види този endpoint в системата но **НЕ МОЖЕ** да го направи **Active** отново. Единственото което може да прави потребителят е да гледа историческите данни за този endpoint.

ВАЖНО

Редактирането на вече добавен endpoint не е имплементирано защото би могло да промени смисълът на данните които сме събрали до текущият момент. Това което потребителят може да направи е да смени статуса на вече съществуващ и да направи нов запис, и така за старият запис ще има правилният контекст когато преглежда събраните резултати.

Преглед на резултатите

След добавянето на желаните endpoint-и, системата започва на определен интервал от време да събира данни за всеки от тях, и на база на тези резултати изпълнява или не изпълнява добавените скриптове към съответните групи. Ето как изглежда основния изглед за един endpoint.



Фиг. 2.6 Изглед на главния екран за избран SNMP endpoint

Най – отгоре можем да видим детайли за самият endpoint. Под тях можем да видим:

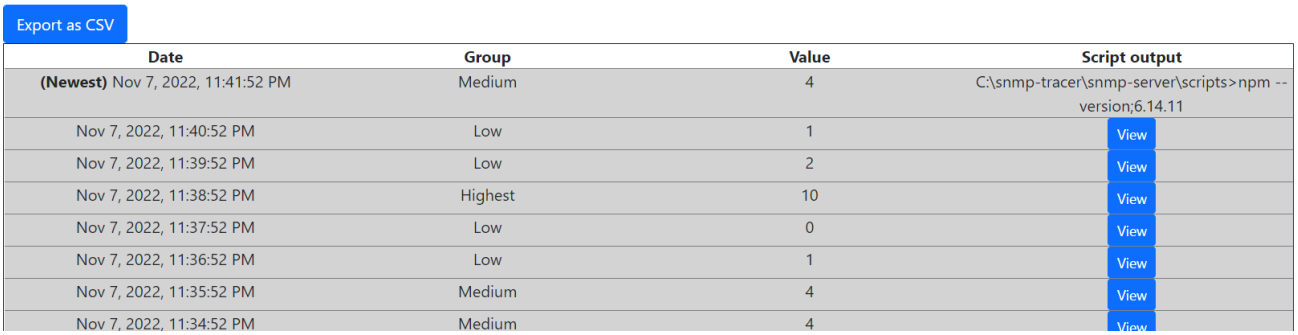
- Избор на начална и крайна дата и час. На база на този избор потребителят може да види резултатите само за определен интервал от време.
- Графика с разбивка на резултатите. Разбивката е по имената на групите, като всички резултати които не попадат в определена група се агрегират в една обща група.
- Таблица с резултатите. В нея потребителят може да види:
    - Дата и час на резултата.
    - Име на група в която той попада или респективно нищо ако не попада в никоя група.
    - Стойността която е била прочетена към конкретният момент.
    - Резултата от скрипт-а ако стойността попада в някоя група и към нея има прикачен скрипт.

  Резултатите са подредени по хронологичен ред (от най-нови към най-стари)
- Бутон за сваляне на съдържанието на таблицата в CSV формат. В сваленият файл винаги има само информацията която е визуализирана в таблица под него.

ВАЖНО

1. По подразбиране няма зададена крайна дата и час. Това значи, че за крайна дата и час ще се използва текущият момент.
2. Потребителят може да селектира всяка една група от графиката и това ще презареди данните в таблицата отдолу само с тези резултати.

Ето как изглежда резултата от скрипта в последната колона.



Фиг. 2.7 Изглед на резултатите за SNMP endpoint в табличен вид

В конкретният случай скрипта показва версията на NPM. Решението да не се показват всички резултати по подразбиране е взето защото:

- Може резултата да е дълъг и ще покажем прекалено много информация на потребителя първоначално което може да е объркващо и не е добро UX решение.
- От performance гледна точка не е добре, да извличаме всички тези детайли винаги. Със сегашната имплементация при нужда зареждаме резултата което е оптимизация.

Глава 3 - Описание на използваната апаратна (схемна) и/или софтуерна част

За имплементация на системата съм ползвал Angular 13.3.0 и Node 12.21.0. Нека разгледаме в детайли.

UI (Client)

Защо Angular?

- Angular е framework който е широко разпространен и е достатъчно доказал се, с огромно количество допълнителни библиотеки.
- Single Page Application. Има вграден two way binding.
- Библиотеката която ползвам за графиките има версия и за Angular.
- Разработен е от Google и съответно има дългосрочен съпорт.
- Typescript базиран е.
- MVC базиран framework.
- Има модулна структура и разделението на отделните feature-и е лесно да бъде разделено. Позволява lazy loading което води до оптимизация на време и на размер на крайните файлове.
- Тестването на отделните части от кода е улеснено поради самата структура на framework-а и поради това, че има много библиотеки които могат да бъдат ползвани. Освен това има и много вградени помощни средства които идват от самият framework.

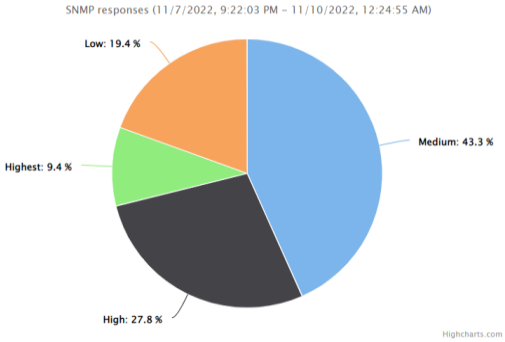
Използвани библиотеки

- **Bootstrap (версия 5.1.3)** – Bootstrap е Sass библиотека която е широко използвана за стилизация на HTML. Тя съдържа стилове за изглед на бутони, таблици, подредба на елементите и други. Тя е ползвана за визуализацията на: бутоните, формата за добавяне на нов endpoint, таблицата с резултатите и др. След добавянето и като dependency трябваше да се добави и в глобалният styles.css на default-ният Angular проект.

```
snmp-client > src > styles.css
1  /* You can add global styles to this file, and also import other style files */
2
3  @import "~ng-pick-datetime/assets/style/picker.min.css";
4  @import "~bootstrap/dist/css/bootstrap.css";
5
```

Фиг. 3.1 Добавяне на bootstrap.css към проекта.

- **Highcharts (версия 10.0.0)** – Highcharts е библиотека за създаване и визуализация на графики. Тя е ползвана за създаването на графиката която визуализира разбивката на резултатите по групи.

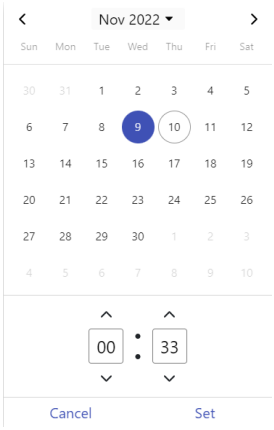


Фиг. 3.2 Графика изобразяваща разпределението на резултатите по групи

- **ng-pick-datetime (версия 7.0.0)** – Това е библиотека която предоставя контроли за избор на дата и час. Тя се използва за контролите от които потребителят избира начална и крайна дата на резултатите които са събрани за даденият endpoint.

Фиг. 3.3 Контроли за избор на начална и крайна дата

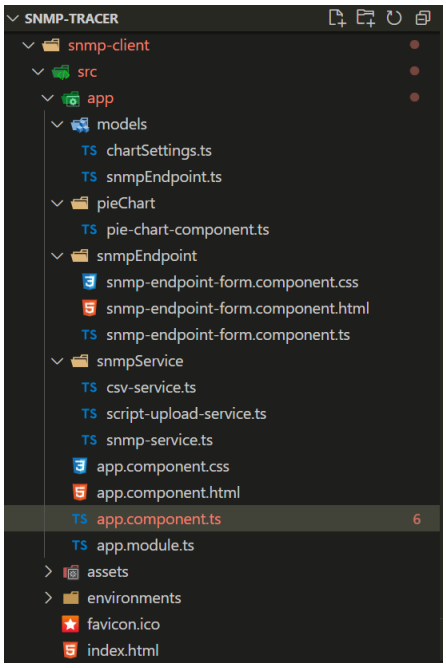
Потребителят има възможност да избере година, месец, ден, час и минути.



Фиг. 3.4 Контрола за избор на дата, час и минути

Комуникацията със сървърната част се осъществява посредством HTTP протокола. За тази цел е ползван вграденият в Angular клиент наречен HttpClient който има функции за изпращане на HTTP заявки. Всички функции за заявки към сървъра се намират в service наречен SNMPService който вътрешно използва HttpClient-а. По – нататък ще разгледаме списък от всички REST API-та които сървърната част предоставя.

За по – добро разделение на отговорностите, по – четим код, по – лесно тестване съм добавил още два service-а: един който съдържа логиката за качване на файл (това са файловете които до момента наричаме скриптове, асоциирани със групи на съответен endpoint) и един който съдържа логиката за генериране и сваляне на резултатите във CSV файл.



Фиг. 3.5 Структура на файловете (UI)

ВАЖНО

1. За да няма колизии на имената на файловете които потребителят асоциира с групите, системата добавя уникалния идентификатор на endpoint-а и на групата към името на файла и така си гарантираме, че името винаги ще е уникално. Когато обаче ги показваме в браузъра, ние скриваме тази допълнителна информация защото тя не носи ползи на потребителя.

2. Когато сваляме събраните данни в CSV файл, файла съдържа следните колони: Дата, Група и Стойност. Не е случайно, че не добавяме резултатите на скриптовете в сваленият файл. Причините са няколко:

- Прекалено много допълнителна информация която може да е объркваща за потребителя при голямо количество данни.
- Резултата може да съдържа всякакво съдържание.
- На база на датата потребителят може да провери за изпълнение на скрипт в системата.
- Тези данни са достатъчни за извличане на статистики на база на дата и резултат.

Backend (Server)

Защо Node?

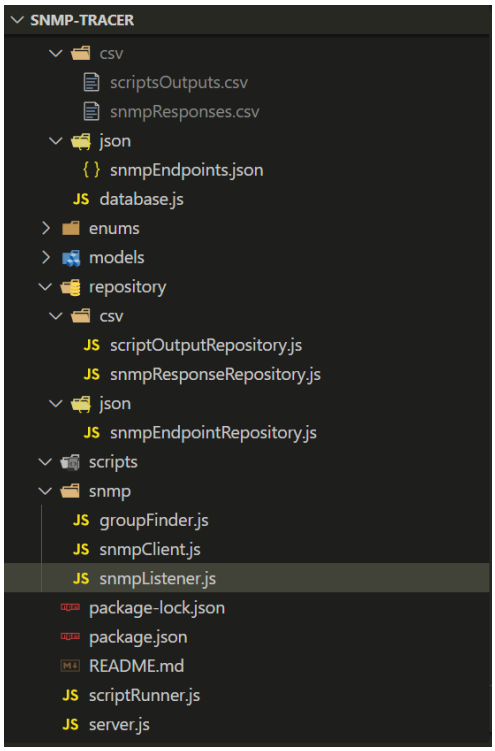
- Node е широко разпространен и е достатъчно доказал се, с огромно количество допълнителни библиотеки.
- Не изисква платени IDE-та за разработка.
- NPM (Node Package Manager).
- Single-Threaded Event Loop Architecture. Когато приложението се стартира се инициализира event loop който изпълнява инструкция след инструкция. Това елиминира нуждата от мениджмънт на множество нишки.
- Подходящ е за real-time приложения.
- Native Support в AWS.
- Javascript базиран. (Позволява и ползването на Typescript).
- Удобен за работа с нерелационни бази данни.

Използвани библиотеки

- **Express (версия 4.16.3)** – Web Framework. Ползвал съм го за вдигането на Web Server с необходимите RESTful APIs които се ползват от клиентската част.
- **Fast-csv (версия 2.4.1)** – Това е библиотека която предоставя функции за работа със CSV файлове.
- **Multer (версия 1.4.5-lts.1)** – Библиотека която ползвам за upload-а на скриптове.
- **Snmp-native (версия 1.1.2)** – Една от основните библиотеки в текущата система. Тя се използва за изпращане на SNMP заявки и получаването и обработването на резултатите до получаването им във формат удобен за работа.

Нека разгледаме в детайли имплементацията на сървърната част.
Входната точка на node приложението е server.js. В този файл се намира следната логика:

- Инициализация на express.js. Дефиниране на всички REST API-та.
- Инициализация на database фасада.
- Стартиране на SNMP Listener който е отговорен за извличане на данни посредством SNMP заявки за всички добавени и активни endpoints добавени от потребителя. Детайлна разбивка и обяснение как работи SNMP Listener-а ще направим малко по-късно.



Фиг. 3.6 Структура на файловете (Backend)

REST APIs

Както споменахме по – рано за HTTP сървър се използва Express библиотеката. За улеснение на разработката на дипломната работа CORS политиките са настроени да не спират request-и.

```
app.use(function (req, res, next) {
   res.header("Access-Control-Allow-Origin",  "*");
   res.header(
      "Access-Control-Allow-Headers",
      "Origin, X-Requested-With, Content-Type, Accept");
   next();
});
```

Фиг. 3.7 CORS конфигурация

Нека разгледаме списък от имплементираните REST API endpoints:

HTTP метод	**REST API**	**Описание**
GET	/snmpEndpoints	връща списък от всички добавени SNMP endpoints.
POST	/snmpEndpoints	добавя нов SNMP endpoint.
POST	/snmpEndpoint/responses	връща списък от резултатите на конкретен SNMP endpoint за определен интервал от време.
POST	/snmpEndpoint/setStatus	сменя статуса на конкретен SNMP endpoint.
POST	/snmpEndpoint/updateGroupScript	обновява асоциираните скриптовете за списък от групи.
POST	/snmpEndpoints/test	връща subtree по зададен OID. Този endpoint се използва от “Extract Subtree” бутона когато е част от формата за добавяне на нов SNMP endpoint. Идеята е да дадем на потребителят начин да види какъв е типа на резултата който стои зад това OID.
POST	/scriptsOutputs	връща резултата от изпълнен скрипт.
POST	/snmpEndpoint/upload	записва скрипт подаден от потребителя за определен endpoint и група.

Фиг. 3.8 REST APIs - описание

Database

Нека разгледаме следващата точка – database фасадата и къде точно съхраняваме данните. Нека започнем с това, че тази фасада се намира в database.js. Защо я наричам фасада? Защото изпълнява тази роля (това е така наречения **Facade design pattern**) и неговата роля е да скрие както комплексна логика зад опростен интерфейс който да е по-лесен за ползване, така и да позволи разделението на вътрешната логика в отделни обекти като това остава скрито за ползватора на този обект. Нашият случай е точно такъв. За съхранение на данните една от опциите беше да се използва релационна база данни, но взех решението, че това ще усложни имплементацията и нямаме нужда в момента от това. Затова взех решението да ползвам файлове за съхранение на данните като това остава скрито зад поредният design pattern – **Repository design pattern**. Неговата роля е да скрие цялата логика за запис/четене на данни от конкретен носител зад определен интерфейс. Това позволява лесно да можем да подменим носителя стига да сме спазили публичния интерфейс на repository class-a. В нашият случай имаме следните 3 repository class-a:

- **scriptOutputRepository** – Зад този интерфейс се съдържа логиката за запис и четене на output-а от изпълнените скриптове за определените групи за определените endpoints.
- **snmpResponseRepository** – Зад този интерфейс се съдържа логиката за запис и четене на резултатите извлечени на определен период от време, за съхранените от потребителя endpoints.
- **snmpEndpointRepository** – Зад този интерфейс се съдържа логиката за:
    - Взимане на всички добавени endpoints
    - Добавяне на нов endpoint
    - Смяна на статуса на конкретен endpoint
    - Асоцииране на скрипт към група (between или match)

В момента в database.js ползваме следните имплементации

```
const snmpEndpointRepository = require("../repository/json/snmpEndpointRepository");
const snmpResponseRepository = require("../repository/csv/snmpResponseRepository");
const scriptOutputRepository = require("../repository/csv/scriptOutputRepository");
```

Фиг. 3.9 Списък от repository инстанции

но ако решим да ползваме релационна база данни или например MongoDB можем да ги заменим тук и ще почнем да четем и пишем на новото място. По принцип за подобен тип обекти като database се използва и термина Unit Of Work но той има някои характеристики които сегашната имплементация няма като например осигуряване на транзакция (тоест всичко да мине като атомарна операция и ако има проблем да rollback-не всички направени промени), затова използвам и термина фасада защото по-добре описва функциите на сегашната имплементация.

Нека разгледаме отделните entity-та които имаме в нашата система, каква информация точно пазим за тях и в какъв формат.

1. **SNMP endpoint**
    a. **Repository файл**: snmpEndpointRepository.js
    b. **Къде пазим данните и в какъв формат**: JSON файл с име snmpEndpoints.json. Пазим следната информация за всеки endpoint:

```
{
  "id": number,
  "friendlyName": string,
  "description": string | null,
  "oid": number[],
  "host": string,
  "port": number,
```

```
  "community": string,
  "status": {
    "id": number,
    "name": string
  },
  "supportGrouping": boolean,
  "groupingMatch": [
    {
      "original": string,
      "result": string,
      "id": number,
      "script": string | null, // null ако няма прикачен скрипт
    }
  ],
  "groupingBetween": [
    {
      "from": number,
      "to": number,
      "result": string,
      "id": number,
      "script": string | null, // null ако няма прикачен скрипт
    }
  ]
}
```

Фиг. 3.10 SNMP endpoint – формат

2. **SNMP response**
    a. **Repository файл**: snmpResponseRepository.js
    b. **Къде пазим данните и в какъв формат**: CSV файл с име snmpResponses.csv. Пазим следната информация за всеки резултат:

```
// 16 символен стринг. Пример: ffa3cc1e1ac6cdbd15c80e6711e68e65
id: string;
endpointId: number;
// SNMP OID. Пример: "1,2,4,22,4,1,3,2,11,5"
oid: string;
host: string;
port: number;
community: string;
// Стойността която сме взели за конкретното OID
value: string;
groupId: number;
// Result полето от групата. Пример: Medium
group: string;
// Времето на взимане на резултата в милисекунди. Пример: 1668251321555
dateticks: number;
```

Фиг. 3.11 SNMP response – формат

snmp-server > database > csv > snmpResponses.csv

```
id,endpointId,oid,host,port,community,value,groupId,group,dateticks
ffa3cc1e1ac6cdbd15c80e6711e68e65,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,4,3,Medium,1668250841492
cf9daa17e0d17f22c116de33e97e9399,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,1,2,Low,1668250901495
af0de1678d70f61e421d29164e97f8ac,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,8,4,High,1668250961504
3b9135d1447d398e1b3e81826fa08ced,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,8,4,High,1668251021515
3961edd9ea2ae2c067cd1a3929336c86,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,10,1,Highest,1668251081523
0d4d98dcb517cce240ee82e211267001,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,9,4,High,1668251141535
a6643502c4f9e6a6de694debf8efc16a,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,10,1,Highest,1668251201548
ea4b8ca66fd41283ff890588b0c2b02b,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,7,4,High,1668251261550
6ec05002efbd498cf07eaa9f5ea36e9c,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,3,3,Medium,1668251321555
808becd4d2477b137bc5050d4f2fdc2c,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,0,2,Low,1668251381568
bdd499adc1029738fbcc13f102856fe1,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,2,2,Low,1668251441579
f89be5de735e626dd9ff9c9f9962475d,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,1,2,Low,1668251501592
b4947f5353f7e9d3deeafac4f05634cf,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,7,4,High,1668251561601
ba4e3ba66d05024e73b26c8296d8d9d9,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,3,3,Medium,1668251621604
d37a2a3d82dbb28afbbf209d04594a19,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,4,3,Medium,1668251681608
53e7d785d26efeb86534ed15ab60f97d,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,3,3,Medium,1668251741609
8595afff62c130acb06bf2393382ee93,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,0,2,Low,1668251801618
70772e2cc2da91c15b4903187d3dcb0a,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,7,4,High,1668251861622
1a16c4a0b89ff7dca815ba8a85a0a509,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,4,3,Medium,1668251921636
9adc824c6b0cf10f4ed5ba818ec10c38,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,2,2,Low,1668251981644
092237f60a01ac9f86a9831308ba9d96,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,1,2,Low,1668252041645
f13fa05b8057d695b391426406f27367,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,5,3,Medium,1668252101654
1bab82255e25770795b480e53b811867,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,2,2,Low,1668252161664
40e63f707920b8911cfda177b94979cd,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,6,3,Medium,1668252221665
a5f3a902b3c1ad991778e0f1c5cf06ce,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,8,4,High,1668252281675
e47e889c1bff57c73b5f2b7c92e15811,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,5,3,Medium,1668252341682
```

Фиг. 3.12 SNMP response – данни

3. **Script output**
   a. **Repository файл**: scriptOutputRepository.js
   b. **Къде пазим данните и в какъв формат**: CSV файл с име scriptsOutputs.csv. Пазим следната информация за всеки резултат:

```
// 16 символен стринг. Пример: ffa3cc1e1ac6cdbd15c80e6711e68e65
snmpResponseId: string;
// Output-а от скрипта.
// Пример: C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11
text: string;
endpointId: string;
groupId: number;
// Името на скрипт файла. Пример: 2_3_test_script.bat
script: string;
// Времето на взимане на резултата в милисекунди. Пример: 1668251321555
dateticks: number;
```

Фиг. 3.13 Script output – формат

snmp-server > database > csv > scriptsOutputs.csv

```
snmpResponseId,text,endpointId,groupId,script,dateticks
ffa3cc1e1ac6cdbd15c80e6711e68e65,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
af0de1678d70f61e421d29164e97f8ac,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
3b9135d1447d398e1b3e81826fa08ced,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
3961edd9ea2ae2c067cd1a3929336c86,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
0d4d98dcb517cce240ee82e211267001,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
a6643502c4f9e6a6de694debf8efc16a,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
ea4b8ca66fd41283ff890588b0c2b02b,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
6ec05002efbd498cf07eaa9f5ea36e9c,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
b4947f5353f7e9d3deeafac4f05634cf,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
ba4e3ba66d05024e73b26c8296d8d9d9,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
d37a2a3d82dbb28afbbf209d04594a19,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
53e7d785d26efeb86534ed15ab60f97d,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
70772e2cc2da91c15b4903187d3dcb0a,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
1a16c4a0b89ff7dca815ba8a85a0a509,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
f13fa05b8057d695b391426406f27367,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
40e63f707920b8911cfda177b94979cd,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
a5f3a902b3c1ad991778e0f1c5cf06ce,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
e47e889c1bff57c73b5f2b7c92e15811,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
65609c339c82f177af2fc3fdd6eb22fd,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
80bee4a49444bff1dc49abcafd84f709,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
0cb553de651b9e15a6f1ce450c3d92b6,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
2dd71dc19872c9c85ec8254eed868749,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
f9c6d7c45ce874b2b511ad75e08ddb65,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
ff9a0b7a5db6d8b97f68be09240ac773,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
e1ebcb339ba7eb548a201ea674af7319,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
0f7703a23afe685d83ad3d25d4d070aa,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
```

Фиг. 3.14 Script output – данни

SNMP Listener

В snmpListener.js се съдържа core логиката на самата система свързана със изпращането на SNMP заявки, записа на резултатите и стартирането на скриптове. Нека първо разгледаме кода в самият файл и после детайлно ще разгледаме всички помощни функции които се ползват.

```
const snmpClient = require("./snmpClient");
const snmpEndpointRepository = require("../repository/json/snmpEndpointRepository");
const snmpResponseRepository = require("../repository/csv/snmpResponseRepository");
const scriptRunner = require("../scriptRunner");
const scriptOutputRepository = require("../repository/csv/scriptOutputRepository");
const SNMPResponse = require("../models/snmpResponse");
const ScriptOutput = require("../models/scriptOutput");
const endpointStatus = require("../enums/endpoint-status");
const groupFinder = require("./groupFinder");
const snmp = new snmpClient();
const endpoints = new snmpEndpointRepository();
const responses = new snmpResponseRepository();
const scripts = new scriptRunner();
const scriptsOutputs = new scriptOutputRepository();

function start() {
    visitNodes();
}
function visitNodes() {
    visitEachNode();
    setTimeout(visitNodes, seconds(60));
}
function seconds(sec) {
    return sec * 1000;
}
function visitEachNode() {
    endpoints.read().then(endpoints => {
        const activeEndpoints = endpoints.filter((node) => node.status.id ===
endpointStatus.Active);

        activeEndpoints.forEach((node) => {
            snmp.extractSubtree(node).then(varbinds => {
                if (varbinds) {
                    varbinds.forEach(bind => {
                        const group = groupFinder.find(node, bind.value);
                        const snmpResponse = SNMPResponse(
                            node.id,
                            node.oid,
                            node.host,
                            node.port,
                            node.community,
                            bind.value,
                            group.id,
                            group.value
                        );
```

```
                        const snmpResponseId = responses.write(snmpResponse);

                        if (group.script) {
                            scripts.run(group.script).then((output) => {
                                const scriptOutput = ScriptOutput(
                                    snmpResponseId,
                                    node.id,
                                    group.id,
                                    group.script,
                                    output
                                );
                                scriptsOutputs.write(scriptOutput);
                            });
                        }
                    });
                }
            });
        })
    });
}

const snmpListener = function () {
    return {
        start: start,
        endpointData: snmp.extractSubtree
    }
};

module.exports = snmpListener;
```

Фиг. 3.15 SnmpListener – имплементация

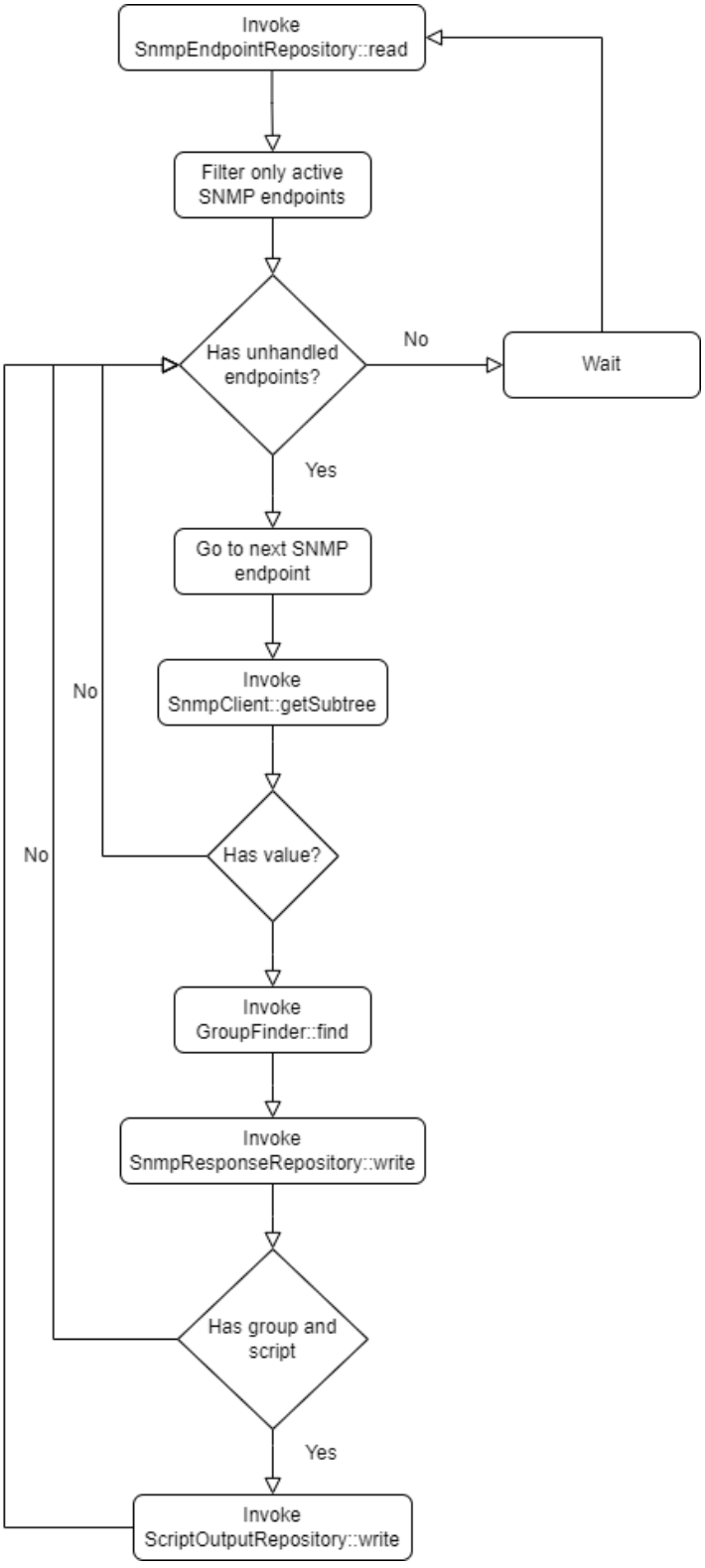
Имаме 2 публични функции “start” и “endpointData”. Функцията “endpointData” от своя страна извиква SnmpClient. (който ще разгледаме малко по-късно) и по - конкретно неговата единствена функция “extractSubtree”. Функцията “start” от своя страна при нейното извикване вика функцията “visitNodes”. Тя започва да се извиква рекурсивно на определен интервал от време като при всяко нейно извикване се вика и функцията “visitEachNode”. Това което тя прави е следното:

1. Зарежда всички запазени от потребителя endpoint-и посредством snmpEndpointRepository.
2. Филтрира и премахва всички които не са активни.
3. За всеки от тези активни endpoint-и
    a. Ползвайки “extractSubtree” функцията от snmpClient – а взима стойността на OID – то ползвайки нужната информация от endpoint-а.
    b. Проверява дали има стойност
        i. При наличие на стойност
            1. Търси група в която тази стойност попада.
            2. Записва всички детайли за стойността посредством snmpResponseRepository, като записва и наличието или липса на група.
            3. Проверява дали има група и дали има скрипт за изпълнение
                a. При наличие на група и скрипт
                    i. Изпълнява скрипта и записва резултатите от

изпълнение на скрипта посредством scriptOutputRepository.

b. При липса на група или скрипт
   i. Не извършва никакво действие

ii. При липса на стойност
   1. Не извършва никакво действие



Фиг. 3.16 SnmpListener workflow

SNMP Client
В snmpClient.js се намира имплементацията на функцията “extractSubtree”.

```
const snmp = require("snmp-native");

function extractSubtree(node) {
    const session = new snmp.Session({
        host: node.host, port: node.port, community: node.community
    });

    return new Promise((resolve) => {
        session.getSubtree({
            oid: node.oid
        }, function (error, varbinds) {
            if (error) {
                console.log("Fail :(", error);
            } else {
                resolve(varbinds);
            }
        });
    })
}

const snmpClient = function () {
    return {
        extractSubtree: extractSubtree
    }
};

module.exports = snmpClient;
```

Фиг. 3.17 SnmpClient – имплементация

Тя използва snmp-native библиотеката като първо се инициализира нова сесия като се подават host, port и community. След като сесията е готова ползваме функцията “getSubtree” която приема OID като параметър и връща обект който съдържа стойността.

Group Finder

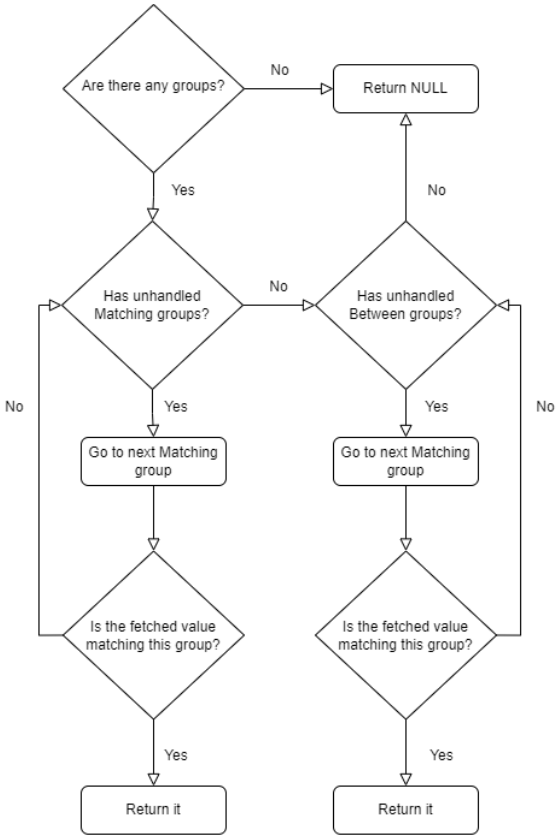
В groupFinder.js се намира имплементацията на функцията “find” която търси група по зададена стойност.

```
function find(node, value) {
    let group;
    if (node.supportGrouping) {
        if (!group) {
            group = searchInMatchingGroups(node.groupingMatch, value);
        }
        if (!group) {
            group = searchInBetweenGroups(node.groupingBetween, value);
        }
    }
    if (!group) {
        group = emptyGroup();
    }
    return group
}

function searchInBetweenGroups(groupingBetween, value) {
    if (groupingBetween.length > 0) {
        for (let i = 0; i < groupingBetween.length; i++) {
            const from = Number(groupingBetween[i].from);
            const to = Number(groupingBetween[i].to);
            if (from <= Number(value) && Number(value) <= to) {
                return {
                    id: groupingBetween[i].id,
                    value: groupingBetween[i].result,
                    script: groupingBetween[i].script,
                }
            }
        }
    }
    return null;
}

function searchInMatchingGroups(groupingMatch, value) {
    if (groupingMatch.length > 0) {
        for (let i = 0; i < groupingMatch.length; i++) {
            if (groupingMatch[i].original == value) {
                return {
                    id: groupingMatch[i].id,
                    value: groupingMatch[i].result,
                    script: groupingMatch[i].script,
                }
            }
        }
    }
    return null;
}
```

```
function emptyGroup() {
    return {
        id: null,
        value: "Unknown"
    };
}

module.exports = {
    find: find
};
```

Фиг. 3.18 GroupFinder – имплементация

1. Първо се прави проверка дали има създадени групи за конкретният endpoint.
    a. Ако има групи
        i. Проверява се дали има Matching група която има стойност равна на стойността която сме получили.
            1. Ако има такава група
                a. Връщаме я
            2. Ако няма такава група
                a. Проверяваме дали има Between група за която получената стойност попада в нейният интервал.
                    i. Ако има такава група
                        1. Връщаме я
                    ii. Ако няма такава група
                        1. Връщаме, че няма намерена група
    b. Ако няма групи
        i. Връщаме, че няма намерена група



Фиг. 3.19 GroupFinder workflow

Script Runner

В scriptRunner.js се намира имплементацията на функцията “run” която изпълнява скрипт и връща резултата от изпълнението му.

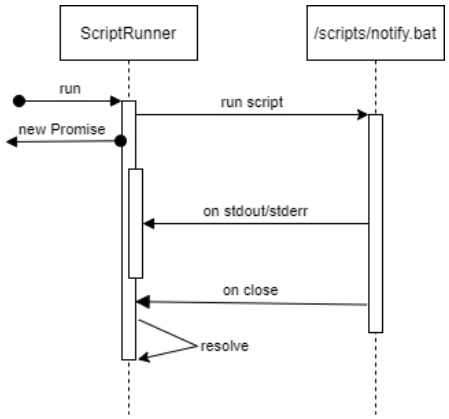
```
const { spawn } = require("child_process");

async function run(script){
    return await new Promise(resolve => {
        let output=[];
        let options={shell:true, cwd: "./scripts/"};
        let child=spawn(script, options);
        child.stdout.on(
           "data",
           txt => output.push(txt.toString().replace(/(\r\n|\n|\r)/gm, "").trim()));
        child.stderr.on(
           "data",
           txt => output.push(txt));
        child.on(
           "close",
           () => resolve(output.filter(chunk => chunk.length).join(";")));
    });
}

module.exports = function () {
    return {
        run: run
    }
};
```

Фиг. 3.20 ScriptRunner – имплементация

При подадено име на скрипт, функцията го търси в парката “/scripts”, като събира всички output-и от изпълнението на скрипта и накрая връща един общ стринг от конкатинираните отделни output-и като използва “;” за разделител. Нещо много важно което трябва да се спомене е, че изпълнението на скриптовете не блокират главната нишка. Функцията стартира скрипт и слуша за определени събития от неговото изпълнение

- output от скрипта
- грешка при изпълнение на скрипта
- край на скрипта

При нотификация за приключване на скрипта, връщаме събраната информация и я записваме.



Фиг.3.21 ScriptRunner workflow

Глава 4 - Изчислителна част

В процеса на анализ и разработка на дипломната работа преминах през няколко стадия на тестване и допълнителни разработки.

Стъпка 1 (Online)

Първоначално използвах **online SNMP агент с MIB** за да мога да тествам, че всички модули са интегрирани по правилният начин. Това което постигнах като резултат беше следното:

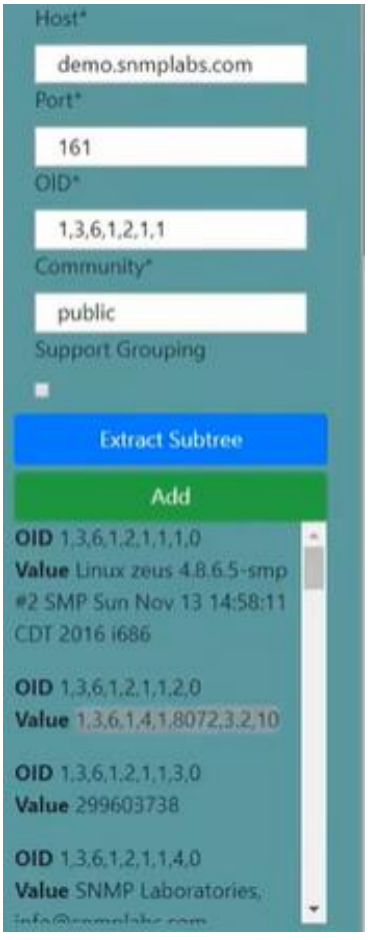
- Верификация, че SnmpClient::extractSubtree изпълнява своите функции и за подаден Host, Port, OID и Community връща правилните данни.
- Верификация, че успяваме да запишем върнатите данни.
- Верификации, че правилно намираме група на база на получен резултат.

Използвани входни данни:

- **HOST** – damo.snmplabs.com
- **PORT** – 161
- **COMMUNITY**: public

Тест 1: Extract Subtree

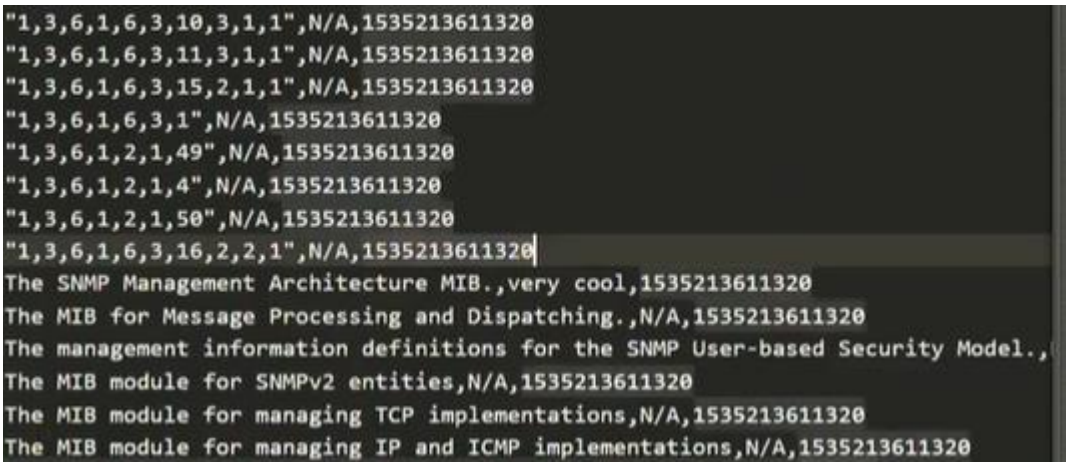
За тестване на extract subtree функционалността ползвах **OID = 1,3,6,1,2,1,1**. За целта използваме “Extract Subtree” бутона който изпраща HTTP заявка със съответните входни данни.



Фиг. 4.1 Extract Subtree - множество резултати (версия 1 с Online MIB)

Както се вижда от резултатите (Фиг 4.1), по зададеното OID, получаваме списък от всички OID-та под него в MIB дървото, заедно с техните стойности. В следващите версии на текущата система, показваме допълнителни Name и Description за всяко OID.

Тест 2: Записване на резултатите и намиране на група
Нека разгледаме част от данните които сме прочели и записали.



Фиг. 4.2 Записани резултати (версия 1 с Online MIB)

Тези които не попадат в никоя група, за група има записано **N/A**. Само за един резултат има записано **very cool** защото имаме Matching група която има original value което е "**The SNMP Management Architecture MIB.**".

Фиг. 4.3 Matching група (версия 1 с Online MIB)

Към конкретният момент формата за добавяне на скриптове, и тяхното изпълнение още не бяха имплементирани. Също така беше трудно да бъде тествано графичното представяне на данните защото стойностите не се променяха. Това наложи нуждата от последващите промени.

Стъпка 2 (Mock)
След като се появиха тези лимитации, следващата стъпка която направих беше да ползвам локален MIB с mock данни, като за нуждите на проекта направих резултатите да са на случаен принцип в определен интервал. Стъпките които предприех бяха следните:

Сваляне на MIB
Свалих MIB в JSON формат от платформа която предоставя MIB-ве на различни устройства (можете да намерите линк в Използвана Литература) . Файлът се казва **HOST-RESOURCES-MIB.json**. Избрах да е в **JSON** формат заради по - лесната обработка в Node. Другите опции бяха: **MIB**, **CSV** и **YAML**. Ето как изглежда част от съдържанието на файла:

```
{
  … // Before
  "hrSystemMaxProcesses": {
    "name": "hrSystemMaxProcesses",
    "oid": "1.3.6.1.2.1.25.1.7",
    "nodetype": "scalar",
    "class": "objecttype",
    "syntax": {
      "type": "Integer32",
```

```
      "class": "type",
      "constraints": {
        "range": [
          {
            "min": 0,
            "max": 2147483647
          }
        ]
      }
    },
    "maxaccess": "read-only",
    "status": "current",
    "description": "The maximum number of process contexts this system can support.
If there is no fixed maximum, the value should be zero. On systems that have a fixed
maximum, this object can help diagnose failures that occur when this maximum is
reached."
  },
  "hrStorageTypes": {
    "name": "hrStorageTypes",
    "oid": "1.3.6.1.2.1.25.2.1",
    "class": "objectidentity"
  },
  "hrMemorySize": {
    "name": "hrMemorySize",
    "oid": "1.3.6.1.2.1.25.2.2",
    "nodetype": "scalar",
    "class": "objecttype",
    "syntax": {
      "type": "KBytes",
      "class": "type"
    },
    "units": "KBytes",
    "maxaccess": "read-only",
    "status": "current",
    "description": "The amount of physical read-write main memory, typically RAM,
contained by the host."
  },
  … // After
}
```


Фиг. 4.4 HOST-RESOURCES-MIB.json съдържание (версия 2 с Local MIB)

Наличието на този MIB файл ми предостави списък от записи със следната информация която беше нужна за целите на тестване на системата:

- OID
- Name
- Type
- Description

Mock на данните

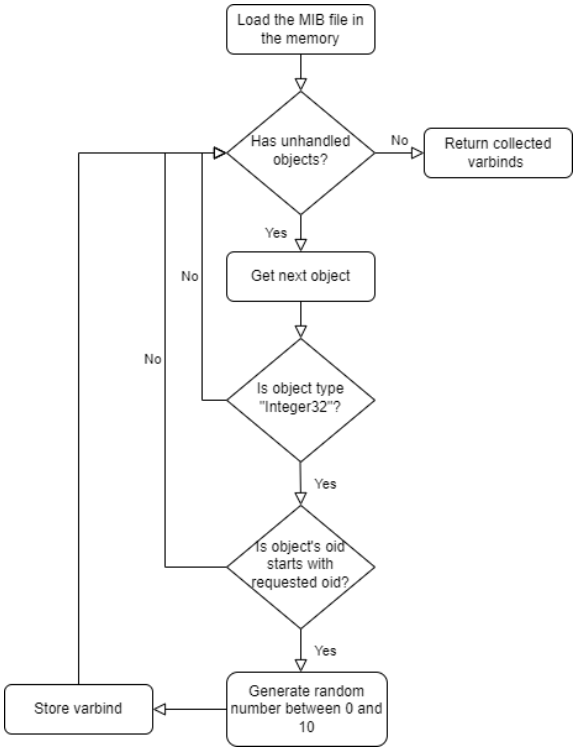
Като втора стъпка трябваше да добавя функция която да връща като резултат новите данни вместо текущата имплементация с SNMP заявки.

За целта добавих нова функция която да подмени вътрешната имплементация на SnmpClient:: extractSubtree която да има същият публичен интерфейс, и така това да не повлияе на останалата част от системата.

```
const mib = require("./HOST-RESOURCES-MIB.json");

function mockExtractSubtree(node) {
    const varbinds = [];
    Object.keys(mib).forEach(key => {
        const obj = mib[key];

        if (obj.oid !== undefined &&
            obj.syntax !== undefined &&
            obj.syntax.type === "Integer32") {

            if (obj.oid.startsWith(node.oid.join("."))) {
                varbinds.push({
                    oid: obj.oid.split(".").join(","),
                    name: obj.name,
                    value: Math.floor(Math.random() * 11),
                    description: obj.description
                });
            }

        }
    });
    return Promise.resolve(varbinds);
}

const snmpClient = function () {
    return {
        extractSubtree: mockExtractSubtree
    }
};

module.exports = snmpClient;
```

Фиг. 4.5 mockExtractSubtree - имплементация (версия 2 с Local MIB)

Както се вижда функцията приема **node** обект като в този случай ползваме само **oid** от него. Като резултат отново функцията връща обект който има **oid**, **name**, **value** и **description**. За да успея да тествам скриптовете и графиките ми трябваха различни данни. Най – лесно това може да бъде тествано с числа. Затова независимо, че в MIB-а има различни типове данни, филтрираме само тези който са от тип **Integer32**. След това филтрираме само тези, чиито OID започва с OID-то което е подал потребителят. След това за стойност генерираме случайно число между 0 и 10. Това позволява да тестваме както Matching групи, така и Between групи. След като тази имплементация беше готова и тествана, че работи, последва тестването с групи и скриптове. Нека видим блок диаграма на функцията за връщане на mock данни.



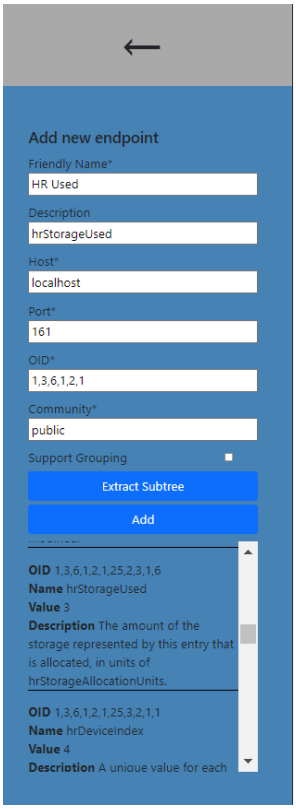
Фиг. 4.6 mockExtractSubtree workflow

Нека проследим през какви тестове минах ползвайки тази имплементация.

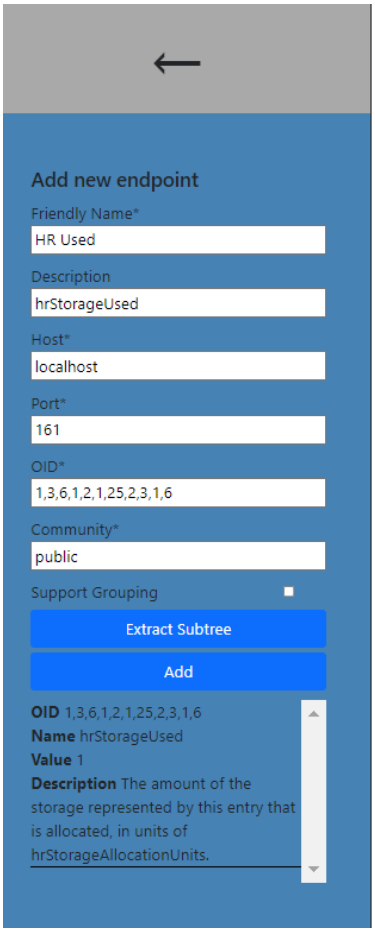
Тест 1: Extract Subtree

Резултати:

- при **OID = 1,3,6,1,2,1** получаваме всички OID-та които са под това OID в MIB дървото.
- при **OID = 1,3,6,1,2,1,25,2,3,1,6** получаваме един единствен резултат който е за това конкретно OID



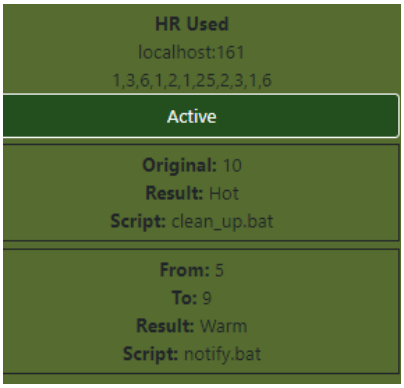
Фиг. 4.7 Extract Subtree - множество резултати (версия 2 с Mock Data)



Фиг. 4.8 Extract Subtree - резултат (версия 2 с Mock Data)

Тест 2: Добавяне на endpoint с Matching и Between групи
Резултати:

- Успешно беше записан endpoint със следните групи
  - Matching
    - **Original**: 10
    - **Result**: Hot
    - **Script**: clean_up.bat
  - Between
    - **From**: 5
    - **To**: 9
    - **Result**: Warm
    - **Script**: notify.bat



Фиг. 4.9 Добавен endpoint (версия 2 с Mock Data)

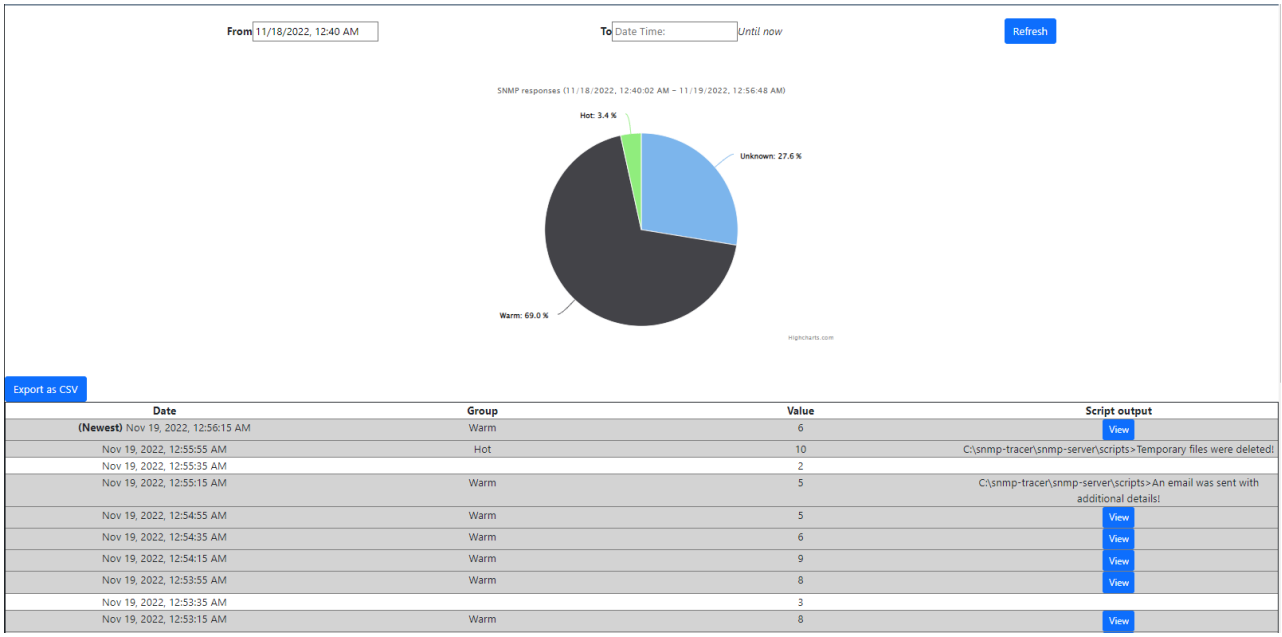
Резултатите които искаме да постигнем в следващите тестове с тези данни са следните

- когато стойността е между 5 и 9, ще бъде в група с име Warm и ще се изпълни notify.bat.
- когато стойността е равна на 10, ще бъде в група с име Hot и ще се изпълни clean_up.bat.

Тест 3: Преглед на резултати – таблица и резултати от скриптове

Резултати:

- Всички стойности за определеният интервал от време са видими в таблицата.
- При стойност между 5 и 9 групата е Warm.
- При стойност равна на 10 групата е Hot.
- Само за стойности попадащи в една от двете групи са изпълнени скриптове.
- При стойност между 5 и 9 е изпълнен скрипт notify.bat.
- При стойност равна на 10 е изпълнен скрипт clean_up.bat.
- Резултатите от скриптовете се визуализират успешно в таблицата.
- В pie chart графиката разбивката по групи показва правилните групи и правилните проценти.



Date	Group	Value	Script output
(Newest) Nov 19, 2022, 12:56:15 AM	Warm	6	View
Nov 19, 2022, 12:55:55 AM	Hot	10	C:\snmp-tracer\snmp-server\scripts>Temporary files were deleted!
Nov 19, 2022, 12:55:35 AM		2	
Nov 19, 2022, 12:55:15 AM	Warm	5	C:\snmp-tracer\snmp-server\scripts>An email was sent with additional details!
Nov 19, 2022, 12:54:55 AM	Warm	5	View
Nov 19, 2022, 12:54:35 AM	Warm	6	View
Nov 19, 2022, 12:54:15 AM	Warm	9	View
Nov 19, 2022, 12:53:55 AM	Warm	8	View
Nov 19, 2022, 12:53:35 AM		3	
Nov 19, 2022, 12:53:15 AM	Warm	8	View

Фиг. 4.10 Endpoint – резултати - таблица (версия 2 с Mock Data)

Тест 4: Преглед на резултати – CSV файл

Резултати:

- CSV файла съдържа само SNMP responses които се визуализират в таблицата.
- Съдържа следните данни: Date, Group и Value.
- Когато стойността не попада в никоя група записваме Unknown за група.

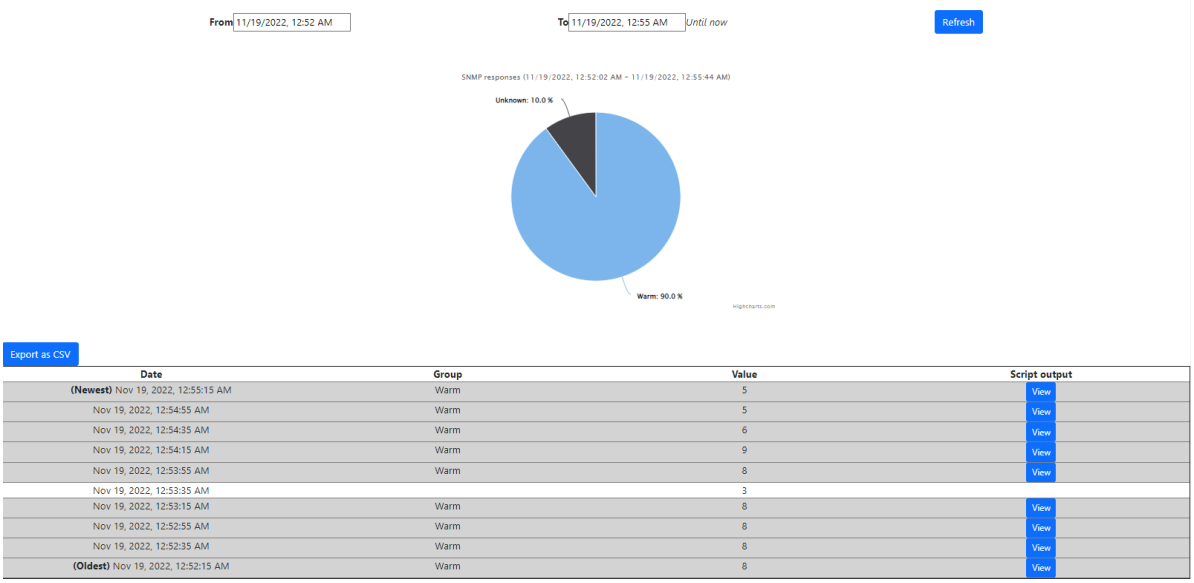
```
Date,Group,Value
Sat Nov 19 2022 00:56:15 GMT+0200 (Eastern European Standard Time),Warm,6
Sat Nov 19 2022 00:55:55 GMT+0200 (Eastern European Standard Time),Hot,10
Sat Nov 19 2022 00:55:35 GMT+0200 (Eastern European Standard Time),Unknown,2
Sat Nov 19 2022 00:55:15 GMT+0200 (Eastern European Standard Time),Warm,5
Sat Nov 19 2022 00:54:55 GMT+0200 (Eastern European Standard Time),Warm,5
Sat Nov 19 2022 00:54:35 GMT+0200 (Eastern European Standard Time),Warm,6
Sat Nov 19 2022 00:54:15 GMT+0200 (Eastern European Standard Time),Warm,9
Sat Nov 19 2022 00:53:55 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:53:35 GMT+0200 (Eastern European Standard Time),Unknown,3
Sat Nov 19 2022 00:53:15 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:52:55 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:52:35 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:52:15 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:51:55 GMT+0200 (Eastern European Standard Time),Warm,9
Sat Nov 19 2022 00:51:35 GMT+0200 (Eastern European Standard Time),Warm,5
Sat Nov 19 2022 00:51:15 GMT+0200 (Eastern European Standard Time),Unknown,3
Sat Nov 19 2022 00:50:55 GMT+0200 (Eastern European Standard Time),Warm,9
Sat Nov 19 2022 00:50:35 GMT+0200 (Eastern European Standard Time),Warm,9
Sat Nov 19 2022 00:50:15 GMT+0200 (Eastern European Standard Time),Warm,9
Sat Nov 19 2022 00:49:55 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:49:35 GMT+0200 (Eastern European Standard Time),Unknown,1
Sat Nov 19 2022 00:49:15 GMT+0200 (Eastern European Standard Time),Warm,9
Sat Nov 19 2022 00:48:55 GMT+0200 (Eastern European Standard Time),Unknown,0
Sat Nov 19 2022 00:48:34 GMT+0200 (Eastern European Standard Time),Unknown,4
Sat Nov 19 2022 00:48:14 GMT+0200 (Eastern European Standard Time),Unknown,3
Sat Nov 19 2022 00:47:54 GMT+0200 (Eastern European Standard Time),Warm,7
Sat Nov 19 2022 00:47:34 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:47:14 GMT+0200 (Eastern European Standard Time),Warm,6
```

Фиг. 4.11 Endpoint – резултати – CSV (версия 2 с Mock Data)

Тест 5: Преглед на резултати – филтрация по интервал от време

Резултати:

- При промяна на времевият интервал и натискане на бутона “Refresh”, данните в pie chart и таблицата се актуализират и се визуализират данни само за този интервал от време.
- CSV файла съдържа само SNMP responses само за този интервал от време.



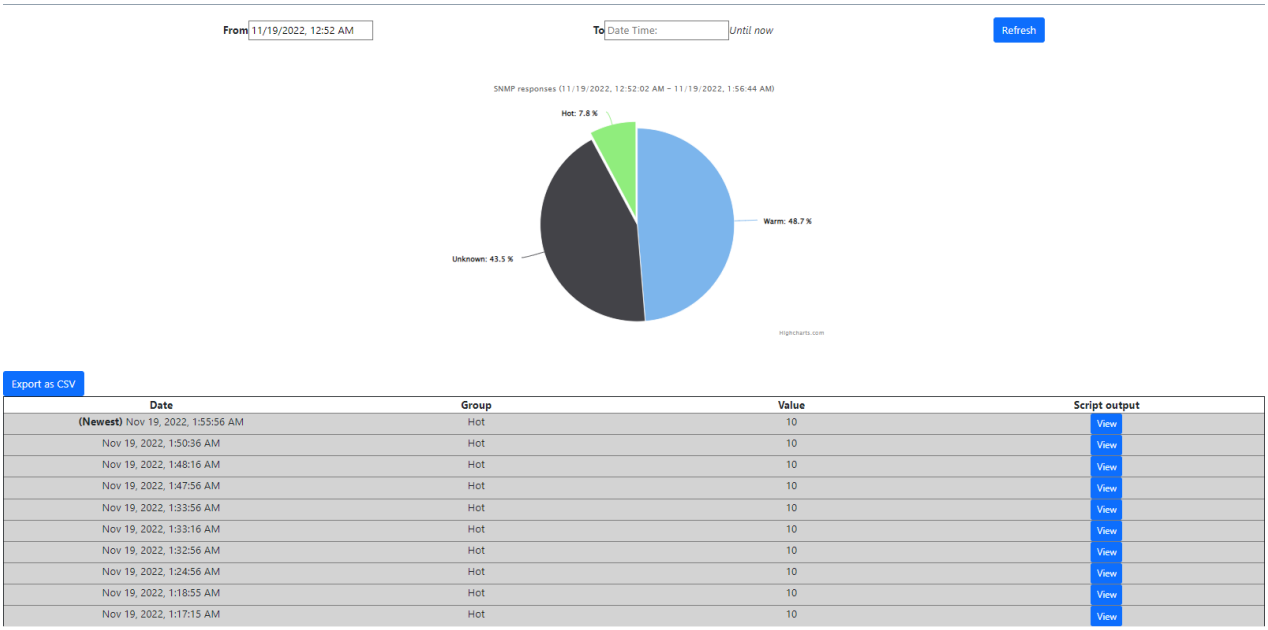
Фиг. 4.12 Endpoint – резултати – таблица – интервал от време (версия 2 с Mock Data)

```
Date,Group,Value
Sat Nov 19 2022 00:55:15 GMT+0200 (Eastern European Standard Time),Warm,5
Sat Nov 19 2022 00:54:55 GMT+0200 (Eastern European Standard Time),Warm,5
Sat Nov 19 2022 00:54:35 GMT+0200 (Eastern European Standard Time),Warm,6
Sat Nov 19 2022 00:54:15 GMT+0200 (Eastern European Standard Time),Warm,9
Sat Nov 19 2022 00:53:55 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:53:35 GMT+0200 (Eastern European Standard Time),Unknown,3
Sat Nov 19 2022 00:53:15 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:52:55 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:52:35 GMT+0200 (Eastern European Standard Time),Warm,8
Sat Nov 19 2022 00:52:15 GMT+0200 (Eastern European Standard Time),Warm,8
```

Фиг. 4.13 Endpoint – резултати – CSV – интервал от време (версия 2 с Mock Data)

Тест 6: Преглед на резултати – филтрация по група
Резултати:

- При избор на група от графиката, данните в таблицата се актуализират и се визуализират данни само за тази група.
- CSV файла съдържа само SNMP responses само за тази група.



Фиг. 4.14 Endpoint – резултати – таблица – филтър по група (версия 2 с Mock Data)

```
1  Date,Group,Value
2  Sat Nov 19 2022 01:55:56 GMT+0200 (Eastern European Standard Time),Hot,10
3  Sat Nov 19 2022 01:50:36 GMT+0200 (Eastern European Standard Time),Hot,10
4  Sat Nov 19 2022 01:48:16 GMT+0200 (Eastern European Standard Time),Hot,10
5  Sat Nov 19 2022 01:47:56 GMT+0200 (Eastern European Standard Time),Hot,10
6  Sat Nov 19 2022 01:33:56 GMT+0200 (Eastern European Standard Time),Hot,10
7  Sat Nov 19 2022 01:33:16 GMT+0200 (Eastern European Standard Time),Hot,10
8  Sat Nov 19 2022 01:32:56 GMT+0200 (Eastern European Standard Time),Hot,10
9  Sat Nov 19 2022 01:24:56 GMT+0200 (Eastern European Standard Time),Hot,10
10 Sat Nov 19 2022 01:18:55 GMT+0200 (Eastern European Standard Time),Hot,10
11 Sat Nov 19 2022 01:17:15 GMT+0200 (Eastern European Standard Time),Hot,10
12 Sat Nov 19 2022 01:16:35 GMT+0200 (Eastern European Standard Time),Hot,10
13 Sat Nov 19 2022 01:03:15 GMT+0200 (Eastern European Standard Time),Hot,10
14 Sat Nov 19 2022 01:00:15 GMT+0200 (Eastern European Standard Time),Hot,10
15 Sat Nov 19 2022 00:57:35 GMT+0200 (Eastern European Standard Time),Hot,10
16 Sat Nov 19 2022 00:55:55 GMT+0200 (Eastern European Standard Time),Hot,10
```

Фиг. 4.15 Endpoint – резултати – CSV файл – филтър по група (версия 2 с Mock Data)

Глава 5 - Приложимост на дипломната работа

Системата би могла да намери приложимост най – вече за следене на мрежи с малко на брой устройства. Ако бъде дистрибутирана и отделни хора следят отделни групи устройства, тя може да има приложимост и в по - големи мрежи. Системата би могла да послужи и за източник на данни за системи които могат да анализират резултатите посредством AI. Събирането на историческа информация може да послужи като помощно средство за анализиране на проблеми като например проблеми с определен Virtual Machine за определено време, като събраната информация може да послужи като помощно средство за да се провери дали всички параметри които биват следени за този интервал от време имат някакви отклонения. Тези исторически данни могат да послужат и като източник на информация с цел сравнение преди и след определена хардуерна/софтуерна промяна в устройството. Например upgrade на някой компонент или на софтуер който консумира от ресурсите на устройството.

Глава 7 - Изводи и претенции за самостоятелно получени резултати

Разработената Web базирана програмна система, е изцяло авторска. При разработката на дипломната работа на тема „Система за следене на мрежови услуги и устройства“ преминах през всички стадии на проектиране на софтуер. Първоначално трябваше да бъде проучен обстойно проблема както и подобни системи които са се опитали да го решат. След това се направи анализ на предметната област като се имаше в предвид системата да бъде оптимално ефективна. Премина се през множество вариации на крайния алгоритъм. След формулировка на алгоритъма се премина към създаването на системата като за целта беше предпочетено тя да бъде Web базирана за улеснение на потребителите които ще я използват. След имплементацията на основните модули на системата се премина през тестване на всяка една нейна функционалност.

Възможни бъдещи разработки за подобряване на функционалността на системата

- Да се добави функционалност за промяна на даден endpoint която да създава нов запис и да маркира старият като изтрит. Това ще позволи пазенето на история която е със съответният контекст.
- При промяна на даден endpoint да пазим допълнително metadata която да може да послужи като входна информация за AI който да анализира всички промени.
- Да се предостави опция за избор от предефинирани скриптове. Да има опция да се види съдържанието на скриптовете.
- Да се добави функционалност за AI анализ на резултатите.
- Да се предостави начин за интеграция с различни бази данни.
- Да се предостави възможност за ръчно избиране на интервала от време за всеки endpoint, за извличане на данните.

Използвана литература

1. Essential SNMP, O'Reilly, July 2001
2. SNMP MIB Handbook, Wyndham Press, March 2008
3. http://stackoverflow.com/
4. https://github.com/calmh/node-snmp-native
5. Cacti
6. SolarWinds Network Performance Monitor
7. Grafana Cloud
8. https://bestmonitoringtools.com/mibdb/mibdb_search.php

Приложения

GitHub repository
https://github.com/TrifonDardzhonov/snmp-tracer