

**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ- СОФИЯ, ФИЛИАЛ ПЛОВДИВ**  
**ФАКУЛТЕТ ПО ЕЛЕКТРОНИКА И АВТОМАТИКА**

---

КАТЕДРА .....

Дата на задаване:.....  
Срок за защита: декември 2022г

Утвърждавам: .....  
Декан: .....  
(.....)

**ЗАДАНИЕ**  
**ЗА ДИПЛОМНА РАБОТА**

на студента **Трифон Христов Дарджонов**, фак.№. **611316**  
Образователна степен: **ОКС магистър**, Специалност: **КСТ**

Тема: **Система за следене на мрежови услуги и устройства.**

1. Описание на конкретната инженерна задача и вида на крайните резултати, които трябва да се решат в оригиналната част от дипломната работа:

Целта на дипломната работа е да се реализира система за следене на мрежови устройства и услуги чрез протокола **SNMP**. Системата ще позволява да се следи **SNMP** трафика спрямо конкретна конфигурация, за конкретен **OID** в конкретна **MIB**. Системата трябва да позволява задаване и следене на интервали или конкретни стойности, като има възможност да изпълни дадено действие (скрипт, команда) ако текущата стойност попадне в интервала или е равна на конкретната търсена стойност. Да се направи сравнение със съществуващ софтуер за следене на **SNMP** трафика – **Casti**.

2. Изходни цифрови данни за изчислителната част. Функционални изисквания

- Да поддържа създаване/промяна/изтриване на конфигурация през графичния интерфейс на системата;
- Да визуализира графики за събраните данни към сегашният момент и история за определен период, за конкретен **endpoint**;
- Да могат да се извличат данните за конкретен **endpoint** в **CSV** файл;
- Да показва какви действия (скриптове или команди) са били изпълнени (и техният **output**) за определен период на базата на извлечените данни за конкретен **endpoint**;
- Да поддържа визуализация на **subtree** за конкретен **OID**.

3. Изходни литературни и други източници:

4. Съдържание на дипломната работа

4.1. Заглавна страница по образец

4.2. Оригинал на завереното дипломно задание

4.3. Съдържание

4.4. Увод

4.5. Глава 1 - Обзор - състояние на проблема по литературни данни;

4.6. Глава 2 - Теоретично решение на поставената задача;

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

4.8. Глава 4 - Изчислителна част;

4.9. Глава 5 - Приложимост на дипломната работа;

4.10. Глава 6 – Оценка на икономическата ефективност на разработката (ако е възможна);

4.11. Глава 7 - Изводи и претенции за самостоятелно получени резултати;

4.12. Използвана литература

4.13. Приложения

Консултант:.....  
( )

Научен ръководител: .....  
( )

Студент:.....

Ръководител катедра: .....  
( )

# Дипломна работа

Тема:.....  
.....  
.....

Студент: .....  
( две имена)

Фак №:.....

Специалност: .....

Образователна степен: ОКС .....

Факултет: ФЕА

Ръководител:.....

ТУ – София, Филиал Пловдив, 20xx г.

## ДЕКЛАРАЦИЯ

### ЗА ОРИГИНАЛНОСТ НА ДИПЛОМНАТА РАБОТА

Долуподписаният .....  
студент в специалност ..... на ФЕА, ТУ - София, Филиал Пловдив,  
дипломиращ се през 20xx/xx учебна година, фак. № .....,

ДЕКЛАРИРАМ, че изложеното по изпълнението на конкретните задачи в дипломната ми  
работа на тема:.....

с ръководител: .....

в обем ..... страници текст и .....страници Приложения;

включително: брой фигури: ..... брой таблици: ....., е резултат от собствена работа.

дата:.....

Подпис: .....

---

### МНЕНИЕ НА ДИПЛОМНИЯ РЪКОВОДИТЕЛ

#### ЗА РАБОТАТА НА ДИПЛОМАНТА

Дипломната работа е изпълнена според заданието в пълен обем / в обем ..... % (правилното се подчертава, ако изпълнението е под 100% при допускане до защита задължително се попълва мотивация за допускане) и може / не може да бъде допусната до защита.

Мотивация:.....  
.....  
.....  
.....

Дипломен ръководител:.....

(.....)

Избран рецензент: .....

Ръководител катедра :.....

(.....)

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ, ФИЛИАЛ ПЛОВДИВ

ФАКУЛТЕТ ЕЛЕКТРОНИКА И АВТОМАТИКА

Катедра: .....  
Образователна степен: .....

Утвърждавам.....  
(Председател на ДИК: .....)

**РЕЦЕНЗИЯ**  
**НА ДИПЛОМНАТА РАБОТА**

на студента: ....., фак.№.....спец.....

на тема: .....

1. Описание на зададената инженерна задача и исканите крайни резултати:.....
2. Характер на дипломната работа:.....
3. Актуалност на разглеждания проблем:.....
4. Трудност на поставената задача:.....
5. Обем на дипломната работа: брой страници: ..... брой фигури: ..... брой таблици: .....
6. Оформление: форма и шрифт: ..... номерация: ..... качество на чертежите:.....  
общо оформление: .....
7. Обща и техническа грамотност: правопис: ..... стил: .....  
владее на терминологията: ..... коректност на означенията: .....
8. обоснованост на решенията и изводите:.....
8. В каква степен абсолвентът е проявил познаване на общите и специални проблеми на специалността: .....
9. В каква степен е решена поставената конкретна инженерна задача:.....
10. Доколко са съвременни използваните методи на решение: .....
11. Кой от претенциите са основателни и има ли в нея необявени оригинални резултати: .....
13. Оценка на увода и на работата с литературните източници: .....
14. Оценка на глава 3 - апаратна и/или софтуерна част и експеримент: .....
15. Оценка на глава 4 - изчислителна част: .....
16. Оценка на глава 5 -- приложимост на дипломната работа .....
17. Оценка на глава 6: икономическа оценка на резултатите от дипломната работа и оценка на техническата ефективност.....
18. Оценка на глава 7 - изводи :.....
19. Основни достойнства на дипломната работа:.....
20. Забележки, недостатъци и допуснати грешки в дипломната работа: .....
21. Предложение за допускане до защита и оценка: .....

Дата:.....

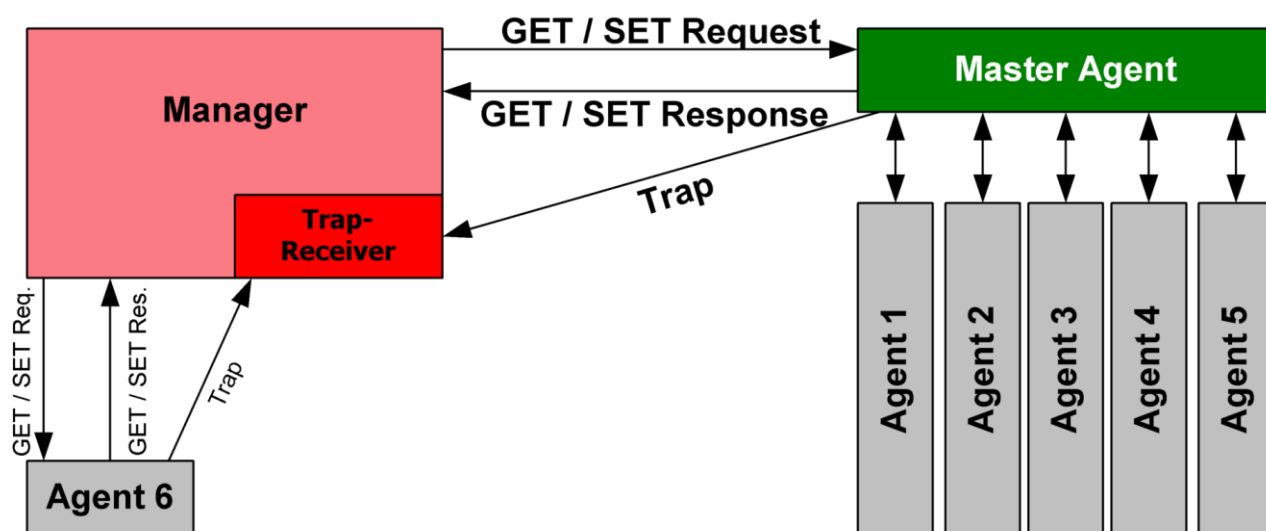
Рецензент: .....  
(.....)

## Съдържание

Увод.....	- 7 -
Глава 1 - Обзор - състояние на проблема по литературни данни.....	- 8 -
Глава 2 - Теоретично решение на поставената задача.....	- 9 -
Компоненти на SNMP: .....	- 9 -
MIB (Management Information Base).....	- 9 -
OID (Object Identifier) .....	- 10 -
SNMP агент .....	- 11 -
Отдалечен мониторинг RMON.....	- 11 -
Основни функции на SNMP .....	- 11 -
Използване на SNMP .....	- 11 -
Обобщение.....	- 11 -
Теоретична основа на системата цел на дипломната работа .....	- 12 -
Създаване на конфигурация .....	- 13 -
Преглед на конфигурация .....	- 15 -
Преглед на резултатите.....	- 16 -
Глава 3 - Описание на използваната апаратна (схемна) и/или софтуерна част.....	- 18 -
UI (Client) – Angular .....	- 18 -
Backend (Server) – Node .....	- 21 -
REST APIs .....	- 22 -
Database.....	- 23 -
SNMP Listener .....	- 26 -
SNMP Client.....	- 29 -
Group Finder.....	- 30 -
Script Runner .....	- 32 -

## Увод

Още от 80-те години на 20 век хората са имали нужда от tool-ве и средства за мониторинг на техните мрежи и системи. 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 десетки устройства това може бъде предизвикателство поради ред на брой причини като се почне от различният софтуер до типа на устройствата (компютри, рутери и др.). 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.



Фиг. Принцип на SNMP комуникацията

В следващите глави ще разгледаме в детайли какво представлява SNMP протокола. Това което искам да въведа тук като встъпваща информация е, че този протокол стандартизира взимането на информация за различни параметри на различни типове 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 не само с проследяването, но и с взимането на 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 ползват. Всеки хардуер има срок на експлоатация и потенциални проблеми винаги може да има. 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 устройства и всякакъв вид хардуер в тях, 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 специален софтуер който следи десетки дори 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 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 адресират.

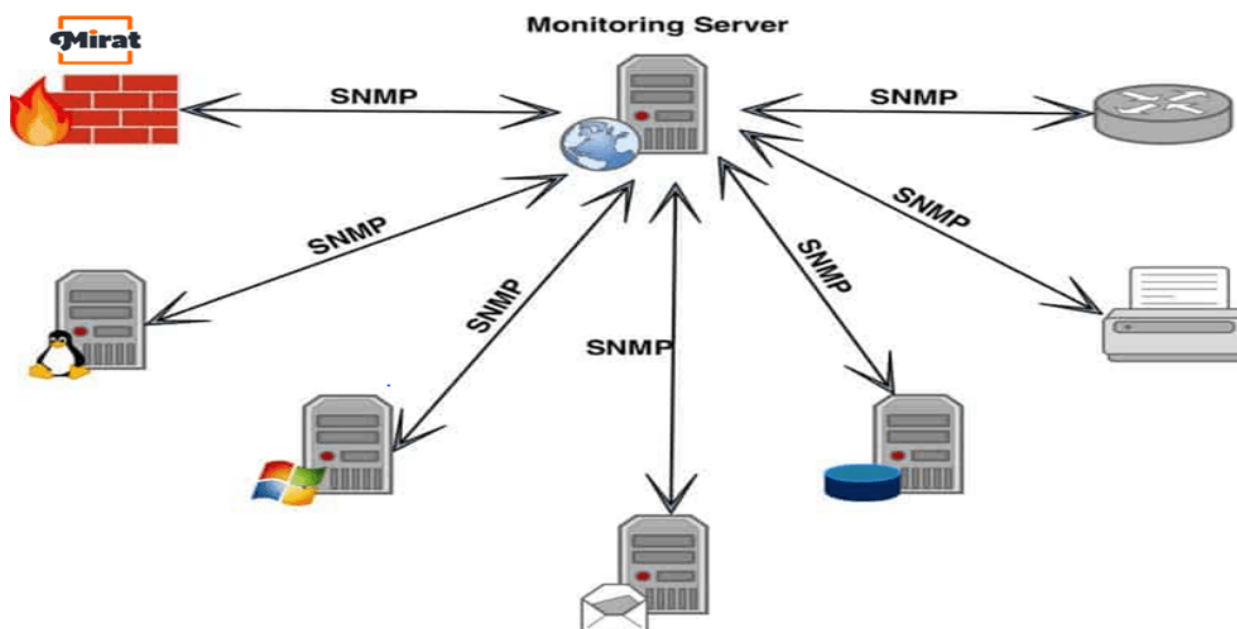
Друг от основните проблеми е, че самите потребители нямат голяма свобода в самия мониторинг, имат възможност да видят тестовете които системата предоставя и ако има проблем, трябва сами да правят troubleshooting къде точно е проблема понеже понякога самата грешка не е много разбираема.

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 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.



## Глава 2 - Теоретично решение на поставената задача

Решението което е имплементирано в тази дипломна работа е въз основа на SNMP протокола. 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).

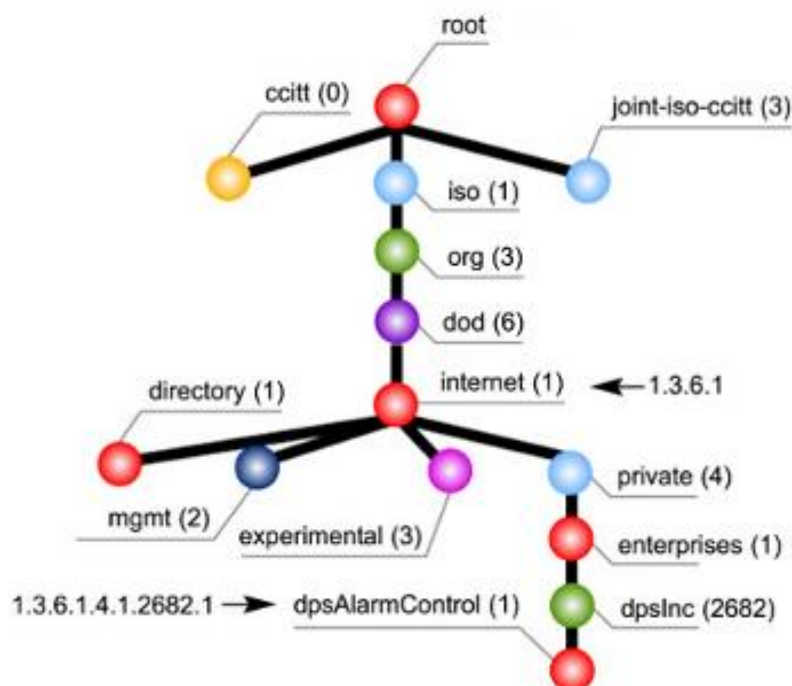


Фиг. SNMP overview

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. 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. Пример за OID).

1 . 3 . 6 . 1 . 4 . 1 . 2682 . 1 . 4 . 5 . 1 . 1 . 99 . 1 . 1 . 6

*Фиг. Пример за OID*

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

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 Display for this alarm point
.1	Point	This is the alarm Point number
.6	dpsRTUASState	This is the state of the alarm point (set, clear, etc.)

Фиг. 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.

#### 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.

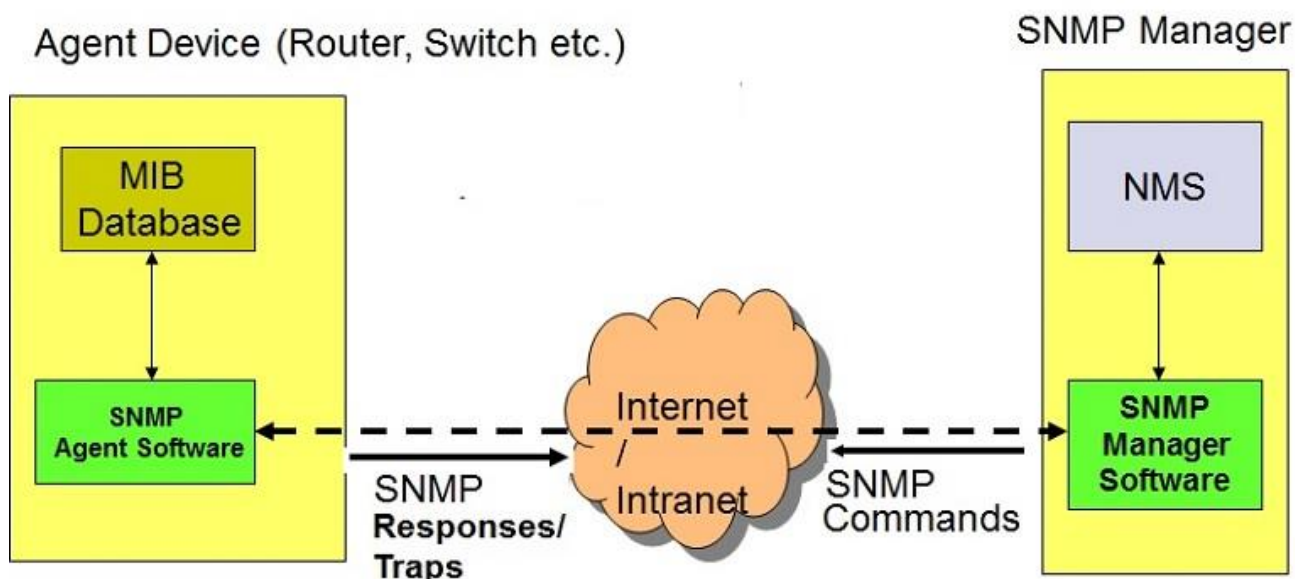
#### Използване на SNMP

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.

# SNMP Architecture



Фиг. SNMP - Принцип на действие

Теоретична основа на системата цел на дипломната работа

Целта на текущата дипломна работа не е самото конфигуриране на SNMP агентите, а създаването на система (SNMP Manager Software) която ги ползва (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 ползва 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.

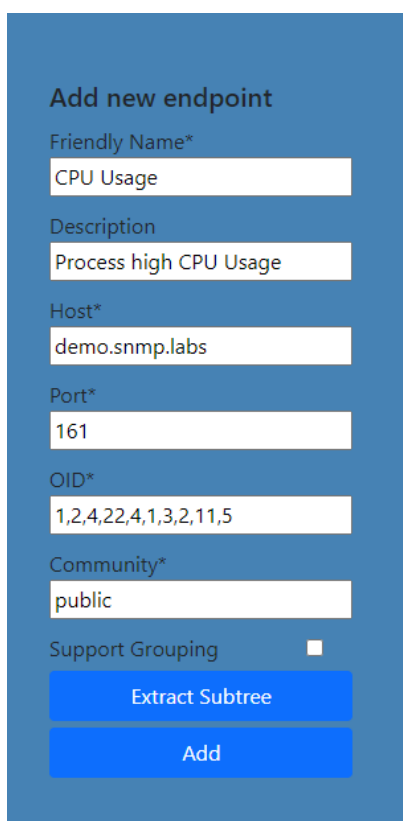
## ВАЖНО

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 ползва 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. Форма за добавяне на нов SNMP endpoint

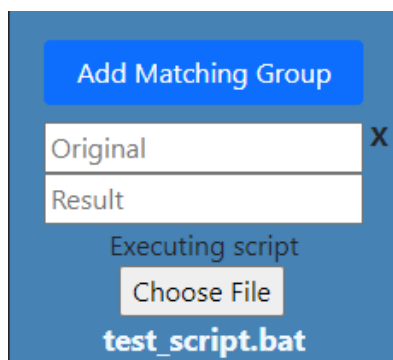
Под самата форма има checkbox наименован “Support Grouping”. Неговата функция е да индикираме, че искаме на базата на стойностите които получаваме за този OID да изпълняваме определен скрипт. При избиране на тази опция се визуализират следните две опции:



Фиг. 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:

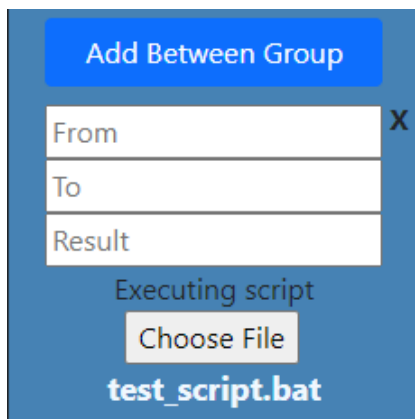


Фиг. Форма за добавяне на Matching група

- **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 ползва 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. Форма за добавяне на Between група

- **From** – Това е минималната стойност на която стойността трябва да отговаря за да можем да идентифицираме, че тя попада в тази група. Например “0”.
- **To** - Това е максималната стойност на която стойността трябва да отговаря за да можем да идентифицираме, че тя попада в тази група. Например “30”.
- **Result** – Това е стойност която потребителят може да ползва 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

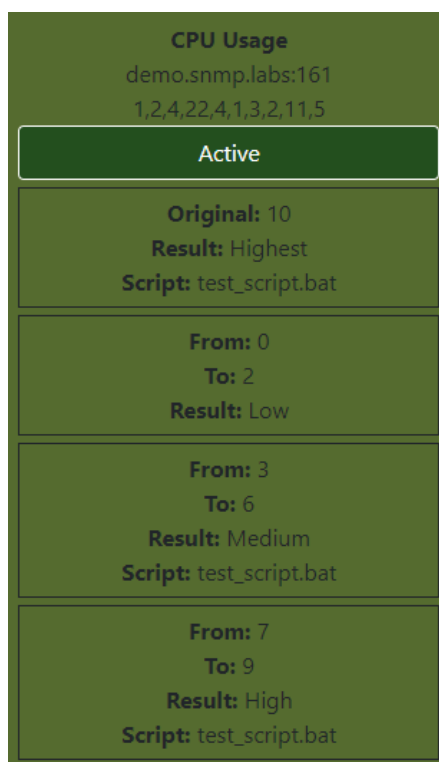
70 – 90

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. 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:

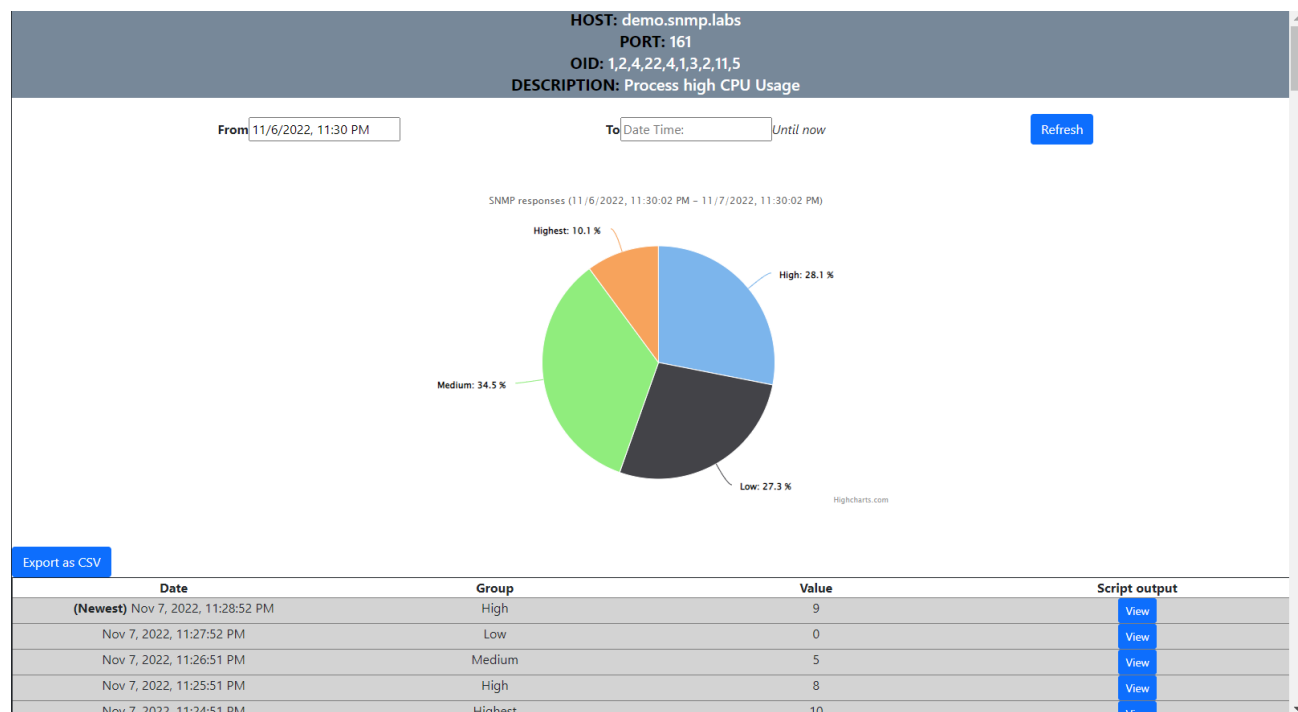
- **Active** – Когато е в този статус, системата ще събира и обработва информацията за този endpoint.
- **Deactivated** – Когато е в този статус, потребителят ще може да види този endpoint в системата, но системата няма да извършва никакви действия свързани с него. Този статус е удобен когато искаме временно да спрем следенето на даден 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 няма опцията 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 сега. 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. 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 конкретния 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 ползва 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.

Export as CSV			
Date	Group	Value	Script output
(Newest) Nov 7, 2022, 11:41:52 PM	Medium	4	C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11
Nov 7, 2022, 11:40:52 PM	Low	1	<a href="#">View</a>
Nov 7, 2022, 11:39:52 PM	Low	2	<a href="#">View</a>
Nov 7, 2022, 11:38:52 PM	Highest	10	<a href="#">View</a>
Nov 7, 2022, 11:37:52 PM	Low	0	<a href="#">View</a>
Nov 7, 2022, 11:36:52 PM	Low	1	<a href="#">View</a>
Nov 7, 2022, 11:35:52 PM	Medium	4	<a href="#">View</a>
Nov 7, 2022, 11:34:52 PM	Medium	4	<a href="#">View</a>

Фиг. 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 и двете.

## UI (Client) – Angular

Избрах Angular за клиентската страна поради следните причини:

- 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.

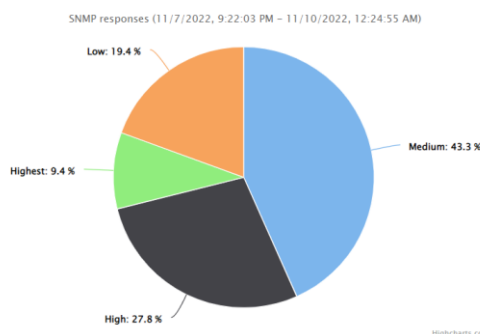
Ползвам съм следните допълнителни библиотеки:

- **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. 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.



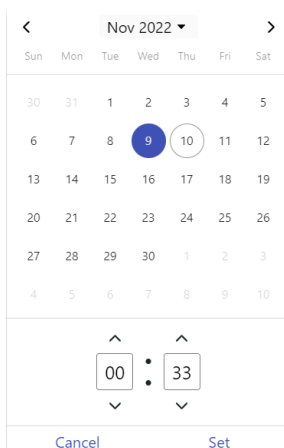
Фиг. Графика изобразяваща разпределението на резултатите по групи

- **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.

From  To

*Фиг. Контроли за избор на начална и крайна дата*

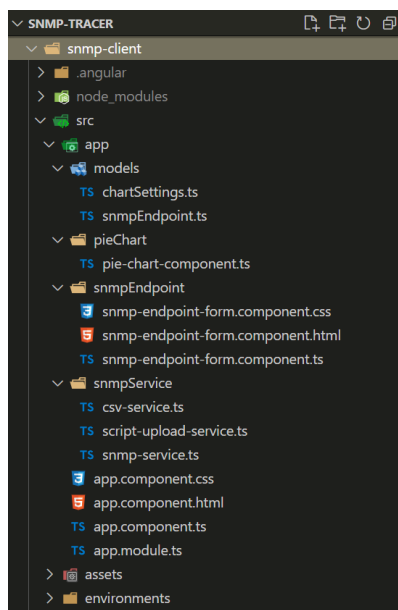
Потребителят има възможност да избере година, месец, ден, час и минути.



*Фиг. 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 един service наречен SNMPService който вътрешно ползва HttpClient-a. 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 тестваме код съм добавил още 2 service-a: един който съдържа логиката за качване на файл (това са файловете които до сега наричаме скриптове, асоциирани със групи на съответен endpoint) и един който съдържа логиката за генериране и сваляне на резултатите във вид на CSV файл.



*Фиг. Структура на файловете в клиента*

### **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) – Node

Избрах Node за сървърната част поради следните причини:

- 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.

Ползвам съм следните допълнителни библиотеки:

- **Express (версия 4.16.3)** – Web Framework. Ползвам 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 ползва за изпращане на SNMP заявки и получаването и обработването на резултатите до получаването им във формат удобен за работа.

Сега 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.

## REST APIs

Както споменахме по – рано за HTTP сървър се ползва Express. 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. 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 ползва от “Extract subtree” бутона когато е част от формата за добавяне на нов SNMP endpoint. Идеята е да дадем на потребителя начин да види какъв е типа на резултата който стои зад това OID.
POST	/scriptsOutputs	връща резултата от изпълнен скрипт в контекста на определен резултат във времето.
POST	/snmpEndpoint/upload	записва скрипт подаден от потребителя за определен endpoint и група.

## 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 нареченият **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 ползвателя 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-a 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 съхранение от потребителя endpoints.
- **snmpEndpointRepository** – Зад този интерфейс се съдържа логиката за:
  - Вземане 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. 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 ползва 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 нещо се случи да rollback-не всички направени промени), затова използвам и термина фасада защото по-добре описва функциите на сегашната имплементация.

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

### 1. SNMP endpoint

- Repository файл:** snmpEndpointRepository.js
- 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
        }
    ]
}

```

## 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;

```



Отдолу можете да видите реални данни.

```
snmp-server > database > csv > snmpResponses.csv
1 id,endpointId,oid,host,port,community,value,groupId,group,dateticks
2 ffa3cc1e1ac6cddb15c80e6711e68e65,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,4,3,Medium,1668250841492
3 cf9daa17e0d17f22c116de33e97e9399,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,1,2,Low,1668250901495
4 af0de1678d70f61e421d29164e97f8ac,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,8,4,High,1668250961504
5 3b9135d1447d398e1b3e81826fa08ced,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,8,4,High,1668251021515
6 3961edd9ea2ae2c067cd1a3929336c86,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,10,1,Highest,1668251081523
7 0d4d98dcb517cce240ee82e211267001,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,9,4,High,1668251141535
8 a6643502c4f9e6a6de694debfbefc16a,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,10,1,Highest,1668251201548
9 ea4b8ca66fd41283ff890588b0c2b02b,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,7,4,High,1668251261550
10 6ec05002efbd498cf07eaa9f5ea36e9c,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,3,3,Medium,1668251321555
11 808becd4d2477b137bc5050d4f2fdc2c,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,0,2,Low,1668251381568
12 bdd499ad1029738fbcc13f102856fe1,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,2,2,Low,1668251441579
13 f89be5de735e626dd9ff9c9f9962475d,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,1,2,Low,1668251501592
14 b4947f5353f7e9d3deafac4f05634cf,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,7,4,High,1668251561601
15 ba4e3ba66d05024e73b26c8296d8d9d9,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,3,3,Medium,1668251621604
16 d37a2a3d82dbb28afbbf209d04594a19,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,4,3,Medium,1668251681608
17 53e7d785d26efeb86534ed15ab60f97d,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,3,3,Medium,1668251741609
18 8595afff62c138ac8b6bf2393382ee93,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,0,2,Low,1668251801618
19 70772e2cc2da91c15b4903187d3dcb0a,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,7,4,High,1668251861622
20 1a16c4a0b89ff77dca815ba8a5a0a509,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,4,3,Medium,1668251921636
21 9adc824c6b0cf10f4ed5ba818ec10c38,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,2,2,Low,1668251981644
22 092237f60a01ac9f86a9831308ba9d96,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,1,2,Low,1668252041645
23 f13fa05b8057d695b391426406f27367,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,5,3,Medium,1668252101654
24 1ba82255e2570795b480e53b811867,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,2,2,Low,1668252161664
25 40e63f707920b8911cfda177b94979cd,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,6,3,Medium,1668252221665
26 a5f3a902b3c1ad991778e0f1c5cf06ce,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,8,4,High,1668252281675
27 e47e889c1bfff57c73b5f2b7c92e15811,2,"1,2,4,22,4,1,3,2,11,5",demo.snmp.labs,161,public,5,3,Medium,1668252341682
```

### 3. Script output

- Repository файл: scriptOutputRepository.js
- Къде пазим данните и в какъв формат: CSV файл с име scriptsOutputs.csv.  
Пазим следната информация за всеки резултат:

```
// 16 символен string. Пример: ffa3cc1e1ac6cddb15c80e6711e68e65
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;
```

Отдолу можете да видите реални данни.

```
snmp-server > database > csv > scriptsOutputs.csv
1 snmpResponseId,text,endpointId,groupId,script,dateticks
2 ffa3cc1e1ac6cddb15c80e6711e68e65,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
3 af0de1678d70f61e421d29164e97f8ac,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
4 3b9135d1447d398e1b3e81826fa08ced,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
5 3961edd9ea2ae2c067cd1a3929336c86,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
6 0d4d98dcb517cce240ee82e211267001,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
7 a6643502c4f9e6a6de694debfbefc16a,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
8 ea4b8ca66fd41283ff890588b0c2b02b,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
9 6ec05002efbd498cf07eaa9f5ea36e9c,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
10 b4947f5353f7e9d3deafac4f05634cf,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
11 ba4e3ba66d05024e73b26c8296d8d9d9,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
12 d37a2a3d82dbb28afbbf209d04594a19,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
13 53e7d785d26efeb86534ed15ab60f97d,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
14 70772e2cc2da91c15b4903187d3dcb0a,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
15 1a16c4a0b89ff77dca815ba8a5a0a509,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
16 f13fa05b8057d695b391426406f27367,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
17 40e63f707920b8911cfda177b94979cd,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
18 a5f3a902b3c1ad991778e0f1c5cf06ce,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
19 e47e889c1bfff57c73b5f2b7c92e15811,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
20 65609c339e82f177af2f3d6deb22fd,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
21 808becd4d2477b137bc5050d4f2fdc2c,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
22 0c0535de651b9e15a6f1ce450c3d92b6,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
23 2dd71dc19872c9c85ec8254eed868749,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
24 f9c6d7c45ce874b2511ad75e08ddb65,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,1,2_1_test_script.bat,166
25 f9a0b7a5db6d8b97f68be09240ac773,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
26 e1ebcb339ba7eb548a201ea674af7319,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,3,2_3_test_script.bat,166
27 0f7703a23afe685d83ad3d25d4d070aa,C:\snmp-tracer\snmp-server\scripts>npm --version;6.14.11,2,4,2_4_test_script.bat,166
```

## SNMP Listener

В snmpListener.js се съдържа core логиката на самата система свързана със изпращането на SNMP заявки, записа на резултатите и пускането 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
    }
};

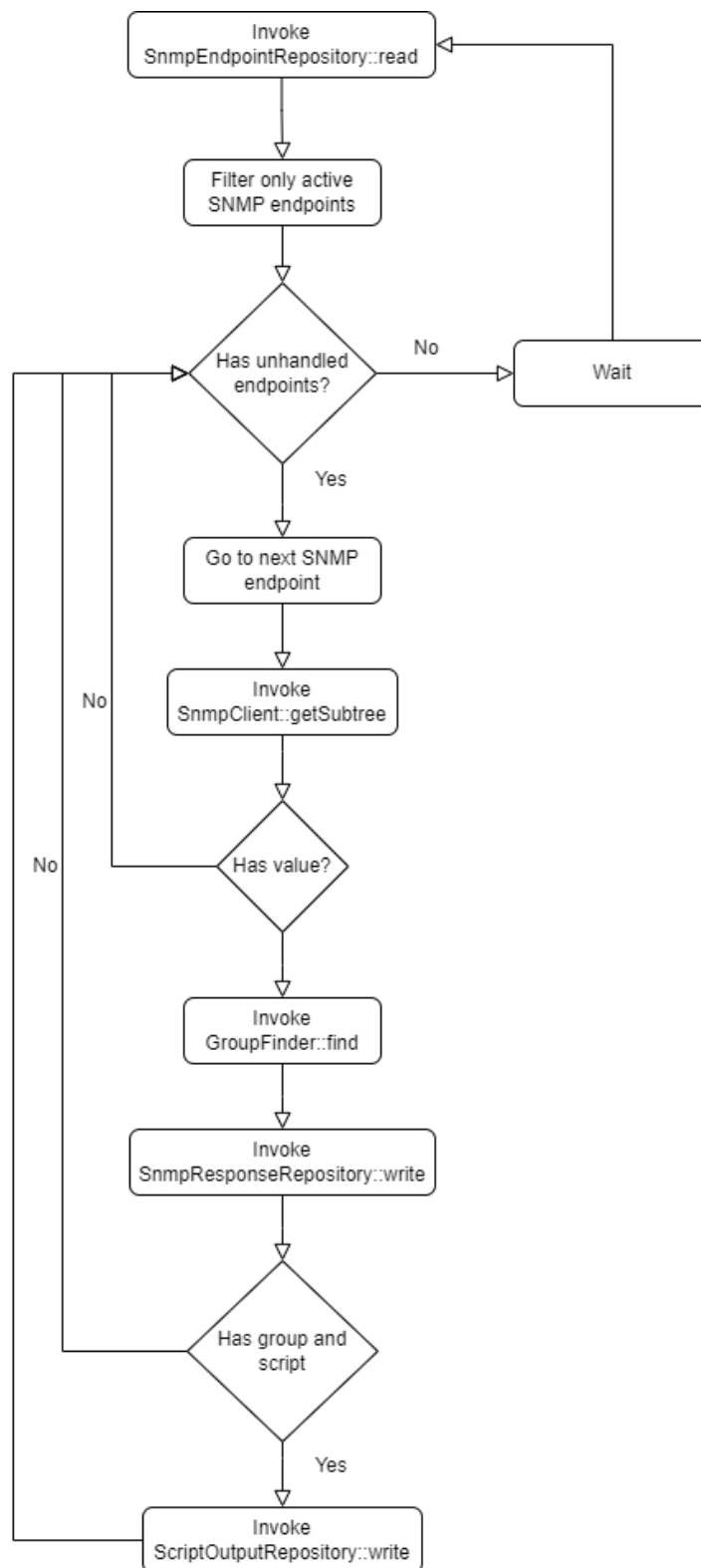
module.exports = snmpListener;

```

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-a.
  - 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.

1. Не извършва никакво действие



## *Φu2. 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;
```

Тя ползва snmp-native пакета 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
}

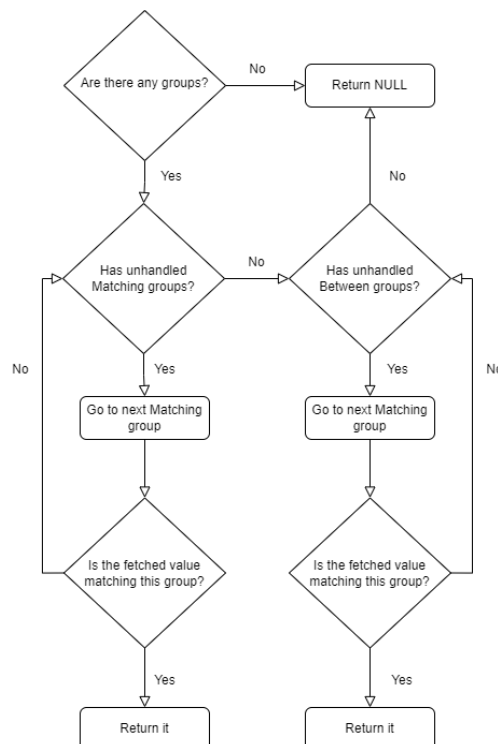
function searchInBetweenGroups(groupingBetween, value) {
  if (groupingBetween.length > 0) {
    for (let i = 0; i < groupingBetween.length; i++) {
      const from = Number(groupingBetween[i].from);
      const to = Number(groupingBetween[i].to);
      if (from <= Number(value) && Number(value) <= to)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

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. 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
  }
};
```

При подадено име на скрипт, функцията го търси в папката “scripts”, като събира всички output-и от изпълнението на скрипта и накрая връща един общ стринг от конкатинираните отделни output-и като ползва “;” за разделител.