TecnoCampus

Centres universitaris adscrits a la

upf. Universitat Pompeu Fabra Barcelona

Grau en Enginyeria Informàtica de Gestió i Sistemes de Informació

Plataforma de gestió de dispositius de xarxa

Memòria final

Albert Castaño Bret
Tutor: Pere Barberan Agut

TecnoCampus Mataró-Maresme

Dedicatòria

A la meva mare. Gràcies pel suport incondicional durant tot el procés.

Índex

1. Objecte del treball 1

1.1. Motivacions 1

1.2. Justificacions 1

2. Estudi previ 3

2.1. Context 3

2.2. Antecedents 4

2.3. Necessitats d'informació 7

2.3.1. Emuladors de xarxa 7

2.3.2. Llenguatges i Frameworks de programació 9

2.3.3. Eines d'Automatització de Xarxes 10

3. Objectius i abast 13

4. Metodologia 15

4.1. Iteracions **17**

5. Requeriments 19

5.1. Requeriments funcionals 19

5.2. Requeriments tecnològics 19

6. Desenvolupament 21

6.1. Backend 21

6.1.1. API 21

6.1.2. Models 25

6.1.3. Serveis – Automatització de xarxa 32

6.1.4. Base de dades 36

6.1.5. Data 37

6.2. Frontend 38

6.2.1. Pages 38

6.3. GNS3 44

6.3.1. Disseny de topologia 44

6.3.2. Configuració de GNS3 45

7. Conclusions 47

8. Possibles ampliacions 49

9. Bibliografia 51

Índex de figures

Fig. 6.1. Declaració de Blueprints....21
Fig. 6.2. Model Device 25
Fig. 6.3. Declaració de relacions del model Device 26
Fig. 6.4. Model DeviceArpEntry 27
Fig. 6.5. Model DeviceBgpNeigbor 27
Fig. 6.6. Model DeviceConfig 29
Fig. 6.7. Model DeviceInterface 29
Fig. 6.8. Model DeviceStatus 31
Fig. 6.9. Model DeviceTask 32
Fig. 6.10. Model Host 32
Fig. 6.11. Funció monitor_devices 33
Fig. 6.12. Funció process_device 34
Fig. 6.13. Funció update_device_states 35
Fig. 6.14. Funció monitor_hosts 35
Fig. 6.15. Funció process_hosts 36
Fig. 6.16. Funció update_hosts 36
Fig. 6.17. Devices.yaml 37
Fig. 6.18. HomePage 39
Fig. 6.19. DevicesPage 39
Fig. 6.20. DevicesSubPage 40
Fig. 6.21. TaskDataPage 41
Fig. 6.22. TaskEditPage 42
Fig. 6.23. TaskSchedulerPage 42
Fig. 6.24. HostsPage 43
Fig. 6.25. Topologia de GNS3 44

Índex de taules

Taula 4.1 - Metodologies 16

1. Objecte del treball

L'objecte d'aquest treball es centra en el disseny e implementació d'una solució de software avançada per l'automatització i monitorització de xarxes en entorns empresarials i/o educatius. L'objectiu es crear una plataforma robusta que permeti als administradors de xarxa gestionar dispositius i serveis de manera eficient, automatitzar la configuració i el manteniment i monitoritzar el rendiment de tota la xarxa en temps real. A més a més, es busca fomentar la comprensió de les millors pràctiques envers a l'automatització de xarxes i contribuir al cos de coneixement amb una eina pràctica i adaptada a les necessitats actuals de les empreses.

1.1. Motivacions

La motivació rere el projecte té a veure amb la necessitat creixent de disposar de solucions d'automatització de xarxes que puguin gestionar la complexitat de les últimes i més noves infraestructures tecnològiques. Amb el fort augment dels dispositius connectats i la importància crítica de mantenir una xarxa fiable i segura, les eines que permetin l'automatització, la gestió i la monitorització són essencials per qualsevol organització.

Aquest treball pretén alinear-se amb els últims avanços en tecnologies de xarxes i respon a la demanda del mercat d'eines sofisticades que redueixin el marge d'error humà i optimitzin els recursos operatius.

1.2. Justificacions

La proposta d'aquest treball de final de grau va ser formulada pel professor Pere Barberan amb l'interès de reflectir la rellevància acadèmica i pràctica sobre l'automatització de les xarxes. Aquest projecte fixa en el marc de referencia de les ciències de la computació i la enginyeria de sistemes, degut a que aborda directament els reptes tècnics associats a la gestió de xarxes de mitjana i alta complexitat.

La implementació d'aquesta solució no tan sols té un valor educatiu significatiu, sinó que també té el potencial d'oferir millores operatives a qualsevol organització que adopti la tecnologia.

2. Estudi previ

2.1. Context

En la era actual de la transformació digital, les xarxes corporatives s'enfronten a un increment en la complexitat i diversitat de les seves estructures [1]. Impulsades per la migració a solucions en el núvol i la expansió de IOT, aquestes xarxes haurien de procurar ser més flexibles i segures per gestionar el tràfic i les operacions descentralitzades [1]. Per altra banda, la proliferació de dispositius IOT i mòbils ha incrementat la demanda de l'ample de banda i ha plantejat nous reptes de seguretat, mentre que l'increment del treball remot ha ampliat el perímetre de les xarxes corporatives més enllà del límits físics tradicionals [2].

Per tractar aquestes qüestions, s'està impulsant cada cop més l'automatització de xarxes i les operacions de TI intel·ligents per gestionar la configuració, implementació i manteniment d'un gran número de dispositius [3]. La seguretat i la gestió de xarxes estan convergint, i alguns exemples són les implementacions Zero Trust o SASE que combinen serveis de xarxa i seguretat en plataformes unificades, proporcionant accés segur independentment de la ubicació o el dispositiu. [4] També cal mencionar que l'adopció de tecnologies com el 5G privat i el Wi-Fi en casos d'ús específic, juntament amb la creixent dependència d'entorns multi-núvol, està transformant la forma en que les xarxes són construïdes i gestionades.

És així que aquest entorn destaca la necessitat d'eines de gestió de xarxes més eficients i automatitzades que puguin abordar l'expansió de la infraestructura de la xarxa i simplificar la càrrega de treball del personal de TI. Tot això mitjançant la consolidació de la gestió de les xarxes en una única eina per visualitzar i gestionar canvis a les xarxes amb risc mínims.

2.2. Antecedents

A l'explorar els antecedents de l'automatització i monitorització de xarxes ens trobem amb el relat històric d'avenços que han definit les bases per les solucions contemporànies. En un primer moment, la gestió de les xarxes es duia a terme de forma manual. Ben aviat, les xarxes es van tornar més complexes van aparèixer les primeres tecnologies capaces de millorar la configuració i manteniment de les xarxes. La investigació i el desenvolupament han seguit un camí enfocat en la optimització de recursos, millora de seguretat i maximització de l'eficiència operativa. D'entre tota la historia, es poden destacar algunes:

- **Protocols de gestió de xarxa simple (SNMP):** Utilitzat per recopilar informació i configurar dispositius de xarxa de forma remota [5]. Abans de l'automatització, els administradors de xarxa utilitzaven aquest protocol per tasques de monitorització bàsiques i diagnòstics. No obstant, havien d'interpretar i actuar sobre les dades manualment.
- **Eines de Línia de Comandes:** Eren la interfície primària per la configuració de dispositius de xarxa [6]. Cada canvi requeria la intervenció manual de l'administrador per introduir comandes específiques, cosa que suposava pèrdues de temps i disposició a errors.
- **Taules d'enrutament estàtic:** Es configuraven manualment per definir com els paquets havien de ser dirigits a través de la xarxa. Això requeria un coneixement detallat de la xarxa i actualitzacions constants per reflectir qualsevol canvia en la topologia de la xarxa [7].
- **Firewall i Sistemes de Detecció de Intrusions (IDS) Manuals:** Els administradors configuraven i actualitzaven regles de seguretat manualment per protegir contra amenaces, fet que esdevenia un procés continu i detallat per mantenir-se al dia amb les noves vulnerabilitats [8].
- **Panells de Control i Monitors de Xarxa Bàsics:** Proporcionaven visualitzacions rudimentàries de l'estat de la xarxa i el seu rendiment [9]. Els administradors feien ús d'aquesta eina per supervisar la xarxa, però qualsevol acció correctiva necessitava ser duta a terme manualment [10].
- **Remote Monitoring (RMON):** Permetia la monitorització a distància de la xarxa, però requeria que els administradors interpretessin les dades recopilades i determinessin les accions a seguir per resoldre problemes o millorar el rendiment [11].

- **TFTP:** S'utilitzava per transferir arxius de configuració i firmware a dispositius de xarxa. Era una tasca manual, amb els administradors iniciant i supervisant cada transferència per assegurar que els dispositius s'actualitzaven correctament [12].

Amb el temps, els sistemes de gestió de xarxa han anat evolucionant des de simples eines de monitorització fins a plataformes sofisticades que integren AI Ops i ofereixen una visió holística de l'ecosistema de la xarxa. Existeixen diversos estudis que destaquen la necessitat de solucions que siguin capaces d'adaptar-se als canvis i el l'augment del tràfic de les xarxes.

Un exemple, és l'estudi fet per dos enginyers elèctrics pakistanesos l'any 2023, "Network Automation". La investigació duta a terme pel doctor Tayyab Muhammad i el llicenciat Muhammad Tahir Munir té com a propòsit explorar l'automatització de xarxes a l'economia contemporània, enfocant-se en mètodes i beneficis per millorar la gestió de les xarxes. L'estudi duu a terme una forta investigació a través d'una combinació de revisió de literatura i dades empíriques recollides amb enquestes i entrevistes amb administradors de xarxes. En concret, es dediquen a investigar l'arquitectura d'automatització, eines com el motor de plantilles Jinja i el gestor d'inventari Nornir, així com pràctiques de registre d'aplicacions, depuració i proves de codi en temps real. Finalment, conclouen que l'automatització de xarxes és beneficiosa degut a que incrementa l'eficiència operativa, redueix els errors humans i millora la seguretat de la xarxa. També aconsellen la inversió en formació i recursos per una implementació efectiva [13].

En un altre estudi més concret, "Automatización de redes utilizadas para EoT: Automatización de redes con Netmiko" (Jean Carlos Tandazo Tandazo), s'investiga sobre aspectes pràctics i tècnics per implementar l'automatització mitjançant eines específiques com Netmiko, una biblioteca de Python utilitzada per simplificar la gestió de dispositius de xarxa a través de de SSH. Aquest cas és de gran interès ja que s'analitza Netmiko, fet que permetrà conèixer per quin tipus de tasques pot servir la llibreria. També permetrà conèixer altres llibreries a partir de comparacions i escollir la que podria ser més beneficiosa pel nostre projecte.

L'autor analitza Netmiko, Paramiko, SSH2-python i Scrapli i descobreix que tan en un escenari amb un sol dispositiu com en un escenari amb múltiples dispositius, Netmiko sempre mostra temps d'execució més curts que la resta de llibreries [14]. Netmiko és la llibreria més eficient per les tasques d'automatització i es pot assolir que és la més beneficiosa en entorns on el rendiment i l'eficiència són crítics [14].

No obstant, el llicenciat Jean Carlos no s'atura i segueix mencionant altres fets pels quals escollir la llibreria en qüestió és una bona opció. I és que Netmiko ofereix facilitat per establir connexions SSH, fet fonamental per l'execució d'scripts d'automatització i gestió de configuracions [14]. La llibreria també simplifica la recuperació i execució de comandes, ja que permet recuperar dades de sortida de configuracions i executar diverses comandes de manera programada, facilitant tasques rutinàries i complexes [14]. Altres fets a destacar són l'automatització de solucions a problemes i la versatilitat en plataformes i dispositius [14]. I per últim, l'autor destaca que Netmiko permet l'admissió de múltiples tipus de connexions, ja que a part de les connexions SSH, Netmiko admet connexions Telnet, connexions en sèrie i Secure Copy [14].

Finalment, a part de reflectir l'efectivitat i adaptabilitat de Netmiko como una eina valuosa per l'automatització a la gestió de xarxes, l'autor també conclou que l'automatització és beneficiosa i crucial per la gestió eficient de les xarxes modernes [14].

A pesar de tot això i en última instància, s'ha volgut seguir investigant i descobrir si existeixen punts febles que trontollin l'automatització de xarxes. Resulta que aquesta no és del tot perfecta i s'ha observat que si es vol implementar correctament s'han de tenir en compte algunes qüestions en concret:

- **Inversió Inicial:** Implementar l'automatització requereix una inversió inicial en software, eines e infraestructura [15]. Tot i oferir majors beneficis a llarg termini, la inversió inicial pot ser considerable per moltes organitzacions.
- **Necessitat de Personal Qualificat:** L'automatització de xarxes requereix habilitats especialitzades en programació, scripting i protocols de xarxa. Les organitzacions poden necessitar invertir en programes de capacitació i desenvolupament per construir aquestes habilitats, el que representa un cost addicional i reptes en la contractació [15].
- **Augment del Risc de Bretxes de Seguretat:** Si l'automatització no s'implementa correctament, pot augmentar el risc de bretxes de seguretat. Els errors en la configuració de les eines d'automatització o la falta d'actualitzacions i manteniment adequats poden exposar la xarxa a vulnerabilitats [15].

2.3. Necessitats d'informació

Per poder comprendre de manera eficaç el projecte i el seu funcionament, es crucial tenir un coneixement detallat d'algunes de les eines i tecnologies que el conformen. A continuació, es descriuran les eines fonamentals que constitueixen el nucli del projecte, destacant la seves funcionalitats, utilitats i com s'interconnecten per formar un solució cohesiva i robusta que respongui als reptes actuals de la gestió de xarxes.

2.3.1. Emuladors de xarxa

En aquest projecte, es pretén utilitzar una eina que faci possible simular una xarxa amb totes les característiques per posar-la a prova amb el nostre projecte d'automatització. En un primer instant, la opció clara és GNS3.

GNS3 es una eina de simulació de xarxa que permet als seus usuaris dissenyar i configurar topologies de xarxa complexes [16]. GNS3 va ser creada l'any 2007 per l'enginyer de xarxes Jeremy Grossman. Tenia com a objectiu oferir una eina flexible i potent que ajudés a la comunitat dels administradors i enginyers de xarxes a simular entorns complexos per poder practicar i experimentar.

L'eina brinda la capacitat de simular una varietat de dispositius de diferents proveïdors utilitzant imatges de software reals [16]. D'aquesta manera, els usuaris poden practicar i perfeccionar les seves habilitats de configuració i solució de problemes en un entorn controlat i sense riscos. Avui en dia, s'utilitza àmpliament per la formació en xarxes, proves de concepte i validació de disseny abans de la implementació real.

No obstant, també tenim present altres alternatives que estan al mercat. Una d'aquestes alternatives es *Cisco Packet Tracer*. Aquesta eina desenvolupada per Cisco és una eina de simulació de xarxa més accessible i orientada a l'educació i és ideal per tots aquells que acaben de començar en el món de les xarxes [17]. No obstant, té limitacions en termes de dispositius i complexitat d'escenaris. Una altra opció podria ser VIRL, també de CISCO. És una opció més robusta que permet la simulació de xarxes més complexes i realistes però amb un cost associat i una corba d'aprenentatge més pronunciada [18]. Per últim, també cal mencionar EVE-NG que és una plataforma potent i flexible que suporta múltiples imatges de diferents proveïdors, sent una excel·lent opció per de prova complexos, tot i que pot requerir d'una configuració més detallada i recursos de hardware substancials [19].

Tot i així, s'ha escollit GNS3. I l'aspecte que ha estat clau en la elecció ha estat el fet de poder simular xarxes complexes amb varietat de dispositius de diferents proveïdors i ser una eina gratuïta i de codi obert, fet que la fa accessible i personalitzable per una àmplia

gama de necessitats. Un altre aspecte a remarcar és el fet de tenir una àmplia comunitat d'usuaris i de suport.

2.3.2. Llenguatges i Frameworks de programació

Python

Python és un llenguatge de programació d'alt nivell, versàtil i àmpliament adoptat i conegut per la seva sintaxis clara i entenedora [20]. En el context de l'automatització de xarxes, Python és de gran utilitat per la seva àmplia gama de biblioteques i marcs de treball que faciliten la interacció amb dispositius i sistemes de xarxes [21].

Per aquest projecte, Python és de gran utilitat ja que servirà per escriure scripts que automatitzaran les tasques de configuració, monitorització i gestió de xarxes. A més a més, Python té una gran integració amb APIs i biblioteques especialitzades que permet una àmplia personalització i expansió de les capacitats d'automatització, fent d'aquesta manera que Python sigui una elecció robusta i escalable per projectes de xarxes complexos.

Flask

Flask és un micro-framework per aplicacions web a Python [22]. És conegut per ser lleuger i fàcil d'utilitzar [22]. El seu disseny senzill i extensible el fan converteixen en una excel·lent opció per projecte de petita a mitjana escala. En un projecte on sigui present l'automatització de xarxes, Flask pot servir com el nucli d'una interfície web, permetent als usuaris interactuar amb eines d'automatització i visualitzar dades en temps real. La seva compatibilitat amb altres extensions de Python permet una integració sense problemes amb altres eines i llibreries, fet que el fa ideal per projectes escalables.

No obstant, cal tenir en compte que tot i que Flask sigui poderós i flexible, es tracta d'un micro-framework, el que significa que ve amb moltes menys funcionalitats incorporades en comparació amb solucions més grans com Django. Això potser un avantatge o un desavantatge depenent de les necessitats específiques del projecte. És possible que per projecte més grans calgui complementar Flask amb altres frameworks per obtenir les funcionalitats desitjades.

Per aquest projecte, Flask serà utilitzat per desenvolupar una interfície d'usuari web a través de la qual els administradors puguin interactuar amb la xarxa, visualitzar l'estat de la xarxa i controlar algunes configuracions.

2.3.3. Eines d'Automatització de Xarxes

Netmiko

Netmiko és de una las llibreries més utilitzades de Python ja que s'especialitza en la connexió entre dispositius a través del protocol SSH [23]. Aquesta llibreria simplifica el procés d'scripting per configurar dispositius de xarxa, obtenir informació i executar comandes automatitzades en una varietat d'equips de diferents fabricants [23].

Netmiko es torna una eina valuosa en degut a que ostenta una gran capacitat per controlar connexions múltiples i paral·leles, el seu suport extensiu per diferents dispositius i la seva interfície relativament senzilla que redueix significativament la corba d'aprenentatge pels enginyers o els desenvolupadors.

En aquesta plataforma, Netmiko serà utilitzat per automatitzar la configuració, el manteniment de dispositiu i recopilar dades per monitoritzar o analitzar. Cal tenir en compte que la seva flexibilitat y potencia la converteixen en la llibreria adequada per una gran quantitat de tasques d'automatització, des de les més simples fins les més complexes, adaptant-se a les necessitats i l'escala de la infraestructura de xarxa que es tingui. La seva integració amb altres eines de Python com NAPALM o Ansible, amplia encara més la seva utilitat, permetent crear solucions completes i altament customitzables. Gràcies a Netmiko es poden escriure scripts que no només duguin a terme tasques específiques sinó que també responguin dinàmicament als resultats de les comandes realitzades. D'aquesta manera s'obté un nivell de control i automatització que millora significativament l'eficiència, la precisió i la capacitat de resposta de gestió de les xarxes.

Una altra raó significativa per escollir Netmiko és la seva extensa i activa comunitat de desenvolupadors, que proporciona una abundant documentació i suport. Això facilita l'aprenentatge i la implementació efectiva de la llibreria en qualsevol projecte.

NAPALM

NAPALM és una llibreria d'automatització per Python dissenyada per simplificar i unificar la gestió d'equips de xarxes de diferents fabricants [24]. Proporciona una interfície comú per la simplificar i unificar la gestió d'equips de diferents fabricants. Això permet als administradors de xarxes interactuar de manera coherent amb equips de diferents proveïdors. Amb NAPALM, es pot recuperar informació com configuracions, taules d'enrutament i estadístiques d'interfícies de manera estandarditzada, independentment del sistema operatiu del dispositiu.

NAPALM ajudarà a dur a terme la implementació de polítiques de xarxes coherents i eficients, facilitant tasques com el desplegament de configuracions i la recol·lecció i comparació d'estats previs i posteriors als canvis fets i el diagnosi de problemes a la xarxa. El seu disseny modular i el suport per múltiples fabricants fan d'aquesta llibreria una prou valuosa en entorns de xarxa diversos, on es busca reduir la complexitat operativa i augmentar l'eficiència mitjançant l'automatització.

D'igual manera que Netmiko, NAPALM és un bona elecció també perquè disposa d'una gran comunitat activa que garanteix una evolució constant del Framework, amb actualitzacions regulars i un gran suport per enfrontar qualsevol tipus de repte.

3. Objectius i abast

L'abast del projecte comprèn el disseny i el desenvolupament d'una eina integral per la configuració, gestió o monitorització automatitzada de dispositius de xarxa.

El primer que es durà a terme és l'elaboració dels requeriments funcionals i tecnològics per poder iniciar el projecte. Seguidament, es farà un estudi de viabilitat que analitzarà la viabilitat tècnica, econòmica i operativa de la solució proposada. Això servirà per assegurar que l'eina d'automatització no és tan sols realitzable sinó també beneficiosa i sostenible al llarg temps.

Més tard, es dissenyarà i es desenvoluparà en detall el backend que serà essencial per poder realitzar les tasques de processament , gestió i automatització de les tasques de xarxa. Aquest serà el component principal de l'eina, integrant diverses tecnologies com Netmiko i NAPALM per poder interactuar amb dispositius de la xarxa. Aquest pas serà clau per l'èxit final del projecte. També es definirà amb precisió els dispositius i fabricants compatibles, les tasques compatibles i la naturalesa de la integració amb els sistemes TI existents.

Per altra banda, el projecte també inclourà el disseny i desenvolupament d'una interfície web basada en Flask que permetrà la interacció amb l'eina d'automatització. Es crearà documentació per poder configurar i utilitzar l'eina correctament i finalment, s'avaluarà el correcte funcionament del projecte a partir de l'anàlisi de l'eficiència operativa, la freqüència d'errors i la satisfacció d'un usuari de proves.

Un cop descrit l'abast del treball, es mencionaran els objectius del projecte:

Objectius del Client:

- Millorar l'eficiència operativa per tasques de xarxes.
- Reduir el temps de resposta per la gestió de xarxes.
- Minimitzar el errors humans i els costs associats.
- Millorar la monitorització de la xarxa.

- **Objectius del Producte:**
  - Desenvolupar una solució d'automatització de xarxes que sigui adaptable, segura i fàcil d'integrar en diferents entorns de xarxes.
- **Públic potencial:**
  - Dirigit a administrador de xarxes i organitzacions que busquen optimitzar i modernitzar la gestió i el manteniment de les seves infraestructures TI.

4. Metodologia

Per al desenvolupament d'aquest projecte, s'ha optat per adoptar la metodologia Scrum, una derivació pràctica i altament efectiva de la filosofia Agile. Scrum no és només un marc de treball, sinó una estratègia detallada que permet ajudar els equips a estructurar i gestionar la seva feina a través d'un conjunt de valors, principis i pràctiques [25]. Aquesta metodologia es caracteritza per la seva naturalesa iterativa i incremental, dividint el desenvolupament sprints, que solen tenir una durada d'entre dues i quatre setmanes.

Cada sprint comença amb una planificació minuciosa on es defineixen els objectius i les tasques a realitzar. Aquest enfocament fa possible revisar i ajustar el treball amb una freqüència regular, facilitant així la incorporació de canvis o nous requisits que puguin sorgir durant el procés de desenvolupament. Aquesta flexibilitat és vital, ja que reconeix que els projectes de desenvolupament sovint s'enfronten a canvis imprevistos o a l'aparició de noves oportunitats que requereixen una resposta ràpida i adaptativa.

Un dels avantatges més rellevants de Scrum és la seva capacitat per a la detecció i correcció precoç d'errors. Mitjançant revisions al final de cada sprint, serem capaços d'identificar ràpidament qualsevol desviació o problema i prendre les mesures correctives necessàries. Això no només millora la qualitat del producte final sinó que també garanteix un progrés continu i eficient cap a la consecució dels objectius del projecte.

El procés de cada sprint es sol dividir en cinc fases. No obstant com el desenvolupament no es durà a terme en un equip, s'ometrà una fase.

1. **Planificació de l'sprint:** S'estableixen els objectius clars i específics per l'sprint.
2. **Desenvolupament:** Implementar les funcionalitats amb la programació necessària.
3. **Proves i revisions:** Es prova el codi i les funcionalitats desenvolupades per assegurar-se de que funcionin segons les previsions fetes.
4. **Reflexió i correccions:** Analitzar el que ha funcionat bé i el que es pot millorar, ajustant el pla pel següent sprint en conseqüència.

Tot i haver escollit Scrum com la filosofia pertinent que definirà com es durà a terme el nostre projecte, s'ha tingut en compte també Kanban. Kanban és una bona opció degut a que el seu enfocament en la visualització del treball i la flexibilitat en la gestió de tasques permet un seguiment clar del progrés i facilita l'adaptabilitat a canvis o nous requeriments.

Criteri	Scrum	Kanban
Iteracions	Sprints fixes de duració definida.	Flux continu sense sprints fixes.
Rols	Rols definits(Scrum Master, Product Owner, etc,) No és rellevant pel nostre projecte.	No hi ha rols definits. Més flexibilitat.
Canvis	Durant l'sprint els canvis són limitats.	Canvis possibles en qualsevol moment.
Medició	Velocitat de l'sprint.	Temps de cicle.
Visualització	Taula d'Scrum amb sprints	Taula de Kanban amb fluxos continus.

Taula 4.1. - Metodologies

Finalment s'ha seleccionat Scrum perquè s'apropa més al tipus de treball que es vol dur a terme. Scrum disposa d'una estructura iterativa que promou cicles regulars de treball i revisió. L'enfocament d'entregues funcionals al final de cada sprint i la planificació a curt termini que permet una adaptació i revaluació constant crea l'entorn perfecte per poder lliurar fases concretes del treball al professor i a l'hora rebre feedback per poder corregir tots els errors d'aquesta fase sense que l'alumne o el professor col·lapsin per la càrrega de treball.

4.1. Iteracions

A continuació, es fa una aproximació del total de sprints que pot tenir el projecte.

- **Sprint 0: Escriptura de l'avantprojecte.** Estudi previ, objectius, estudi de viabilitat i recopilació de requisits.
- **Sprint 1: Disseny de l'arquitectura.** Esquematització de l'arquitectura del sistema i disseny inicial de la base de dades.
- **Sprint 2: Configuració de l'entorn de desenvolupament.** Preparació de l'entorn de desenvolupament i eines necessàries.
- **Sprint 3: Desenvolupament del backend – Primera fase.** Creació d'scripts d'automatització bàsics i funcionalitats del backend.
- **Sprint 4: Desenvolupament del frontend – Primera fase.** Disseny de la interfície d'usuari i desenvolupament del frontend.
- **Sprint 5: Integració del backend i el frontend.** Connectar backend amb frontend i assegurar la comunicació fluida entre ells.
- **Sprint 6: Escriptura de la memòria intermèdia.**
- **Sprint 7: Desevolupament del backend – Segona fase.** Ampliació de funcionalitats del backend i scripts d'automatització.
- **Sprint 8: Desenvolupament del frontend – Segona fase.** Millores i refinament de la interfície d'usuari.
- **Sprint 9: Proves inicials i depuració.** Realització de proves unitàries i de integració i correcció d'errors.
- **Sprint 10: Proves d'usuari.** Proves d'usuari i recol·lecció de feedback per implementar millores.
- **Sprint 11: Refinament final i preparació pel llançament.** Implementació a partir del feedback rebut i millores finals pel llançament.
- **Sprint 12: Escriptura de la memòria final.**

5. Requeriments

5.1. Requeriments funcionals

En un primer lloc, cal mencionar tots aquells requeriments que definiran el que s'espera que sigui el comportament òptim de l'aplicació des del punt de vista de l'usuari. Aquests requeriments són imprescindibles per guiar el desenvolupament i assegurar que el producte final compleixi amb les necessitats i les expectatives establertes.

Seguidament es llistaran els requeriments que es procuraran assolir per l'aplicació final:

- Visualitzar l'estat de la xarxa i els dispositius en temps real.
- Interfície d'usuari intuïtiva i accessible.
- Configurar automatització pels dispositius de xarxa.
- Documentació completa i accessible.
- Alta escalabilitat i absència d'errors crítics.
- Compatibilitat amb dispositius de xarxa de diferents fabricants.

5.2. Requeriments tecnològics

En segona instància, es mencionen tots aquells requeriments que definiran les especificacions i condicions tècniques necessàries per dissenyar, desenvolupar, implementar i mantenir un sistema o aplicació.

- Imatges dels sistemes operatius pels dispositius de la xarxa que es simularà a GNS3.
- Servidor per hostejar l'aplicació. (Opcional)
- Python pel desenvolupament backend i els scripts d'automatització.
- HTML, CSS i JavaScript pel frontend.
- Llibreries d'automatització Netmiko, NAPALM.
- Framework web Flask.
- Plataforma de simulació de xarxes GNS3.
- Base de dades per emmagatzemar configuracions i dades de la xarxa.
- Entorn de desenvolupament integrat com PyCharm o Visual Studio.
- Sistema de control de versions com Git.

6. Desenvolupament

El següent apartat se centra en la descripció i explicació dels processos més importants que han tingut lloc durant l'etapa de desenvolupament de l'aplicació. En aquesta seran recollits els aspectes clau de la implementació, les decisions de disseny més rellevants, així com els reptes i les solucions adoptades per superar-los.

6.1. Backend

El disseny i desenvolupament d'un backend robust i eficient per a l'aplicació ha estat un dels primers passos crucials en el seu desenvolupament. Aquest backend no només tenia com a objectiu gestionar les peticions dels usuaris de manera eficient i segura, sinó també oferir una arquitectura escalable que pogués adaptar-se als canvis i al creixement previst de l'aplicació. En el backend es duran a terme diversos processos a l'hora per poder implementar l'automatització de xarxes. Per tant, és crític mantenir una estructura escalable i mantenible en el temps que permeti gestionar i implementar tots els mecanismes desitjats.

Per poder dissenyar i crear una arquitectura de tals característiques de manera senzilla i amb Python, s'ha escollit Flask. Aquesta és una de les millors eleccions pel backend a causa de la seva simplicitat i flexibilitat, permetent un desenvolupament ràpid i àgil. I a la vegada, la seva naturalesa de microservei facilita l'escalabilitat i el manteniment que tant es desitja. D'aquesta manera s'ha estructurat el backend en els següents apartats: API, models i serveis.

6.1.1. API

Per disposar d'una millor gestió del codi i facilitat l'enteniment i escalabilitat d'aquesta, s'ha optat per una organització modular.

- **__init__.py:** Aquest fitxer utilitza la funcionalitat de les Blueprints de Flask per organitzar l'aplicació en components lògics. Es creen tres Blueprints: "device_bp", "host_bp" i task_scheduler_bp, que actuen com contenidors per agrupar rutes.

```
from flask import Blueprint

device_bp = Blueprint( name: 'device_bp', __name__)
host_bp = Blueprint( name: 'host_bp', __name__)
task_scheduler_bp = Blueprint( name: 'task_scheduler_bp', __name__)

from . import DeviceRoutes
from . import TaskSchedulerRoutes
from . import HostRoutes
```

Fig. 6.1. Declaració de Blueprints

- **DeviceRoutes.py:** Aquest arxiu conté totes les rutes relacionades amb operacions amb dispositius. Utilitza la Blueprint "device_bp" creada en l'arxiu anterior. Les crides que es poden dur a terme són les següents:
    - */devices/<int:device_id> (GET).* Serveix per obtenir un dispositiu per ID. S'obté a partir de la taula Device.
    - */devices/ (GET)*. Serveix per obtenir tots els dispositius. S'obté a partir de la taula Device.
    - */devices/status/<int:device_id> (GET).* Serveix per obtenir tots els estats guardats que formen part del dispositiu amb l'ID entrat. S'obté a partir de la taula DeviceStatus.
    - */devices/status/ (GET)* Serveix per obtenir tots els estats de tots els dispositius. S'obté a partir de la taula DeviceStatus.
    - */devices/arp_table/<int:device_id> (GET).* Serveix per obtenir les dades de les interfícies que formen part del dispositiu amb l'ID entrat. S'obté a partir de la taula DeviceArpEntry.
    - */devices/interfases/<int:device_id> (GET).* Serveix per obtenir les dades dels comptadors de les interfícies que formen part del dispositiu amb l'ID entrat. S'obté a partir de la taula DeviceInterface.
    - */devices/bpg_neighbors/<int:device_id> (GET).* Serveix per obtenir les dades de la taula BGP que forma part del dispositiu amb l'ID entrat. S'obté a partir de la taula DeviceBgpNeighbor.
    - */devices/cli/<int:device_id> (POST).* Serveix per enviar una o diverses comandes al dispositiu amb l'ID entrat. Per dur a terme aquesta funció, s'utilitza la llibreria Netmiko.

      El body de la petició ha de contenir l'item commands que contindrà un array amb les comandes que es volen executar. Exemple: comands: ['sh ip int brief', 'sh version']. Si la funció s'executa correctament, es retornaran els resultats de les comandes.
    - */devices/backup/<string:backup_id> (GET).* Serveix per obtenir la configuració que conté l'ID entrat. S'obté a partir de la taula DeviceConfig.
    - */devices/backup/<int:device_id> (GET).* Serveix per obtenir totes les configuracions guardades en el temps que formen part del dispositiu amb l'ID entrat. S'obté a partir de la taula DeviceConfig.
    - */devices/backup/<int:device_id> (POST).* Serveix per guardar l'última configuració del dispositiu que conté l'ID entrat. Per guardar la configuració a la taula DeviceConfig s'utilitza la llibreria Napalm, que a partir de la funció get_config(), permet obtenir l'última configuració del dispositiu.

- */devices/backup/<int:device_id> (DELETE).* Serveix per eliminar les backups entrades en el body de la petició i que formen part del dispositiu amb l'ID entrat. El body de la petició ha de contenir l'item backups que contindrà un array amb els IDs de les backups que es volen eliminar. Exemple: backups: ['cc0d49cd-56b1-4b31-be24-d40789aa7931'].
- */devices/restore/<int:device_id>/<string:backup_id> (POST).* Serveix per restaurar la configuració amb el segon ID entrat en el dispositiu amb el primer ID entrat. Per poder dur aquesta funció s'utilitza la llibreria netmiko, que a partir de la funció send_config_set(), permet configurar el dispositiu amb la configuració entrada.
- */compare/<int:device_id>/<string:backup_id> (GET).* Serveix per comparar la configuració actual del dispositiu amb el primer ID entrat amb la configuració amb el segon ID entrat. Per obtenir la configuració actual s'utilitza la llibreria NAPALM, que a partir de la funció get_config(), permet obtenir l'última configuració del dispositiu. Seguidament, la funció compara les dues configuracions i retorna les diferències en format JSON.

- **HostRoutes.py:** Aquest arxiu conté totes les rutes relacionades amb operacions amb hosts. Utilitza la Blueprint "host_bp" creada en l'arxiu anterior. Les crides que es poden dur a terme són les següents:
  - */hosts/<int:device_id> (GET).* Serveix per obtenir un host per ID. S'obté a partir de la taula Host.
  - */devices/ (GET)*. Serveix per obtenir tots els hosts. S'obté a partir de la taula Host.

- **TaskSchedulerRoutes.py:** Aquest arxiu conté totes les rutes relacionades amb operacions del programador de tasques. Utilitza la Blueprint "task_scheduler_bp" creada en l'arxiu anterior. Les crides que es poden dur a terme són les següents:
  - */task_scheduler/<string:task_id> (GET).* Serveix per obtenir la tasca amb l'ID entrat. S'obté a partir de la taula DeviceTask.
  - */task_scheduler (GET).* Serveix per obtenir totes les tasques. S'obté a partir de la taula DeviceTask.
  - */task_scheduler (POST).* Serveix per programar una tasca. Per poder dur a terme aquesta funció s'utilitza la llibreria APScheduler. La funció permet programar tasques perquè s'executin comandes en dispositius seleccionats a una hora determinada. A més a més, la funció permet ajustar un interval diari i setmanal perquè la tasca es pugui repetir en

intervals personalitzats. De totes maneres, aquests intervals són opcionals. Per introduir les dades a la funció cal passar certes dades a través del body de la petició. Els ítems que caldrà introduir són els següents:

- name. Nom de la tasca.
- commands. Array amb les comandes.
- time. Hora d'execució.
- repeat. Booleà per l'interval diari.
- repeatInterval. Interval diari. 30 minuts, una hora, dues hores, tres hores o personalitzat.
- customInterval. Interval diari personalitzat. (Minuts).
- weekRepeat. Booleà per l'interval setmanal.
- weekRepeatInterval. Interval setmanal. Diari, dilluns a divendres o personalitzat.
- customDays. Interval setmanal personalitzat. Array amb dies de la setmana seleccionats.
- selectedDevices. Dispositius seleccionats per programar les tasques.

Exemple. {"name":"Task","commands":"sh ip int brief, sh version","time":"10:00","repeat":true,"repeatInterval":"60","customInterval":"","weekRepeat":true,"weekRepeatInterval":"Daily","customDays":[],"selectedDevices":[1]}

- /task_scheduler/pause/<string:task_id> Serveix per aturar la tasca amb l'ID entrat. Aquesta funció es duu a terme gràcies a la llibreria APScheduler. Amb el scheduler configurat anteriorment i la funció pause_job() s'atura la programació de la tasca i no es torna a repetir fins que es reprengui.
- /task_scheduler/resume/<string:task_id> Serveix per reprendre la tasca amb l'ID entrat. Aquesta funció es duu a terme gràcies a la llibreria APScheduler. Amb el scheduler configurat anteriorment i la funció resume_job() es reprén la programació de la tasca en el cas que estigués aturada.
- /task_scheduler/resume/<string:task_id> Serveix per aturar per sempre la tasca amb l'ID entrat. Aquesta funció es duu a terme gràcies a la llibreria

APScheduler. Amb el scheduler configurat anteriorment i la funció remove_job() s'atura la programació de la tasca per sempre.

6.1.2. Models

Dins del desenvolupament del backend, un altre fonamental és la definició dels models. Aquests representen l'estructura de les dades amb les quals l'aplicació treballarà i la relació entre aquestes dins de la base de dades. Els models que formen part d'aquest projecte són els següents:

- **Device:** El model "Device" és el que s'encarrega de representar aquells dispositius amb els quals durem a terme tasques d'automatització de xarxa.

```python
class Device(db.Model):

    __tablename__ = "device"

    id = db.Column(db.Integer, primary_key=True)
    ip_address = db.Column(db.String(15), nullable=False)
    username = db.Column(db.String(50), nullable=False)
    password = db.Column(db.String(100), nullable=False)

    domain_name = db.Column(db.String(100), nullable=True)
    mac_address = db.Column(db.String(100), nullable=True)
    ssh_port = db.Column(db.Integer, default=22, nullable=True)
    os = db.Column(db.String(50), nullable=True)

    fqdn = db.Column(db.String(100), nullable=True)
    hostname = db.Column(db.String(50), nullable=True)
    model = db.Column(db.String(100), nullable=True)
    os_version = db.Column(db.String(200), nullable=True)
    device_type = db.Column(db.String(50), nullable=True)
    serial_number = db.Column(db.String(100), nullable=True)
    vendor = db.Column(db.String(50), nullable=True)
    uptime = db.Column(db.Float, nullable=True)

    current_status = db.Column(db.Boolean, default=False, nullable=False)
    cpu = db.Column(db.String(50), default=0.0, nullable=False)
    memory = db.Column(db.Float, default=0.0, nullable=False)
    memory_percentage = db.Column(db.Float, default=0.0, nullable=False)

    response_time = db.Column(db.Float, default=0.0, nullable=False)
    last_checked = db.Column(db.DateTime, default=datetime.utcnow, nullable=False)

    current_configuration = db.Column(db.String(), nullable=True)
```

Fig. 6.2. Model Device

Per la utilització de llibreries com NAPALM o Netmiko calen certes variables per poder establir connexió i executar els processos en qüestió. Algunes de les més rellevants per poder establir connexions SSH són l'adreça IP, l'usuari i la

contrasenya. Gràcies al model “Device” aquestes seran definides i guardades a l’aplicació.

Els paràmetres que s’han de monitorar i relacionar amb el model també hauran de ser definits. És aquí on apareixen les variables d’estat *current status, cpu i memory i memory_percentage. Current_status* indica si el dispositiu està encès o apagat. Les altres variables donen percentatges d’ús del processador i la memòria RAM.

A més a més, es guarden les variables *response_time* i *last_checked.* La primera permet conèixer el temps de resposta de l’actual resposta del dispositiu i la segona ens permet saber quan va ser l’última resposta del dispositiu.

Altres variables menys importants però que es monitoren cada cert interval de temps són els “facts” de cada dispositiu. Els “facts” són dades detallades sobre un dispositiu que descriuen característiques físiques i tècniques d’aquest. Alguns dels que es monitoren son *fqdn, hostname, model, os_version, serial_number, vendor i uptime.*

Per últim i menys important, cal mencionar, que al ser aquest model l’estructura central de l’arquitectura de la base de dades se li ha calgut establir les relacions d’un a molts amb la resta de models involucrats en aquest model.

```
device_statuses = db.relationship('DeviceStatus', backref='device', lazy=True)
arp_entries = db.relationship('DeviceArpEntry', backref='device', lazy=True)
interfaces = db.relationship('DeviceInterface', backref='device', lazy=True)
bgp_neighbors = db.relationship('DeviceBgpNeighbor', backref='device', lazy=True)
device_configs = db.relationship('DeviceConfig', backref='device', lazy=True)
tasks = db.relationship('DeviceTask', backref='device', lazy=True)
```

Fig. 6.3. Declaració de relacions del model Device

- **DeviceArpEntry:** El model “DeviceArpEntry” defineix cada fila de la taula ARP. Aquest model s’identifica a partir d’un identificador únic UUID i una clau forana que correspon al dispositiu relacionat amb l’entrada ARP.

```python
class DeviceArpEntry(db.Model):
    __tablename__ = "device_arp_entry"

    id = db.Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()), unique=True)
    device_id = db.Column(db.Integer, db.ForeignKey('device.id'), nullable=False)

    ip_address = db.Column(db.String(45), nullable=False)
    mac_address = db.Column(db.String(17), nullable=False)
    interface = db.Column(db.String(50), nullable=False)

    Albert Castaño i Bret
    def to_dict(self):
        return {c.name: getattr(self, c.name) for c in self.__table__.columns}
```

Fig. 6.4. Model DeviceArpEntry

Les variables que defineixen cada entrada són *interface, mac_addres i interface*. *Interface* és el nom de la interfície i les altres dos son l'adreça IP i l'adreça MAC que té la interfície.

- **DeviceBgpNeighbor:** El model "DeviceBgpNeighbor" defineix cada fila de la taula BGP del dispositiu. Aquest model s'identifica a partir d'un identificador únic UUID i una clau forana que correspon al dispositiu relacionat amb l'entrada BGP.

```python
class DeviceBgpNeighbor(db.Model):
    __tablename__ = "device_bgp_neighbor"

    id = db.Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()), unique=True)
    device_id = db.Column(db.Integer, db.ForeignKey('device.id'), nullable=False)

    neighbor_address = db.Column(db.String(50), nullable=False)
    local_as = db.Column(db.String(50), nullable=False)
    remote_as = db.Column(db.String(50), nullable=False)
    remote_id = db.Column(db.String(50), nullable=False)

    is_up = db.Column(db.Boolean, default=False, nullable=False)
    is_enabled = db.Column(db.Boolean, default=False, nullable=False)
    uptime = db.Column(db.Integer, default=0, nullable=False)

    sent_prefixes_ipv4 = db.Column(db.Integer, default=0, nullable=False)
    accepted_prefixes_ipv4 = db.Column(db.Integer, default=0, nullable=False)
    received_prefixes_ipv4 = db.Column(db.Integer, default=0, nullable=False)
    sent_prefixes_ipv6 = db.Column(db.Integer, default=0, nullable=False)
    accepted_prefixes_ipv6 = db.Column(db.Integer, default=0, nullable=False)
    received_prefixes_ipv6 = db.Column(db.Integer, default=0, nullable=False)

    Albert Castaño i Bret
    def to_dict(self):
        return {c.name: getattr(self, c.name) for c in self.__table__.columns}
```

Fig. 6.5. Model DeviceBgpNeigbor

En primer lloc, es defineix *neighbor_address* que és l'adreça del dispositiu veí que identifica aquesta entrada. Seguidament, tenim *local_as* que determina el número de sistema autònom local i *remote_as* que pertany al número de sistema autònom

del veí remot. L'últim identificador és *remote_id* que és un identificador únic de l'encaminador remot en la sessió BGP.

Les següents variables a mencionar són les variables d'estat. Les primeres a mencionar són *is_up, is_enabled* i *uptime*. *Is_up* indica si la sessió BGP amb el vei es troba actualment activa, mentre que *is_enabled* indica si la configuració de la sessió BGP amb el veí està habilitada en el dispositiu. *Uptime* mostra el temps, en segons, que la sessió BGP ha estat activa de manera contínua.

Finalment, cal mencionar les variables que permeten monitorar i controlar els dispositius veïns amb major profunditat:

- *sent_prefixes_ipv4:* Nombre de prefixos IPv4 enviats per l'encaminador al veí BGP. Monitorar aquesta variable ajuda a entendre quantes rutes IPv4 s'estan anunciant als veïns BGP.
- *accepted_prefixes_ipv4.* Nombre de prefixos IPv4 acceptats per l'encaminador des del veí BGP després d'aplicar polítiques de filtratge i rutes. Controlar aquesta variable assegura que tan sols les rutes vàlides i desitjades s'acceptin en la taula d'encaminament.
- *received_prefixes_ipv4.* Nombre de prefixos IPv4 rebuts des del veï BGP abans d'aplicar qualsevol política de filtratge. Monitorar aquesta variable ajuda a evaluar la quantitat total de rutes IPv4 que es reben del veï.
- *sent_prefixes_ipv6:* Nombre de prefixos IPv6 enviats pel router al veï BGP. Monitoritzar aquesta variable ajuda a entendre quantes rutes IPv4 s'estan anunciant als veïns BGP.
- *accepted_prefixes_ipv6.* Nombre de prefixos IPv6 acceptats pel router des del veï BGP després d'aplicar polítiques de filtratge i rutes. Controlar aquesta variable assegura que tan sols les rutes vàlides i desitjades s'acceptin en la taula d'encaminament.
- *received_prefixes_ipv6.* Nombre de prefixos IPv4 rebuts des del veï BGP abans d'aplicar qualsevol política de filtratge. Monitorar aquesta variable ajuda a avaluar la quantitat total de rutes IPv4 que es reben del veí.

- **DeviceConfig**: El model "DeviceConfig" defineix una configuració amb una marca de temps d'un dispositiu. Aquest model s'identifica a partir d'un identificador únic UUID i una clau forana que correspon al dispositiu relacionat amb la configuració.

```python
class DeviceConfig(db.Model):
    __tablename__ = "device_config"

    id = db.Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()), unique=True)
    device_id = db.Column(db.Integer, db.ForeignKey('device.id'), nullable=False)

    config = db.Column(db.Text, nullable=False)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow, nullable=False)

    Albert Castaño i Bret
    def to_dict(self):
        return {c.name: getattr(self, c.name) for c in self.__table__.columns}
```

Fig. 6.6. Model DeviceConfig

Aquest model tan sols disposa de dues variables més que són *config* i *timestamp.* Config serveix per guardar en una variable “string” tota la configuració del dispositiu i *timestamp* determina el moment en el temps en el qual aquesta configuració ha estat guardada.

- **DeviceInterface:** El model “DeviceInterface” defineix les variables relacionades amb les interfícies del dispositiu. Aquest model s’identifica a partir d’un identificador únic UUID i una clau forana que correspon al dispositiu relacionat amb les variables d’estat.

```python
class DeviceInterface(db.Model):
    __tablename__ = "device_interface"

    id = db.Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()), unique=True)
    device_id = db.Column(db.Integer, db.ForeignKey('device.id'), nullable=False)

    interface_name = db.Column(db.String(100), nullable=False)
    is_up = db.Column(db.Boolean, default=False, nullable=False)
    is_enabled = db.Column(db.Boolean, default=False, nullable=False)
    description = db.Column(db.Text, nullable=True)
    last_flapped = db.Column(db.Float, default=-1.0, nullable=False)
    speed = db.Column(db.Float, default=0.0, nullable=False)
    mac_address = db.Column(db.String(17), nullable=True)

    tx_errors = db.Column(db.BigInteger, default=0, nullable=False)
    rx_errors = db.Column(db.BigInteger, default=0, nullable=False)

    tx_discards = db.Column(db.BigInteger, default=0, nullable=False)
    rx_discards = db.Column(db.BigInteger, default=0, nullable=False)

    tx_octets = db.Column(db.BigInteger, default=0, nullable=False)
    rx_octets = db.Column(db.BigInteger, default=0, nullable=False)

    tx_unicast_packets = db.Column(db.BigInteger, default=0, nullable=False,)
    rx_unicast_packets = db.Column(db.BigInteger, default=0, nullable=False)

    Albert Castaño i Bret
    def to_dict(self):
        return {c.name: getattr(self, c.name) for c in self.__table__.columns}
```

Fig. 6.7. Model DeviceInterface

Monitorar les interfícies i els seus valors és útil per identificar i gestionar cada interfície de manera única. Permet detectar l'estat operatiu, configuracions i documentar característiques de cada interfície. Ajuda a identificar interrupcions, avaluar el rendiment en termes de velocitat, i resoldre problemes de xarxa mitjançant la identificació d'adreces MAC. A més a més, monitora la qualitat de transmissió i recepció de dades, i detecta pèrdues o descarts de paquets, essencial per mantenir l'estabilitat i eficiència de la xarxa. Les variables són les següents:

- *interface_name*: Nom de la interfície.
- *is_up*: Estat de la interfície. Indica si la interfície està operativa, essencial per detectar fallades o desconnexions.
- *is_enabled*: Mostra si la interfície està configurada per funcionar.
- *description*: Descripció de la interfície.
- *last_flapped*: Permet detectar la freqüència i moments d'inestabilitat o caigudes de la interfície.
- *speed*: Indica la capacitat de la interfície, important per avaluar si compleix amb els requisits d'ample de banda.
- *mac_address*: Adreça MAC de la interfície.
- *tx_errors*: Identifica problemes de qualitat en la transmissió de dades.
- *rx_errors*: Identifica problemes de qualitat en la recepció de dades.
- *tx_discards*: Ajuda a identificar pèrdues de dades i possibles colls d'ampolla o configuracions errònies.
- *rx_discards*: Ajuda a identificar pèrdues de dades i possibles colls d'ampolla o configuracions errònies.

- **DeviceStatus:** El model “DeviceStatus” defineix les variables d’estat amb una marca de temps d’un dispositiu. Aquest model s’identifica a partir d’un identificador únic UUID i una clau forana que correspon al dispositiu relacionat amb les variables d’estat.

```python
class DeviceStatus(db.Model):
    __tablename__ = "device_status"

    id = db.Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()), unique=True)
    device_id = db.Column(db.Integer, db.ForeignKey('device.id'), nullable=False)

    timestamp = db.Column(db.DateTime, default=datetime.utcnow(), nullable=False)
    status = db.Column(db.Boolean, default=False, nullable=False)
    cpu = db.Column(db.String(50), default=0.0, nullable=False)
    memory = db.Column(db.String(200), default=0.0, nullable=False)
    response_time = db.Column(db.Float, default=0.0, nullable=False)

    Albert Castaño i Bret
    def to_dict(self):
        return {c.name: getattr(self, c.name) for c in self.__table__.columns}
```

Fig. 6.8. Model DeviceStatus

Aquest model permet crear traces de temps de les diferents variables i poder observar si en algun moment del monitoratge res inusual ha tingut lloc. Les variables són les següents:

- *timestamp:* Marca de temps per identificar el moment en què es guarden les dades.
- *status:* Determina si el dispositiu està encès o apagat.
- *cpu:* Tant per cent d'ús del processador.
- *memory:* Tant per cent d'ús de la memòria RAM.
- *response_time:* Temps de resposta del dispositiu.

- **DeviceTask:** El model "DeviceTask" defineix una tasca programada d'un dispositiu. Aquest model s'identifica a partir d'un identificador únic UUID i una clau forana que correspon al dispositiu relacionat amb la tasca configurada.
  Aquest model representa la configuració d'una tasca programada per un dispositiu amb la llibreria APScheduler. Les variables que defineixen el model són les següents:
  - *name:* Nom de la tasca.
  - *commands:* Comandes que s'executaran en el dispositiu.
  - *execution_time:* Hora en la qual s'inicia la tasca.
  - *repeat_interval:* Interval diari en el qual es repeteix la tasca.
  - *days_of_week:* Dies de la setmana en els que es repeteix la tasca.
  - *results:* Outputs de les comandes.
  - *last_executime_time:* Última data en la que s'ha executat la data.
  - *is_started:* Booleà que determina si la tasca ha començat.
  - *is_finished:* Booleà que determina si la tasca ha acabat.

- *is_paused:* Booleà que determina si la tasca està aturada.

```python
class DeviceTask(db.Model):
    __tablename__ = "device_task"

    id = db.Column(String(36), primary_key=True, default=lambda: str(uuid.uuid4()), unique=True)
    device_id = db.Column(db.Integer, db.ForeignKey('device.id'), nullable=False)
    name = db.Column(db.String(50), default="Default", nullable=False)

    commands = db.Column(db.Text, default='', nullable=False)
    execution_time = db.Column(db.String(50), nullable=False)
    repeat_interval = db.Column(db.Integer, default=0, nullable=False)
    days_of_week = db.Column(db.String(50), default='(None)', nullable=False)

    results = db.Column(db.Text, default='', nullable=False)
    last_execution_time = db.Column(db.DateTime, nullable=True)
    is_started = db.Column(db.Boolean, default=False, nullable=False)
    is_finished = db.Column(db.Boolean, default=False, nullable=False)
    is_paused = db.Column(db.Boolean, default=False, nullable=False)

    Albert Castaño i Bret
    def to_dict(self):
        return {c.name: getattr(self, c.name) for c in self.__table__.columns}
```

Fig. 6.9. Model DeviceTask

- **Host:** El model “Host” és una representació per tots els dispositius que es detectin a la xarxa. Aquests no participen en tasques d’automatització i, per tant, no es monitoren. D’aquesta manera, aquest model no defineix tants paràmetres. Alguns d’aquests són: *name, ip_address, mac_address.*

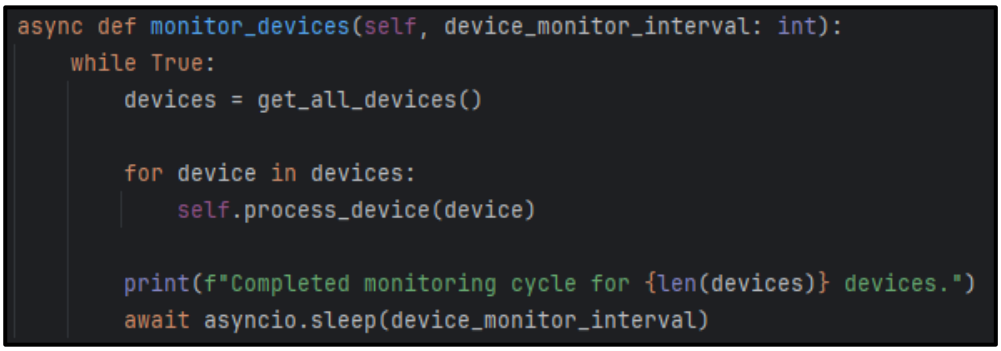
```python
class Host(db.Model):

    __tablename__ = "host"

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    ip_address = db.Column(db.String(15), nullable=False)
    mac_address = db.Column(db.String(100), nullable=False)
```

Fig. 6.10. Model Host

6.1.3. Serveis – Automatització de xarxa

Per tal de dur a terme el monitoratge i la gestió dels dispositius, i a la vegada no blocar l’API, cal executar els processos d’automatització de forma asíncrona i separada al fil principal de l’API. En cas de no ser així, no es podria aconseguir sincronitzar ambdós processos i el rendiment de l’aplicació es veuria molt afectat. D’aquesta manera, s’ha creat la classe “AsyncTaskManager”, una classe cridada en un fil secundari a l’iniciar l’aplicació. Aquesta defineix les següents funcions d’automatització:

- **async def monitor_devices(self, device_monitor_interval):** La funció monitor_devices és un bucle asincrònic que contínuament monitora dispositius de xarxa a intervals de temps específics. En cada iteració del bucle, obté una llista de tots els dispositius mitjançant la funció get_all_devices i, per a cada dispositiu, crida a process_device per obtenir i processar les seves dades. Un cop ha acabat de monitorar tots els dispositius, espera el temps definit per device_monitor_interval abans de començar la següent iteració.



```python
async def monitor_devices(self, device_monitor_interval: int):
    while True:
        devices = get_all_devices()

        for device in devices:
            self.process_device(device)

        print(f"Completed monitoring cycle for {len(devices)} devices.")
        await asyncio.sleep(device_monitor_interval)
```

Fig. 6.11. Funció monitor_devices

- **def process_device(self, device: Device):** La funció process_device s'encarrega de recopilar informació detallada sobre un dispositiu de xarxa específic. Primer, inicia un cronòmetre per mesurar el temps de resposta de l'operació. Després, utilitzant NAPALM, estableix una connexió amb el dispositiu i recull diversos tipus de dades: fets generals del dispositiu, el seu entorn, configuració, taula ARP, interfícies, comptadors d'interfícies i veïns BGP. Després de recollir aquestes dades, tanca la connexió i calcula l'ús de CPU i RAM del dispositiu. Finalment, actualitza l'estat del dispositiu a la base de dades i elimina els registres d'estat més antics si excedeixen un límit anteriorment definit.

```python
def process_device(self, device: Device):
    start_time = time.time()
    try:
        driver = get_network_driver(device.os)
        with driver(device.ip_address, device.username, device.password) as device_conn:
            facts = device_conn.get_facts()
            environment = device_conn.get_environment()
            config = device_conn.get_config()
            arp_table = device_conn.get_arp_table()
            interfaces = device_conn.get_interfaces()
            interfaces_counters = device_conn.get_interfaces_counters()
            bgp_neighbors = device_conn.get_bgp_neighbors()
            device_conn.close()

            used_cpu = round(
                sum(cpu_info["%usage"] for cpu_info in environment['cpu'].values()) / len(environment['cpu']), 2)
            used_ram = environment['memory']['used_ram']
            total_ram = environment['memory']['available_ram']
            used_ram_mb = round(used_ram / (1024 ** 2), 2)
            used_ram_percentage = round((used_ram / total_ram) * 100, 2)

            end_time = time.time()
            response_time = round(end_time - start_time, 2)

            self.update_device_status(device, facts, used_cpu, used_ram_mb, used_ram_percentage,
                                      response_time, config, arp_table, interfaces, interfaces_counters,
                                      bgp_neighbors)
            self.delete_oldest_status(device.id)

    except Exception as e:
        print(f"Failed to connect or retrieve data for {device.ip_address}: {e}")
        self.update_device_status_failure(device.id)
```

Fig. 6.12. Funció process_device

- **def update_device_status(self, device: Device, facts: dict, used_cpu: float, used_ram_mb: used_ram_percentage: float, response_time: float, config: dict = None, arp_table: list = None, interfaces: dict = None, interfaces_counters: dict = None, bgp_neighbors: dict = None):** La funció update_device_status actualitza l'estat d'un dispositiu a la base de dades. Primer, obre un context d'aplicació i comença una transacció a la base de dades. Després, busca el dispositiu a la base de dades i actualitza diversos camps amb la informació obtinguda en process_device, incloent-hi dades generals, ús de CPU i RAM, temps de resposta i l'última hora de verificació. Si es proporciona una configuració, també actualitza la taula de configuració del dispositiu. Així mateix, actualitza les taules corresponents a la taula ARP, interfícies i veïns BGP si es proporciona aquesta informació. Finalment, confirma i tanca la transacció.

```python
def update_device_status(self, device: Device, facts: dict, used_cpu: float, used_ram_mb: float,
                         used_ram_percentage: float, response_time: float, config: dict = None, arp_table: list = None,
                         interfaces: dict = None, interfaces_counters: dict = None,
                         bgp_neighbors: dict = None):
    try:
        with app.app_context():
            db.session.begin()
            device_db = Device.query.filter_by(id=device.id).first()
            device_db.current_status = True

            if facts:
                device_db.fqdn = facts['fqdn']
                device_db.hostname = facts['hostname']
                device_db.model = facts['model']
                device_db.os_version = facts['os_version']
                device_db.serial = facts['serial_number']
                device_db.vendor = facts['vendor']
                device_db.uptime = facts['uptime']

            device_db.cpu = used_cpu
            device_db.memory = used_ram_mb
            device_db.response_time = response_time
            device_db.memory_percentage = used_ram_percentage
            device_db.last_checked = datetime.now()

            if config:
                device_db.current_configuration = config['running']
                self.update_device_config_table(device_db.id, config)

            self.update_device_status_table(device_db.id, status: True, used_cpu, used_ram_percentage, response_time)

            if arp_table:
                self.update_arp_table(device_db.id, arp_table)

            if interfaces_counters:
                self.update_interfaces_table(device_db.id, interfaces, interfaces_counters)

            if bgp_neighbors:
                self.update_bgp_table(device_db.id, bgp_neighbors)

            db.session.commit()
            db.session.close()

    except SQLAlchemyError as e:
        db.session.rollback()
        print(f"Database error: {e}")

    except Exception as e:
        db.session.rollback()
        print(f"An unexpected error occurred: {e}")
```

Fig. 6.13. Funció update_device_states

- **async def monitor_hosts(self, host_monitor_interval: int)**: La funció monitor_hosts és un bucle asincrònic que monitora hosts de xarxa a intervals de temps específics. En cada iteració del bucle, crida a process_hosts per obtenir una llista d'hosts i després actualitza la base de dades amb aquesta llista mitjançant update_hosts. Després de cada cicle de monitoratge, espera el temps definit per host_monitor_interval abans de començar la següent iteració.

```python
async def monitor_hosts(self, host_monitor_interval: int):
    while True:
        hosts = self.process_hosts()
        self.update_hosts(hosts)
        print(f"Completed monitoring cycle for hosts.")
        await asyncio.sleep(host_monitor_interval)
```

Fig. 6.14. Funció monitor_hosts

- **def process_hosts(self):** La funció process_hosts recopila informació sobre els hosts a la xarxa. Obtè la porta d'enllaç predeterminada, la seva adreça IP i màscara de xarxa. Després, utilitza Scapy per enviar sol·licituds ARP a la xarxa i recull les respostes. Per a cada resposta, crea una instància de Host amb l'adreça IP, adreça MAC i un nom d'host desconegut, i agrega aquests hosts a una llista. Finalment, retorna aquesta llista d'hosts.

```python
def process_hosts(self):
    try:
        gateways = netifaces.gateways()
        default_gateway = gateways['default'][netifaces.AF_INET][1]

        addrs = netifaces.ifaddresses(default_gateway)
        ip_info = addrs[netifaces.AF_INET][0]
        ip_address = ip_info['addr']
        netmask = ip_info['netmask']

        network = ip_network( address: f"{ip_address}/{netmask}", strict=False)

        ans, _ = s.arping(str(network))

        hosts = []
        for _, received in ans:
            ip_address = received.psrc
            mac_address = received.hwsrc
            hostname = "Unknown"

            host = Host(name=hostname, ip_address=ip_address, mac_address=mac_address)
            hosts.append(host)

        return hosts
    except Exception as e:
        print(f"Failed to process hosts: {e}")
        return []
```

Fig. 6.15. Funció process_hosts

- **def update_hosts(self, hosts)**: La funció update_hosts actualitza la llista d'hosts a la base de dades. Obre un context d'aplicació, comença una transacció, elimina totes les entrades existents a la taula d'hosts i agrega cada host nou a la sessió de la base de dades. Finalment, confirma i tanca la transacció.

```python
def update_hosts(self, hosts):
    try:
        with app.app_context():
            db.session.begin()
            db.session.query(Host).delete()
            for host in hosts:
                db.session.add(host)
            db.session.commit()
            db.session.close()
    except Exception as e:
        db.session.rollback()
        print(f"Failed to update hosts: {e}")
```

Fig. 6.16. Funció update_hosts

6.1.4. Base de dades

En el desenvolupament d'aquest projecte, s'ha optat per utilitzar SQLite com a sistema de gestió de la base de dades, integrat a través de SQLAlchemy, una eina ORM (Object-Relational Mapping) que facilita la interacció entre l'aplicació Python i la base de dades. La decisió d'escollir SQLite es deu a la seva simplicitat i eficiència per a projectes de mida petita a mitjana, a més de no requerir la configuració d'un servidor de base de dades independent, cosa que simplifica tant el desenvolupament com el desplegament de l'aplicació. Per assegurar un millor rendiment i optimitzar els recursos, s'ha configurat SQLAlchemy per desactivar el seguiment de modificacions, ja que aquesta funció, tot i ser útil en alguns contextos, no és necessària per a la nostra aplicació i podria alentir les operacions de base de dades.

6.1.5. Data

S'ha creat un directori per definir tots aquells objectes que vulguem controlar i es requereixin certes dades per poder-los monitorar. Un cop s'inicia l'aplicació, s'omple la base de dades amb la informació d'aquests fitxers ".yaml". Els fitxers creats són els següents:

- **devices.yaml:** Defineix totes les dades necessàries per establir connexions SSH amb els dispositius.

```yaml
- id: 1
  ip_address: 192.168.0.27
  hostname: FirstRouter
  username: albert
  password: albert
  os: ios
  # transport: napalm
  domain_name:
  mac_address:
  ssh_port: 22
  vendor: cisco
```

Fig. 6.17. Devices.yaml

6.2. Frontend

La capa de presentació o frontend és la interfície a través de la qual els usuaris interactuen amb l'aplicació (backend). Per dur a terme aquesta tasca en el nostre projecte, s'ha escollit el framework React. Aquest facilita la creació d'interfícies d'usuari interactives i dinàmiques gràcies a l'ús de components reutilitzables.

Per altra banda, s'ha dissenyat una estructura que mantingui una organització modular que permeti separar les diferents parts de l'aplicació segons la seva funció i responsabilitat. Això ha permès construir l'aplicació d'una manera més organitzada i mantenible en el temps. Els directoris creats són els següents:

- **Components:** Conjunt d'elements reutilitzables de UI.
  - *App*
  - *Common:* Elements comuns utilitzats durant tot el projecte.
  - *DeviceData:* Gràfics que ens permetran visualitzar dades dels dispositius.
  - *Layout:* Elements del layout.
  - *Views:* Diferents vistes que pot tenir una pàgina.
- **Pages:** Components que representen pàgines senceres o vistes dins de l'aplicació.
- **Routes:** Configuració de rutes que determina quin component es mostra segons l'URL.
- **Styles:** Fules d'estil CSS que defineixen l'aparença visual de l'aplicació.
- **Utils;** Funcions auxiliars i utilitats que ofereixen funcionalitats addicionals a l'aplicació.

6.2.1. Pages

HomePage

La pàgina HomePage del frontend és la pàgina principal del tauler de control de l'aplicació. Serveix com un centre centralitzat on els usuaris poden obtenir una visió general de l'estat de tots els dispositius connectats al sistema. Aquesta pàgina mostra diverses gràfiques i estadístiques que resumeixen informació clau com l'estat dels dispositius (actius o inactius), l'ús mitjà de CPU i memòria, el temps de resposta mitjà, i l'estat de les tasques programades (actives, finalitzades o programades). Les gràfiques es generen utilitzant dades en temps real, proporcionant una representació visual clara i concisa de la salut i el rendiment de la xarxa. En resum, HomePage proporciona als usuaris una visió ràpida i efectiva del funcionament general del sistema, facilitant la presa de decisions informada i el monitoratge continu de la xarxa.

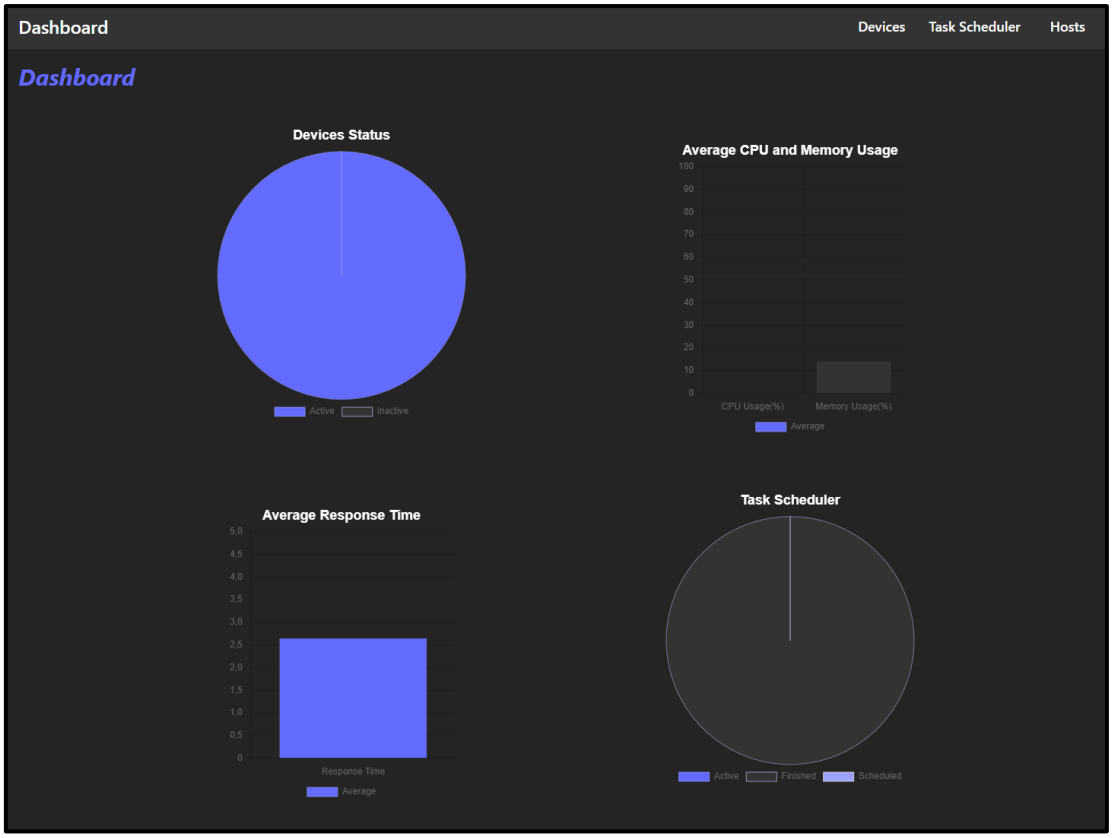


Fig. 6.18. HomePage

DevicesPage

Aquesta pàgina servirà a l'administrador de la xarxa per observar i analitzar l'estat i les estadístiques dels dispositius connectats i supervisats. Per obtenir aquests estats, es fa una crida a l'API cada deu segons. En concret, es crida la petició GET "/devices".

Cada fila de la taula representa un dispositiu i per cada dispositiu, es determina el seu estat (actiu o inactiu), es proporciona informació detallada com l'adreça IP, el fabricant, el sistema operatiu, així com les estadístiques de rendiment com l'ús de la CPU i la memòria. I en el cas que certa informació no estigui disponible, es mostra un valor "N/A" per indicar la seva absència.

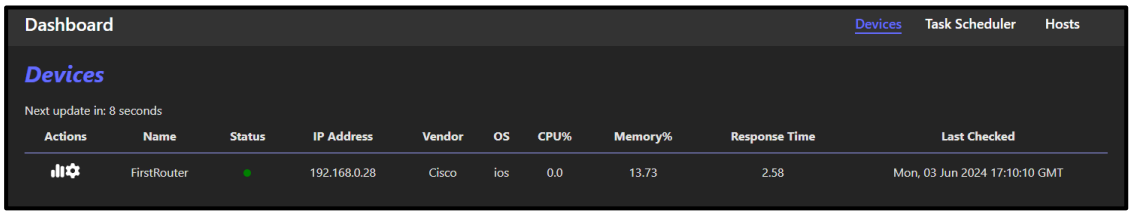


Fig. 6.19. DevicesPage

DevicesSubPage

La pàgina DevicesSubPage del frontend serveix com una vista detallada i específica per a un dispositiu individual dins del sistema. Aquesta pàgina permet als usuaris visualitzar i gestionar informació detallada sobre un dispositiu específic seleccionat de la llista general de dispositius. Inclou característiques com la visualització de dades en temps real sobre l'estat i rendiment del dispositiu, com l'ús de CPU, memòria, temps de resposta, i altres paràmetres importants. També proporciona funcionalitats per executar ordres CLI directament en el dispositiu, veure la seva configuració actual, historial de configuracions, i restaurar configuracions anteriors.

Fig. 6.20. DevicesSubPage

TaskDataPage

La pàgina TaskDataPage del frontend està dissenyada per visualitzar els resultats i les dades generades per les tasques programades dins del sistema. Aquesta pàgina permet als usuaris veure informació detallada sobre l'execució de les tasques, incloent-hi els resultats de les ordres CLI executades, l'estat final de cada tasca, i altres dades rellevants recollides durant la seva execució. Els usuaris poden analitzar aquests resultats per comprendre millor el rendiment i l'eficàcia de les tasques, així com per identificar possibles problemes o àrees de millora. En resum, TaskDataPage proporciona

una interfície per a la visualització i anàlisi de les dades i resultats de les tasques programades, facilitant així la revisió i el seguiment detallat de les operacions de la xarxa.

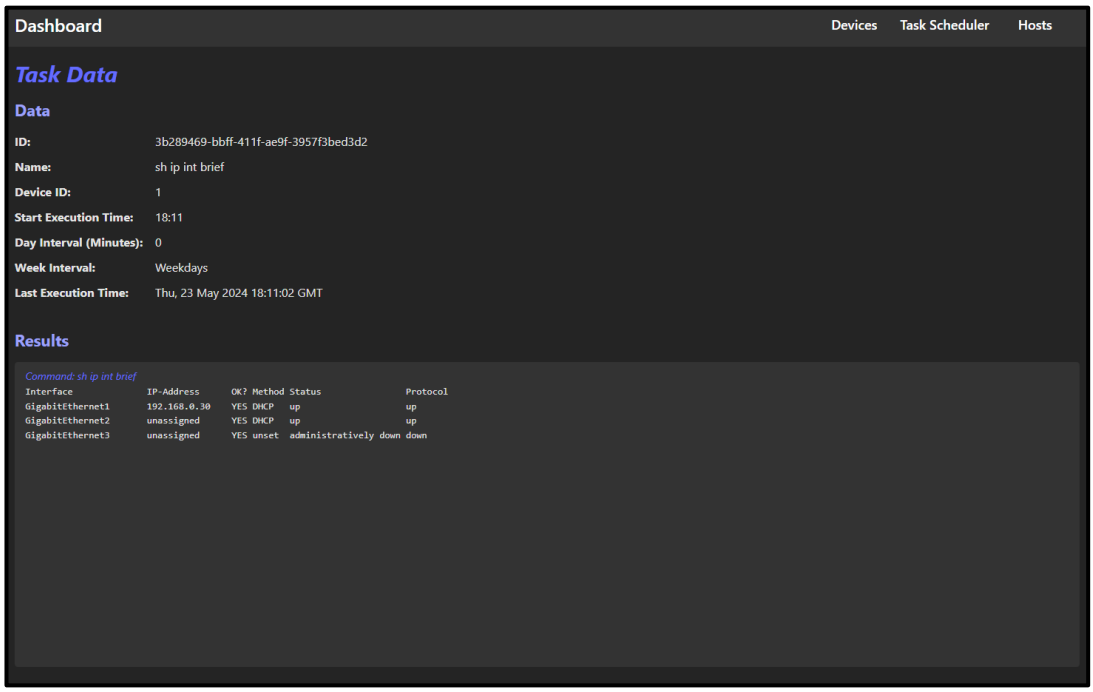


Fig. 6.21. TaskDataPage

TaskEditPage

La pàgina TaskEditPage del frontend està dissenyada per permetre als usuaris crear, editar i configurar tasques programades dins del sistema. Aquesta pàgina proporciona una interfície on els usuaris poden especificar els detalls de les tasques, com ara els dispositius sobre els quals s'executaran, les ordres CLI que es realitzaran, i els horaris d'execució. Els usuaris poden definir intervals de repetició, seleccionar dies específics per a l'execució de les tasques, i establir altres preferències de configuració. Aquesta funcionalitat permet una gestió flexible i detallada de les tasques, assegurant que s'ajustin a les necessitats operatives i de manteniment de la xarxa.

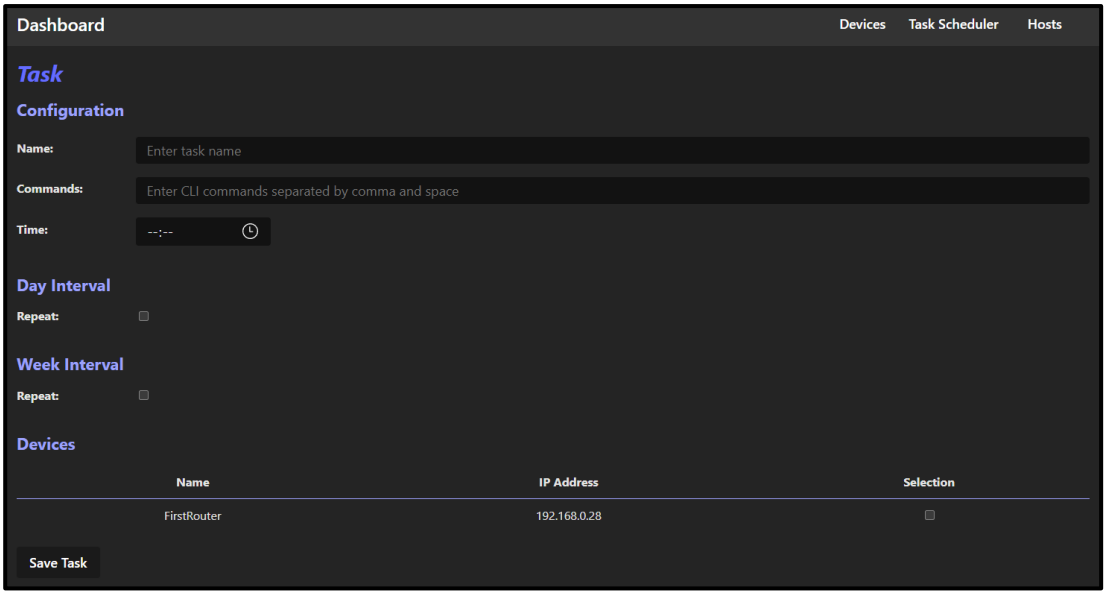


Fig. 6.22. TaskEditPage

TaskSchedulerPage

La pàgina TaskSchedulerPage del frontend està dissenyada per gestionar i supervisar totes les tasques programades dins del sistema. Aquesta pàgina proporciona una visió general de totes les tasques, mostrant informació com l'estat actual (activa, pausada, finalitzada), els dispositius involucrats, els horaris d'execució i altres detalls rellevants. Els usuaris poden utilitzar aquesta pàgina per crear noves tasques, editar les existents, i controlar la seva execució mitjançant opcions per aturar, reprendre o cancel·lar tasques. A més, es pot accedir als resultats de les tasques ja completades per avaluar-ne el rendiment.

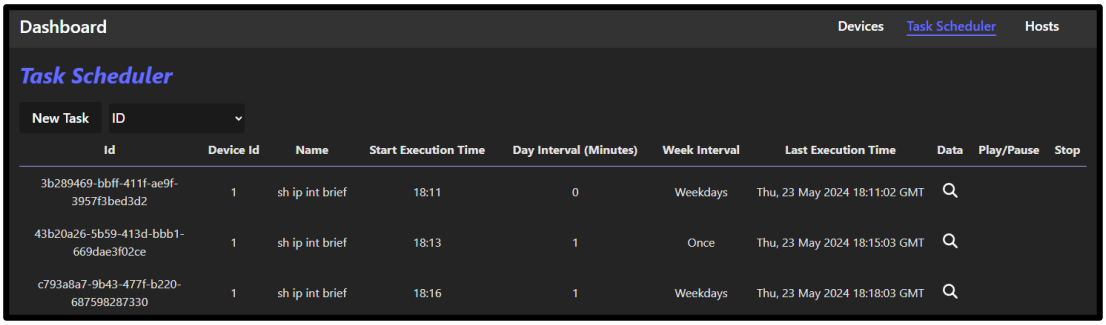


Fig. 6.23. TaskSchedulerPage

HostsPage

De manera similar a la pàgina de dispositius, aquesta pàgina mostra una taula dels equips presents. No tan sols aquells dispositius definits en els fitxers “.yaml” seran els que apareixeran, sinó tots aquells que estiguin disponibles a la xarxa. No obstant això, al no monitorar aquests dispositius, la informació serà més reduïda i menys precisa. Per

obtenir aquestes dades es crida la petició GET “/hosts” i cada de segons es repeteix l’operació per mantenir l’estat de la xarxa actualitzat.

Cada fila de la taula representa un dispositiu i per cada dispositiu, es mostra el nom, l’adreça IP i l’adreça MAC.

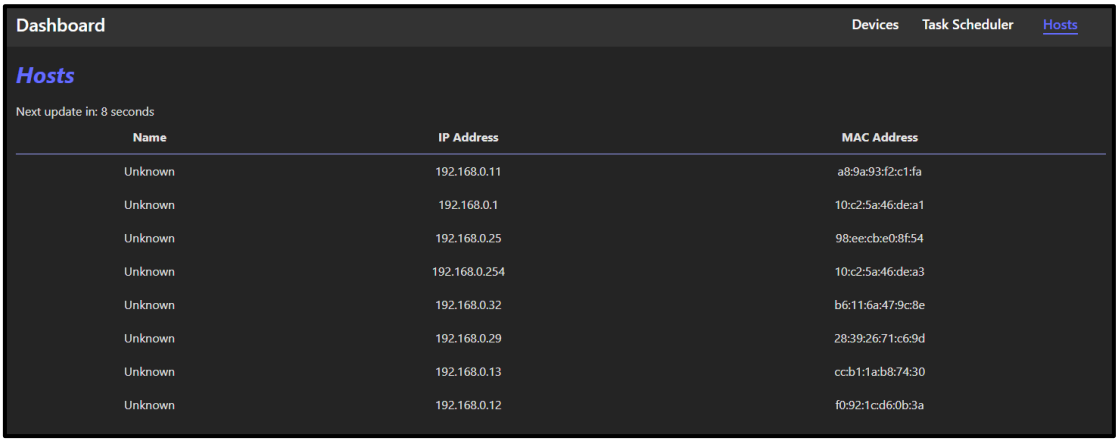


Fig. 6.24. HostsPage

6.3. GNS3

Un cop el projecte ha estat finalitzat, s'ha disposat a crear un entorn de simulació a la plataforma GNS3 per provar en una situació el més versemblant a la realitat la usabilitat i manteniment del projecte.

6.3.1. Disseny de topologia

Per dur a terme aquesta part del projecte, ha calgut en un primer moment dissenyar la topologia per testejar el projecte. En un primer moment es volia optar per utilitzar el Cloud i una interfície amb adaptador pont per poder comunicar l'ordinador amb la màquina virtual de GNS3 i poder executar el projecte fora de GNS3. Malgrat això, l'adaptador pont no ha funcionat adequadament i s'ha optat per la opció de crear una màquina virtual Windows per poder executar el projecte dins de GNS3.

Per altra banda, s'ha optat per utilitzar VMWare per configurar la màquina virtual GNS3 adequadament. Seguidament s'ha dissenyat i configurat la topologia. Es tracta d'una topologia molt senzilla on apareix els següents components:

- **Màquina virtual Windows:** Serveix per muntar i executar el projecte.
- **Switch Level 2:** S'ha utilitzat un switch de capa 2 per poder arribar a dos routers des de l'única interfície de la màquina virtual.
- **2 Routers C3660:** S'utilitzen com vectors de proves per la monitoratge. Amb dos routers és suficient per provar el multi-monitoratge simultània de diversos dispositius.

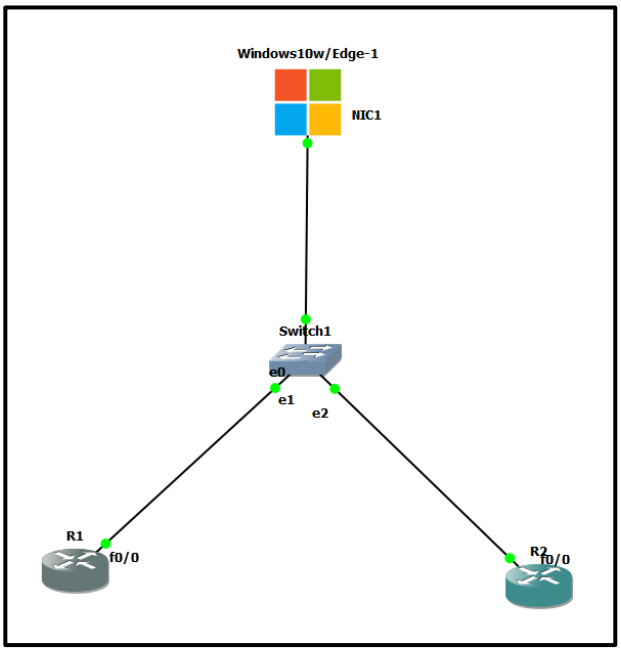


Fig. 6.25. Topologia de GNS3

6.3.2. Configuració de GNS3

La configuració de GNS3 no és quelcom simple i per dur a terme la configuració completa de l'entorn ha calgut dur a terme una sèrie de passes.

Primer de tot, cal descarregar GNS3 a partir del portal corresponent. Seguidament, com s'ha mencionat anteriorment, cal descarregar i configurar una màquina virtual GNS3 que s'encarregarà d'emular i simular els diferents dispositius que afegim a la topologia sense necessitat de hardware físic.

Seguidament, s'ha configurat la màquina en l'entorn de virtualització VMWare i s'ha afegit a les preferències de GNS3. D'aquesta manera, cada cop que s'obri GNS3 s'obrirà la màquina virtual de manera automàtica.

En primer pas és descarregar i importar la imatge d'una màquina virtual Windows. A continuació, es configurà tot l'entorn de la màquina per poder executar el projecte. Ha calgut instal·lar PyCharm pel backend i Visual Studio Code i node.js pel frontend. A més a més, ha calgut instal·lar les llibreries pertinents pel backend i pel frontend.

Seguidament, s'ha afegit un switch de capa dos que connecti la interfície de la màquina virtual Windows. No ha calgut importar cap imatge perquè GNS3 ja disposa de switchos de capa dos

L'últim pas ha consistit en descarregar i importar la imatge del Router C3660. No s'ha escollit aquest model per res en especial, més enllà de què permet configurar el transport SSH. S'han afegit dos routers a la topologia i a cadascun se li hagut de configurar una IP a la interfície de cadascun que connectarà amb el Switch. Per altra banda també cal habilitar les comunicacions SSH per cada router. Per dur a terme aquest pas, cal seguir un conjunt d'instruccions concretes:

- Afegir un hostname personalitzat.
- Habilitar el nom de domini.
- Generar les claus RSA per l'encriptació SSH i afegir la mida de la clau en bits.
- Crear un usuari local amb privilegis.
- Configurar les línies VTY perquè acceptin connexions SSH.
- Assegurar que el servidor SSH estigui habilitat al router amb la comanda ip ssh version 2.
- Guardar la configuració.

7. Conclusions

Durant el desenvolupament d'aquest projecte de creació d'una plataforma per a l'automatització i el monitoratge de xarxes, s'han aconseguit assolir la majoria dels objectius plantejats inicialment. Aquesta plataforma permet millorar l'eficiència operativa, reduir errors humans i millorar el monitoratge de les xarxes de manera significativa. No obstant això, es reconeix que la reducció del temps de resposta no s'ha assolit completament degut a la complexitat inherent en la gestió de múltiples peticions i funcions en temps real.

La millora de l'eficiència operativa ha estat notable gràcies a la capacitat de la plataforma per automatitzar tasques de configuració i manteniment de dispositius de xarxa, reduint així considerablement la càrrega de treball per als administradors de xarxa. A més, l'automatització de processos crítics ha disminuït el marge d'error humà, millorant la fiabilitat i la seguretat de les xarxes gestionades. Pel que fa al monitoratge, la capacitat de supervisar en temps real l'estat i el rendiment dels dispositius de xarxa ha permès detectar i solucionar problemes de manera més eficient, assegurant una operació continuada i òptima de la xarxa. La solució desenvolupada és adaptable a diverses infraestructures de xarxa, independentment del fabricant del dispositiu, i incorpora mesures de seguretat adequades per garantir la integritat i la confidencialitat de les dades gestionades.

Al llarg del projecte, s'ha après i aplicat una sèrie d'eines i tecnologies clau. L'ús del framework Flask ha estat fonamental per al desenvolupament del backend, permetent la creació d'una API robusta i fàcilment escalable. L'aprenentatge i implementació de llibreries com NAPALM i Netmiko han estat crucials per establir connexions amb dispositius de xarxa i automatitzar tasques de configuració i manteniment. A més, s'han adquirit coneixements pràctics sobre el monitoratge de xarxes, la gestió d'interfícies de línia de comandaments i l'automatització de processos complexos.

Un altre èxit important ha estat la creació d'un entorn de simulació en GNS3. Tot i els problemes inicials, com la configuració de la màquina virtual i la integració dels dispositius, s'ha aconseguit muntar una topologia funcional que ha permès provar i validar les funcionalitats de la plataforma en un entorn controlat. Aquesta experiència ha estat enriquidora i ha proporcionat una comprensió més profunda dels desafiaments associats amb la implementació de solucions de xarxa en entorns simulats.

Finalment, es pot afirmar que els objectius del projecte s'han assolit amb èxit. Tot i que hi ha hagut reptes i obstacles, el coneixement adquirit i les solucions desenvolupades són

testimoni del treball dedicat i l'aprenentatge continu. Es pot estar satisfet del treball fet, tot i que en un altre context, potser amb més recursos i temps, la producció i l'impacte d'aquest projecte podrien haver estat encara majors. La base establerta ofereix un gran potencial per a futures millores i expansions, assegurant que aquesta plataforma pugui continuar evolucionant per satisfer les necessitats canviants de les xarxes empresarials i educatives.

8. Possibles ampliacions

Tot i els èxits aconseguits amb aquest projecte, hi ha diverses àrees en les quals es poden realitzar ampliacions per millorar encara més la funcionalitat i l'eficiència de la plataforma d'automatització i el monitoratge de xarxes.

Una ampliació important seria l'ampliació de la compatibilitat de la plataforma per incloure més fabricants i tipus de dispositius. Actualment, la solució és compatible amb un nombre limitat de dispositius, però una ampliació en aquest aspecte faria la plataforma més versàtil i atractiva per a un públic més ampli. Això implicaria l'addició de nous connectors i la capacitat de gestionar dispositius més diversos.

La incorporació de característiques avançades de seguretat, com la implementació de models Zero Trust o Secure Access Service Edge (SASE), seria una altra millora significativa. Aquestes tecnologies ofereixen un enfocament més robust i integrat per a la seguretat de la xarxa, assegurant que només els usuaris i dispositius autoritzats puguin accedir als recursos de la xarxa independentment de la seva ubicació.

També es podria considerar l'optimització i l'escalabilitat del backend per gestionar un major volum de dades i peticions simultànies. Això inclouria la millora de la infraestructura de la base de dades, possiblement mitjançant la transició a una solució de base de dades distribuïda més robusta, i l'optimització de les operacions de xarxa per reduir el temps de resposta i augmentar l'eficiència general del sistema.

Una altra ampliació significativa seria l'expansió de les capacitats de programació de tasques amb noves funcions. Actualment, la plataforma permet programar tasques bàsiques, però afegir funcionalitats més avançades, com ara la possibilitat de programar tasques condicionals, definir seqüències de tasques, o integrar alertes i notificacions automàtiques, augmentaria considerablement la utilitat de la plataforma. Això facilitaria una gestió més precisa i adaptable de les xarxes, permetent als administradors configurar escenaris complexos i respostes automatitzades a esdeveniments específics.

Finalment, una ampliació important seria el desenvolupament d'una interfície d'usuari més rica i personalitzable, amb la incorporació de dashboards avançats i visualitzacions gràfiques que permetin als usuaris obtenir una visió més completa i intuïtiva de l'estat de la seva xarxa. Això podria incloure la possibilitat de crear informes personalitzats i alertes automàtiques que informin als administradors de qualsevol anomalia o problema detectat.

9. Bibliografia

[1] F. S. Lino Codara, «Resilience, complexity and digital transformation: three case studies in the valves industry,» 20 Febrer 2023. [En línea]. Available: https://www.emerald.com/insight/content/doi/10.1108/JMTM-05-2022-0214/full/html.

[2] J. L. Jee Young Lee, «Current Research Trends in IoT Security: A Systematic Mapping Study,» 12 Març 2021. [En línea]. Available: https://www.hindawi.com/journals/misy/2021/8847099/.

[3] H. T. Y. L. F. B. Qi Yao, «The penetration effect of digital leadership on digital transformation: the role of digital strategy consensus and diversity types,» 14 Abril 2023. [En línea]. Available: https://www.emerald.com/insight/content/doi/10.1108/JEIM-09-2022-0350/full/html.

[4] D. H. L. C. Yuanhang He, «A Survey on Zero Trust Architecture: Challenges and Future Trends,» 15 Juny 2022. [En línea]. Available: https://www.hindawi.com/journals/wcmc/2022/6476274/.

[5] M. F. M. S. J. D. J.D. Case, «A Simple Network Management Protocol,» Agost 1988. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc1067.

[6] T. C. Evans Amoany, «7 Linux networking commands that every sysadmin should know,» 7 Juny 2021. [En línea]. Available: https://www.redhat.com/sysadmin/7-great-network-commands.

[7] Cisco, «Cisco Networking Academy's Introduction to Routing Dynamically,» 24 Març 2014. [En línea]. Available: https://www.ciscopress.com/articles/article.asp?p=2180210&seqNum=5.

[8] S. R. H. C. Huseyin Cavusoglu, «Configuration of and Interaction Between Information Security Technologies: The Case of Firewalls and Intrusion Detection Systems,» Juny 2009. [En línea]. Available: https://www.jstor.org/stable/23015480#:~:text=URL%3A%20https%3A%2F%2Fwww.jstor.org%2Fstable%2F23015480%0A.

[9] I. G. V. P. Jakub Svoboda, «Network Monitoring Approaches: An Overview,» Octubre 2015. [En línea]. Available: https://www.researchgate.net/publication/305957483_Network_Monitoring_Approac

hes_An_Overview.

[10] T. H. T. M. M. J. Ved P. Kafle, «Network Control and Management Automation: Architecture Standardization Perspective,» Septembre 2021. [En línea]. Available: https://www.researchgate.net/publication/355419164_Network_Control_and_Management_Automation_Architecture_Standardization_Perspective.

[11] S. Waldbusser, «Remote Network Monitoring Management Information Base,» Maig 2000. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc2819.

[12] K.R. Sollins, «THE TFTP PROTOCOL (REVISION 2),» Juliol 1992. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc1350.

[13] M. T. M. Tayyab Muhammad, «Network Automation,» 2 Agost 2023. [En línea]. Available: https://ajpojournals.org/journals/index.php/EJT/article/view/1547.

[14] Jean Carlos Tandazo Tandazo, «Automatización de redes utilizadas para EoT : Automatización de redes con Netmiko,» Octubre 2022. [En línea]. Available: https://bibdigital.epn.edu.ec/handle/15000/23212.

[15] Terry Slattery, «How to tackle network automation risks and tasks,» 9 Maig 2019. [En línea]. Available: https://www.techtarget.com/searchnetworking/tip/How-to-tackle-network-automation-risks-and-tasks.

[16] GNS3, «GNS3,» [En línea]. Available: https://docs.gns3.com/.

[17] Cisco, «Teaching with Packet Tracer,» [En línea]. Available: https://www.netacad.com/courses/packet-tracer/teaching.

[18] S. R. C. S. Sven Reißmann, «Using Cisco VIRL and GNS3 to Improve the Scale-out of Large Virtual Network Testbeds in Higher Education,» Juliol 2018. [En línea]. Available: https://www.researchgate.net/publication/326580332_Using_Cisco_VIRL_and_GNS3_to_Improve_the_Scale-out_of_Large_Virtual_Network_Testbeds_in_Higher_Education.

[19] Eve-ng, «Eve-ng,» [En línea]. Available: https://www.eve-ng.net/.

[20] Python, «What is Python? Executive Summary,» [En línea]. Available: https://www.python.org/doc/essays/blurb/.

[21] T. C. B. R. C. F. S. Paul Mihăilă, «Network Automation and Abstraction using Python Programming Methods,» Octubre 2017. [En línea]. Available: https://www.researchgate.net/publication/322017645_Network_Automation_and_Abstraction_using_Python_Programming_Methods.

[22] PythonBasics, «What is Flask Python,» [En línea]. Available: https://pythonbasics.org/what-is-flask-python/.

[23] Suresh Vina, «Python Network Automation with Netmiko - Part 1,» 6 Maig 2022. [En línea]. Available: https://www.packetswitch.co.uk/netmiko-intro/.

[24] Cisco, «NAPALM,» [En línea]. Available: https://developer.cisco.com/codeexchange/github/repo/napalm-automation/napalm/.

[25] Atlassian, «What is scrum and how to get started,» [En línea]. Available: https://www.atlassian.com/agile/scrum.