СИСТЕМА КЕРУВАННЯ КОНФІГУРАЦІЯМИ КОМПОНЕНТІВ МЕРЕЖІ ПІДПРИЄМСТВА

Кваліфікаційна робота

за спеціальністю "Комп’ютерні науки" 122

Київ 2023

Керівник курсової роботи
к. т. н. **Черкасов Д.І.**

Виконала студентка КН-4
Пономаренко К.О.

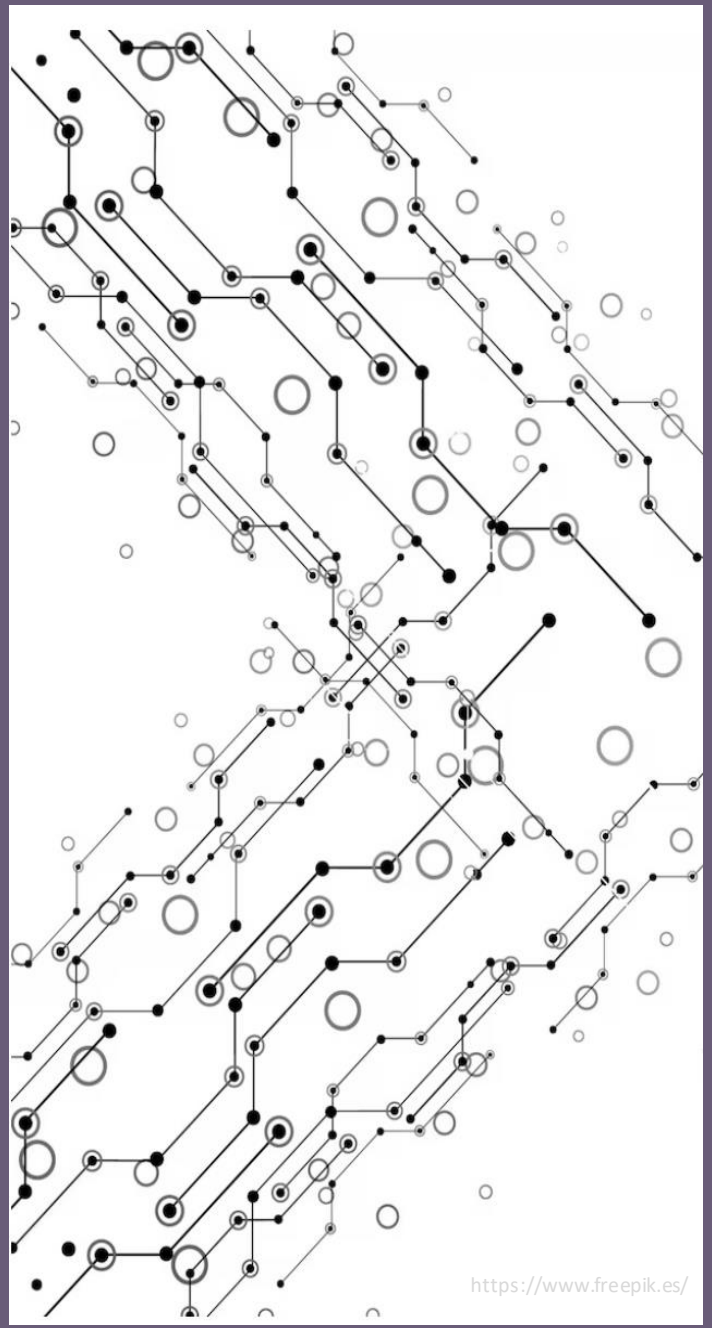
СИСТЕМА КЕРУВАННЯ КОНФІГУРАЦІЯМИ КОМПОНЕНТІВ МЕРЕЖІ (СККККМ)

Комунікаційна мережа (КМ) - невід'ємна складова частина інформаційної інфраструктури сучасних підприємств.

Основні показники ефективності функціювання КМ:

- швидкість передачі даних,
- запобігання втратам інформації,
- збереження інформації про стан мережі та можливість її швидкого відновлення;
- висока доступність за рахунок стійкості до відмов, які можуть виникати під час її роботи.

Для надійної роботи мережі необхідно вирішити **задачу збереження конфігурацій пристроїв та їх оперативного відновлення** у разі потреби.



СИСТЕМА КЕРУВАННЯ КОНФІГУРАЦІЯМИ КОМПОНЕНТІВ МЕРЕЖІ (СККККМ)

Основні задачі, які вирішувалися під час розробки СККККМ:

- визначення ключових функцій СККККМ підприємства;
- дослідження можливих варіантів архітектури СККККМ та їх порівняння;
- дослідження можливих підходів до проектування СККККМ;
- визначення продуктів, що можуть бути використані як складові частини системи та їх функціональних можливостей і обмежень;
- виконання структурної розробки власного рішення СККККМ;
- детальна розробка одного з компонентів змодельованого рішення.

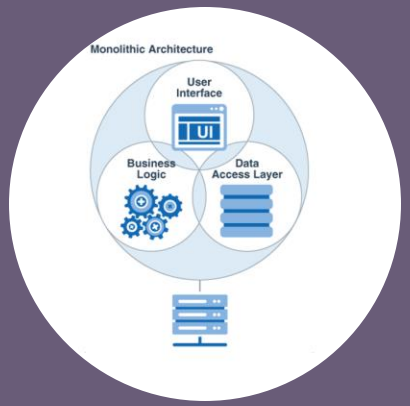
СИСТЕМА КЕРУВАННЯ КОНФІГУРАЦІЯМИ КОМПОНЕНТІВ МЕРЕЖІ (СККККМ)

Основні функції СККККМ:

- керування конфігураціями;
- контроль поточного стану мережі;
- керування резервними копіями конфігурацій;
- керування всіма елементами програмного забезпечення мережі.

Можливі варіанти архітектури СКККМ

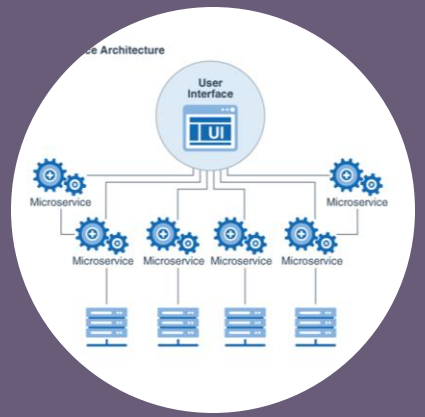
Монолітна архітектура



+ простота організації процесу розробки та впровадження;
+ швидке наскрізне тестування.

− обмежене масштабування;
− не підходить для систем з багаторівневою структурою;
− повне повторне тестування після внесення змін.

Модульна архітектура - мікросервіси



+ висока швидкість запуску системи та її розгортання;
+ гнучкість у виборі елементів системи;
+ можливість масштабування, оновлення та повторне тестування окремих сервісів;
+ висока стійкість системи.

− складність та високі витрати.

Продукти, які можуть бути використані як складові частини СККККМ:

Об'єктне сховище Amazon S3 має управління версіями, яке дозволяє:

- зберігати,
- отримувати,
- відновлювати

кожну версію будь-якого об'єкта, що зберігається.

MinIO - мульти-хмарне об’єктне сховище, яке:

- є сумісним з Amazon S3 API,
- є нативним для Kubernetes,
- може бути розгорнутим локально чи на будь-якій хмарі.

Ansible - система автоматизованого збору конфігурацій, яка полегшує масштабування інфраструктури:

- відкритий код,
- push-архітектура і механізм інвертування в pull-архітектуру,
- Playbooks пишуться на YAML.

Puppet - інструмент керування конфігураціями для автоматизації та централізації процесу управління налаштуваннями:

- використовує клієнт-серверну архітектуру,
- Використовує доменно-орієнтовану мову (DSL) на базі Ruby.

Інструменти для керування конфігураціями пристроїв

RANCID

(Really Awesome New Cisco config Differ):

- використовує простий текстовий формат для зберігання та отримання конфігурацій,
- підтримує різноманітні механізми автентифікації, включаючи SSH, Telnet і SNMP для безпечного доступу до мережевих пристроїв,
- списки групи дозволяють розділити пристрої згідно архітектурних особливостей мережі на окремі множини,
- оптимізує роботу у випадках, коли конфігурація має бути змінена для великої кількості пристроїв: створюється файл, у якому зберігаються всі необхідні команди, що мають бути виконаними для певного типу пристроїв.

Oxidized:

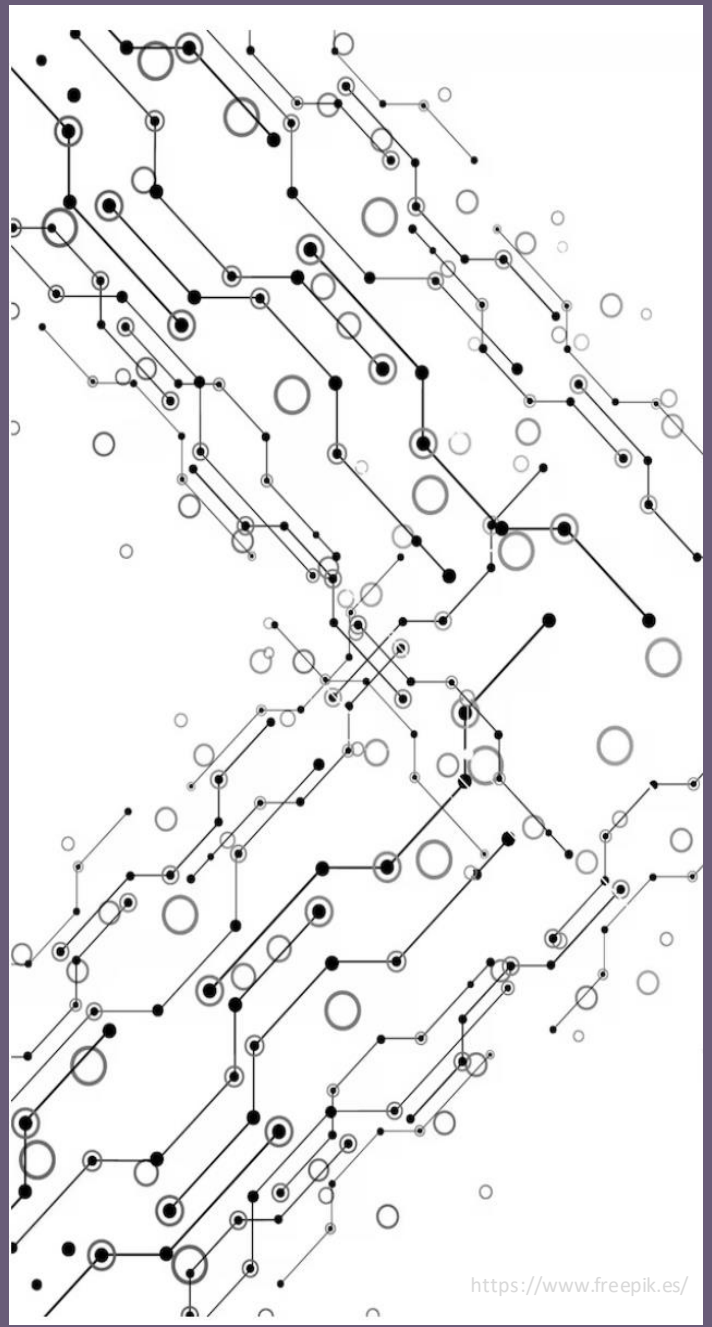
- має ширший функціонал, розроблений з використанням модульної архітектури,
- складається з 4 модулів: Source, Input, Output, Model,
- використовує потоки для кожного окремого пулу конфігурацій,
- автоматично пристосовується до кількості потоків, надаючи можливість користувачам встановлювати ліміти на кількість потоків.

СИСТЕМА КЕРУВАННЯ КОНФІГУРАЦІЯМИ КОМПОНЕНТІВ МЕРЕЖІ (СККККМ)

Результати проведених досліджень дозволили визначити основні інструменти та типові рішення, які доцільно використати під час моделювання та імплементації власної СККККМ.

Це дозволило:

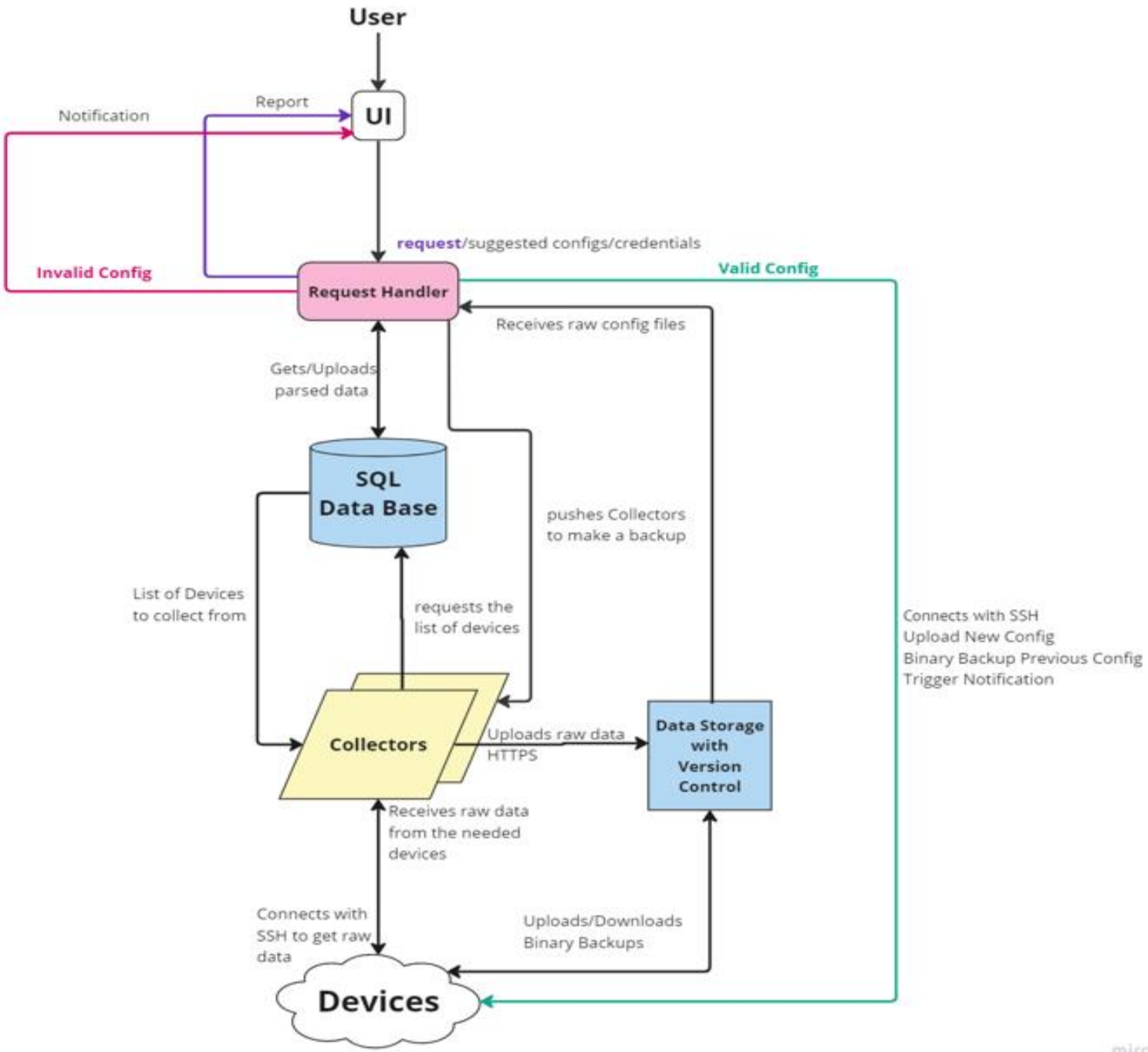
- оптимізувати час моделювання окремих компонентів та СККККМ в цілому;
- визначити напрямки автоматизації виконання окремих та комплексних задач, які вимагаються від СККККМ.



Структурна схема СККККМ

Результат практичної частини роботи - структурно розроблена система керування конфігураціями пристроїв в мережі, яка містить значну кількість пристроїв, розподілених у просторі, та потенційно може вимагати глобального масштабування:

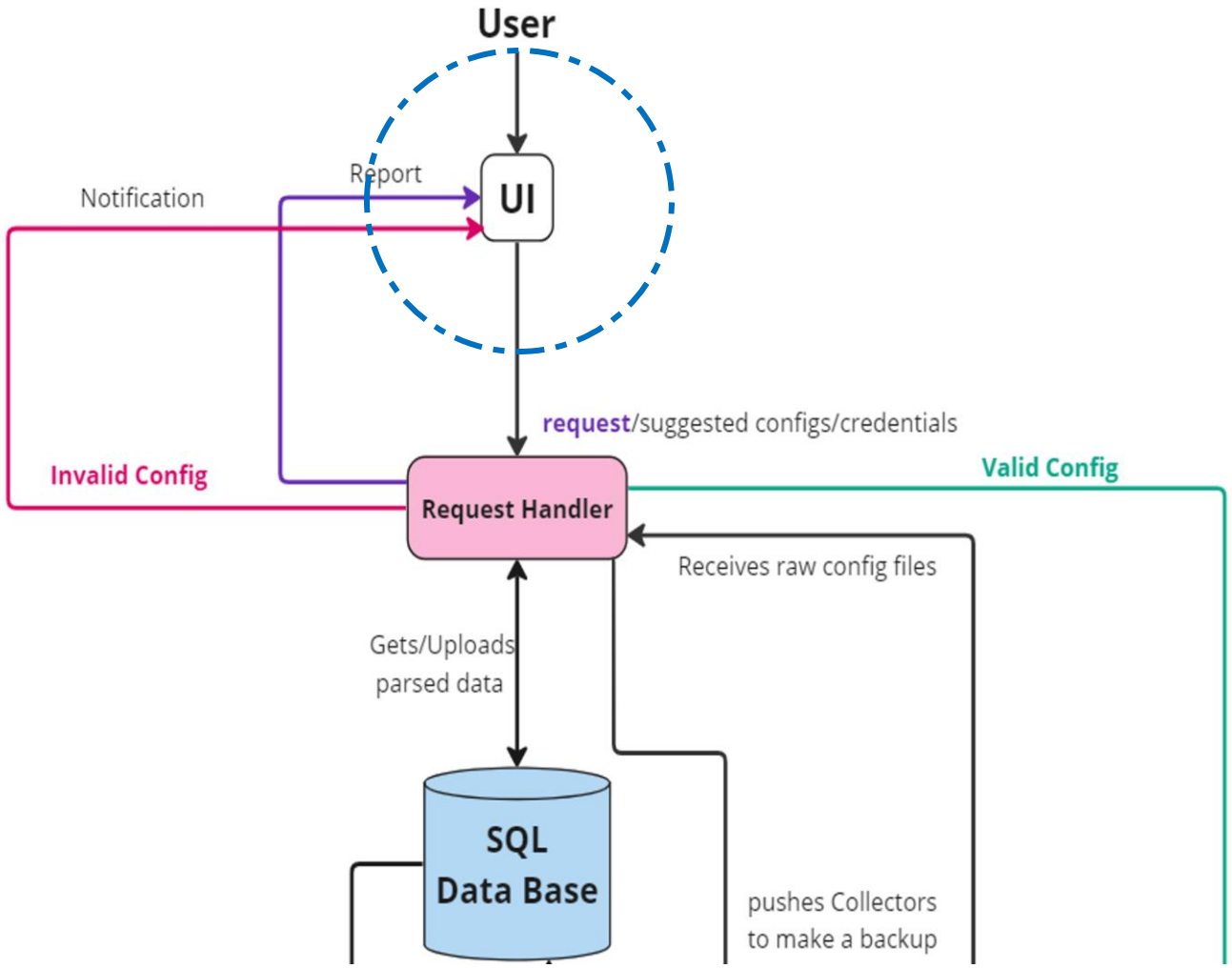
- має модульну архітектуру - це полегшує її масштабування та управління конфігураціями пристроїв,
- забезпечує регулярне створення копій поточних конфігурацій на пристроях.
- дозволяє користувачам аналізувати наявність, стан і особливості конфігурації кожного пристрою;
- надає звіти про різницю в конфігураціях пристроїв.



Структурна схема СККККМ

UI: користувацький веб-інтерфейс (Apache Server)

- дозволяє користувачам швидко отримати доступ до застосунку незалежно від свого місцезнаходження та з будь-якого пристрою;
- ефективно масштабується без додаткових ресурсів на стороні клієнта;
- має полегшений процес обслуговування та оновлення веб-інтерфейсів *(не треба вносити зміни для кожного клієнта, достатньо змінити чи оновити код застосування на стороні сервера)*;
- складається з модулів, які можна конфігурувати вручну;
- може оброблювати всі різні типи запитів зі сторони клієнта.

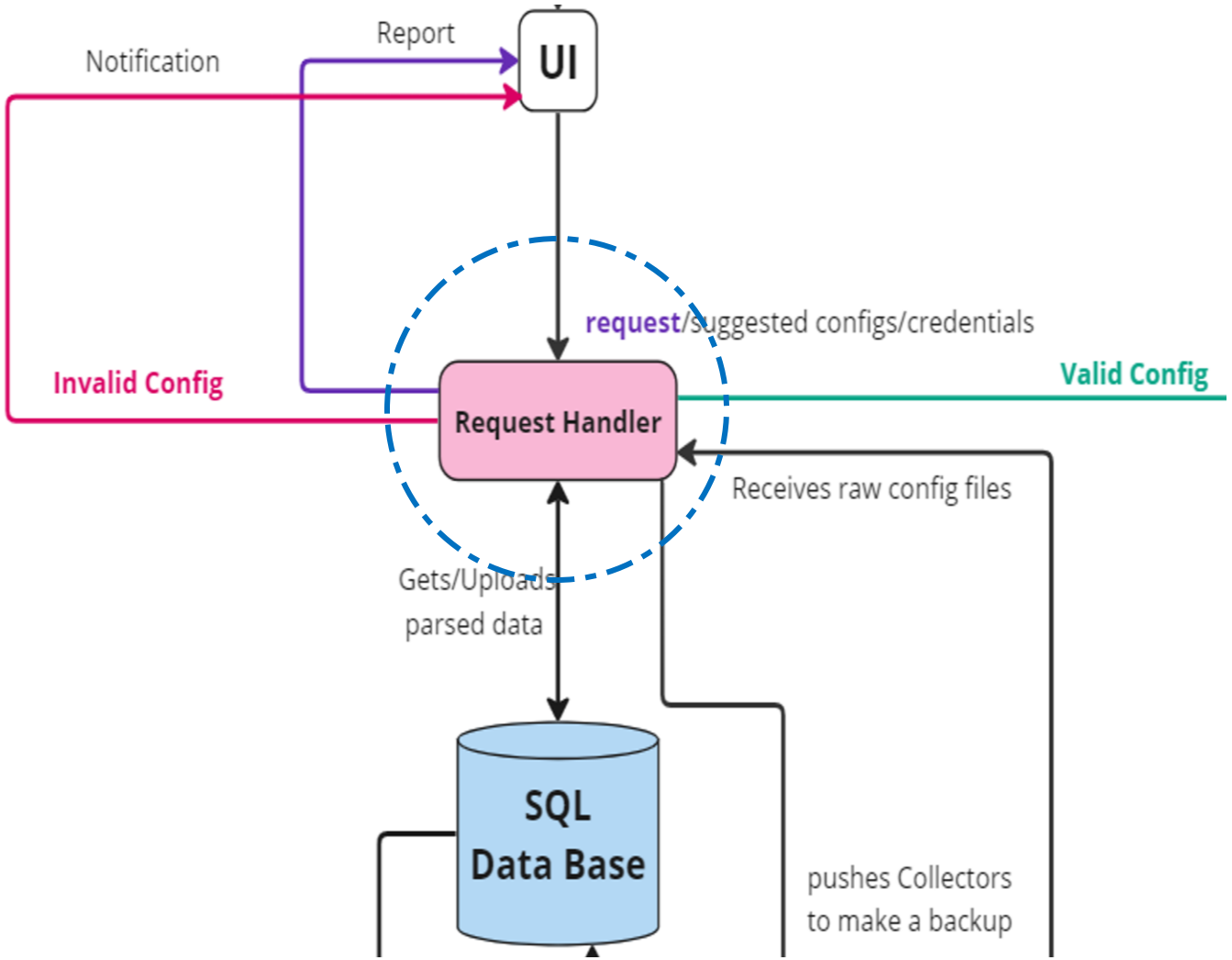


Структурна схема СККККМ

Request Handler - компонент, що обробляє запити від UI, отримуючи та маніпулюючи даними та файлами конфігурацій з DB та Data Storage.

Request Handler виконує наступні функції:

- **Функція «User Access Control» -** керування доступом користувачів до мережевих пристроїв;
- **Функція «Analyze Configuration`s Validity» -** запобігання завантаженню некоректних конфігурацій на пристрої;
- **Функція «Backup» -** запускання процесу створення резервних копій конфігурацій пристрою на вимогу;
- **Функція «Uploading New Config» -** додавання або оновлення конфігурацій пристроїв;

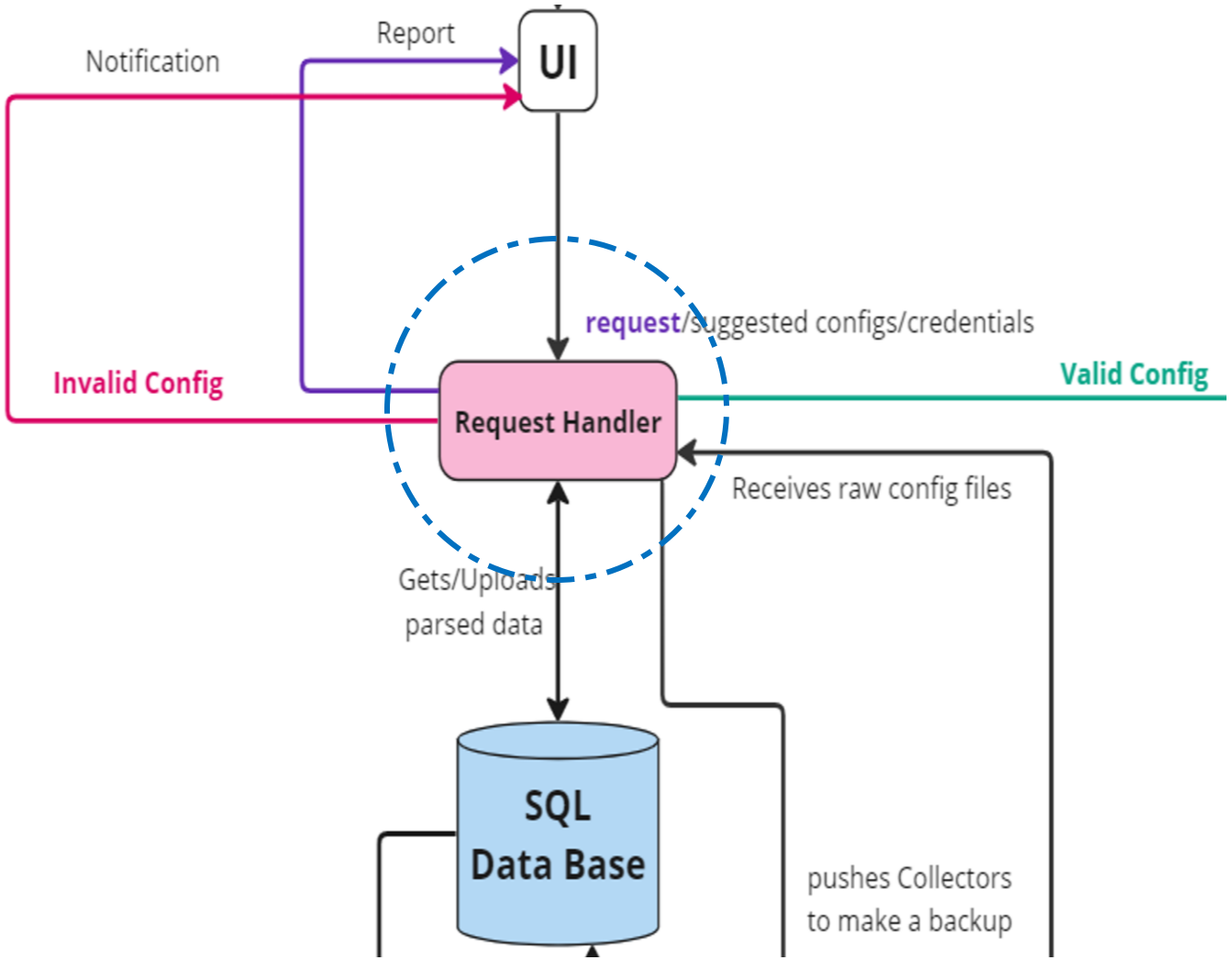


Структурна схема СККККМ

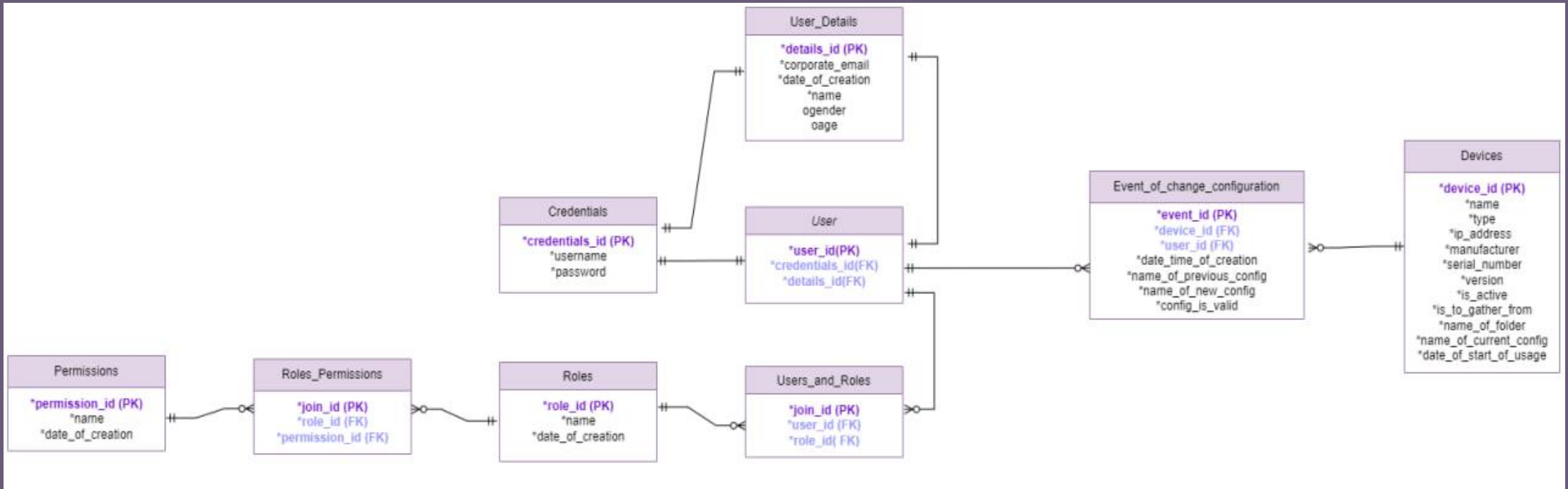
Request Handler - компонент, що обробляє запити від UI, отримуючи та маніпулюючи даними та файлами конфігурацій з DB та Data Storage.

Request Handler виконує наступні функції:

- **Функція «Notification of Users»** - функціонал, що відповідає за сповіщення авторизованих користувачів системи про здійснені зміни до конфігурацій, проблеми роботи мережі тощо;
- **Функція «Compare Configs»** - порівняння поточної конфігурації з попередніми резервними копіями, а також порівняння всіх наявних резервних копій між собою;
- **Функція «Inventory Management»** - відстеження всіх пристроїв в мережі, включаючи основні параметри їх конфігурації, деталі апаратного забезпечення та фізичного розташування.



SQL Data Base



ER-діаграма, яка відображає всі сутності, їх атрибути та зв'язки між ними:

- сутності зображено у вигляді таблиць із списками атрибутів (обов'язкові атрибути позначені символом «*****», а не обов'язкові – «**o**»),
- відношення сутностей одної до іншої позначено стрілками (**craw`s foot notation**),
- якщо атрибут є **primary key** або **foreign key** – це зазначається у дужках.

SQL Data Base

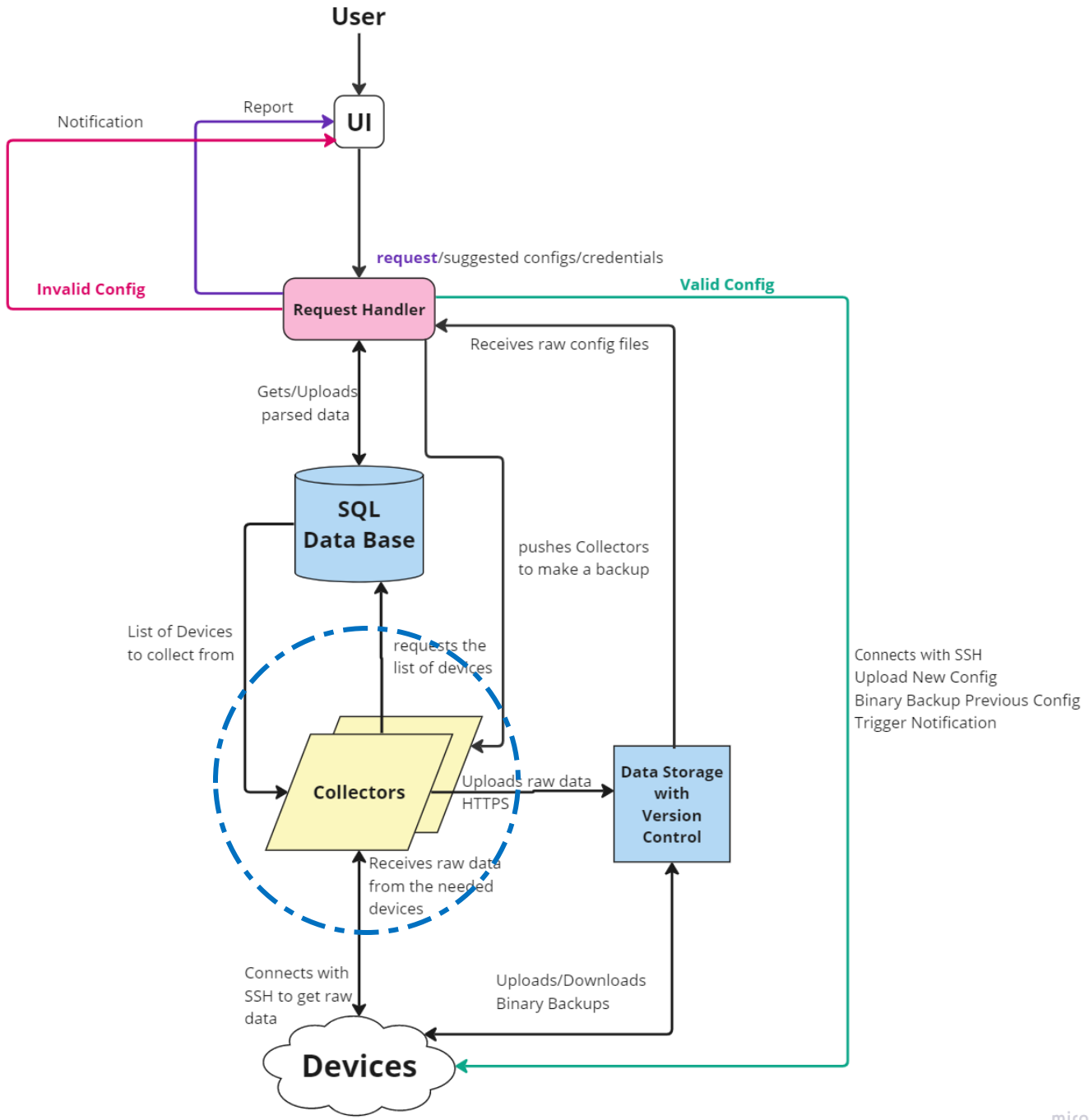
СУТНОСТІ:

User_Details, Credentials	Roles, Permissions	Event_of_Change_of_Configuration	Devices
Для полегшення роботи з інформацією та підвищення рівня безпеки у системі, деталі користувача та його облікові дані було віднесено до окремих сутностей User_Details та Credentials відповідно	Roles зберігає в собі інформацію про всі наявні ролі, які можуть належати користувачам, Permissions – дані про всі можливі дозволи на проведення операцій та дій у системі	Сутність Event_of_Change_of_Configuration зберігає інформацію про всі події, коли відбувалася спроба змінити конфігурацію на якомусь пристрої.	Сутність Devices зберігає інформацію про кожний наявний в мережі пристрій

Структурна схема СККККМ

Data Storage with Version Control - зберігає виключно файли конфігурацій та їх резервні копії:

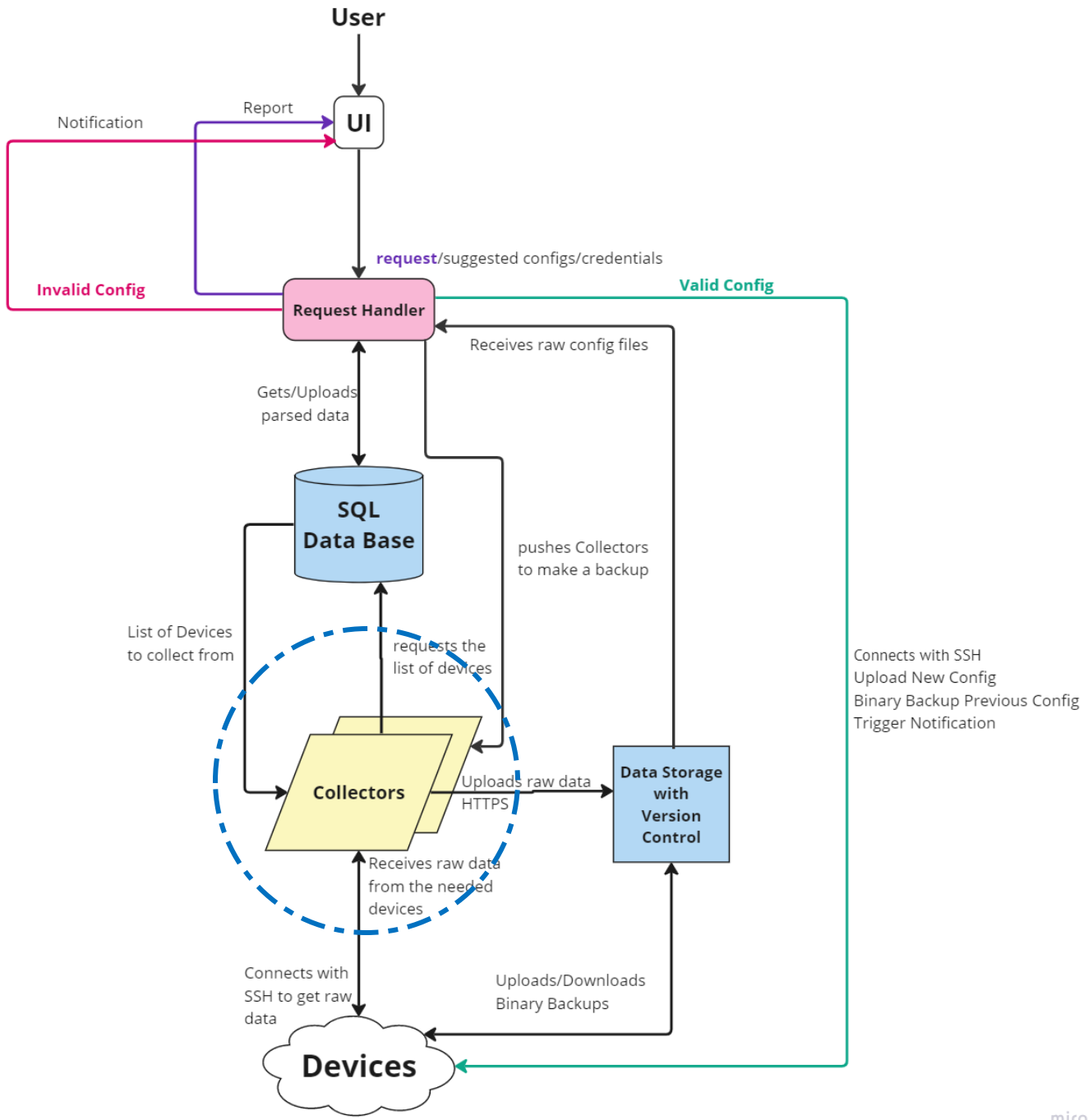
- кожний тип пристроїв має окремі папки, в яких для кожного представника певного типу створюються власні папки, що називаються згідно до імен пристроїв.
- кожний файл має назву у форматі «назваПристрою_датаЧас.conf», що дозволяє унікально ідентифікувати всі конфігураційні файли, тримаючи їх у хронологічному порядку.



Структурна схема СККККМ

Collector - збирають дані з різних мережевих пристроїв та передають їх до сховища даних:

- виконують запит до БД та отримують список пристроїв, з яких необхідно зібрати дані і передати їх до сховища із контролем версій;
- отримують айпіадреси пристроїв, конфігурації яких мають бути зібраними, та дистанційно підключаються до цих пристроїв за протоколом SSH і збирають поточні конфігурації;
- завантажують зібрані файли до сховища даних із контролем версій, використовуючи протокол HTTPS, що шифрує дані протягом передачі.
- у разі потреби зробити резервні копії конфігурацій пристроїв Request Handler надсилає запит до колекторів та запускає процес збору та передачі даних до сховища.



Розробка Collector

Компонент **Collector** може бути програмою, що запускається із різними аргументами утилітою Crontab, що автоматично планує та запускає завдання.

У випадку змодельованої системи визначено дві cronjobs:

- одна виконується щогодини та збирає конфігурації з усіх активних пристроїв,
- друга запускається щохвилини і робить резервну копію конфігурації для новосконфігурованого пристрою, у разі, якщо такий є (is_to_gather_from встановлено в true).

Код колекторів розроблений для системи, що складається з трьох типів девайсів:

- switches,
- routers,
- firewalls.

Розробка Collector

Програма може бути запущеною з наступними аргументами:

all – конфігурації з усіх пристроїв, що є активними буде зібрано;

new – конфігурації з усіх пристроїв, що є активними та мають флаг is_to_gather_from встановленим в true, будуть зібрані;

switch – конфігурації з усіх активних пристроїв типу switch будуть зібрані;

router - конфігурації з усіх активних пристроїв типу router будуть зібрані;

firewall - конфігурації з усіх активних пристроїв типу firewall будуть зібрані

Код Collector

Атрибути класу Device:

- ім’я хоста,
- айпі-адреса девайсу,
- шлях до папки, в якій зберігаються всі конфігураційні файли даного представника класу девайсів.
- **ssh** та **config** - встановлене з девайсом ssh з’єднання та назва конфігураційного файлу, які при ініціалізації отримують значення **None**.

```
class Device:
    def __init__(self, hostname, ip_address, s3_object_prefix):
        self.hostname = hostname
        self.ip_address = ip_address
        self.s3_object_prefix = s3_object_prefix
        self.ssh = None
        self.config = None
```

Розробка Collector

Клас Device має метод connect, який встановлює SSH підключення до мережевого пристрою за указаною його айпі-адресою, іменем користувача та шляхом до ключа SSH, використовуючи модель paramiko.

```python
def connect(self):
    try:
        print(f"Connecting to {self.hostname} ({self.ip_address})...")
        #ssh = paramiko.SSHClient()
        #ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        #ssh.connect(self.ip_address, username=self.username, key_filename=self.ssh_key_path)
        #self.ssh = ssh
        print(f"Connected to {self.hostname} ({self.ip_address}) successfully!")
    except Exception as e:
        print(f"Error occurred while connecting to {self.hostname}: {e}")
```

Розробка Collector

Асинхронний метод **collect_configuration** призначений для збору конфігурації пристроїв.

Він повертає стрічку з конфігурацією та надає значення атрибуту **config** назви файлу, в якому зберігатиметься конфігурація у форматі «НазваХоста_Дата_Час».

Метод **_collect_configuration** є приватним та збирає конфігурацію, запускаючи та записуючи результати команди **show running-config** на пристроях.

```python
async def collect_configuration(self) -> Optional[str]:
    try:
        loop = asyncio.get_running_loop()
        config = await loop.run_in_executor(None, self._collect_configuration)
        now = datetime.datetime.now()
        timestamp = now.strftime('%Y-%m-%d_%H-%M-%S')
        filename = f"{self.hostname}_{timestamp}.conf"
        self.config = filename
        print(f"The configuration for {self.hostname} was collected with a file name {self.config}")
        return config

    except Exception as e:
        print(f"Error occurred while collecting configuration for {self.hostname}: {e}")
        return None

def _collect_configuration(self) -> str:
    #stdin, stdout, stderr = self.ssh.exec_command('show running-config')
    config = f"I`m {self.hostname} and this is my configuration" #stdout.read().decode('utf-8')
    return config
```

Розробка Collector

Device. upload_configuration(s3_bucket_name, config)

Асинхронний метод, що завантажує до **s3 bucket** файл з конфігурацією.

Використовує **configparser**, аби отримати значення ключів для доступу до **AWS s3 bucket** та бібліотеку **boto3** для з'єднання з сховищем та завантаженням сформованого файлу згідно визначеного шляху.

```python
async def upload_configuration(self, s3_bucket_name, config) -> None:
    try:
        if not self.config:
            raise Exception("Configuration file is not found. Please collect configuration first.")

        c = configparser.ConfigParser()
        c.read('config.ini')
        aws_access_key_id = c['s3']['aws_access_key_id']
        aws_secret_access_key = c['s3']['aws_secret_access_key']

        s3 = boto3.resource('s3', aws_access_key_id=aws_access_key_id, aws_secret_access_key=aws_secret_access_key)
        s3_object_key = f"{self.s3_object_prefix}/{self.config}"
        s3.Bucket(s3_bucket_name).put_object(Key=s3_object_key, Body=config)
        print(f"Successfully uploaded configuration to S3 bucket {s3_bucket_name}")
        self.config = s3_object_key
    except Exception as e:
        print(f"Error occurred while uploading configuration for {self.hostname}: {e}")
```

Розробка Collector

Device.disconnect()

Метод перевіряє, чи є встановленим SSH з'єднання з девайсом, і у разі його наявності, закриває його.

```
def disconnect(self) -> None:
    try:
        print(f"{self.hostname} successfully disconnected")
        if self.ssh:
            self.ssh.close()
            self.ssh = None
            print(f"{self.hostname} successfully disconnected")
    except Exception as e:
        print(f"Error occurred while disconnecting from {self.hostname}: {e}")
```

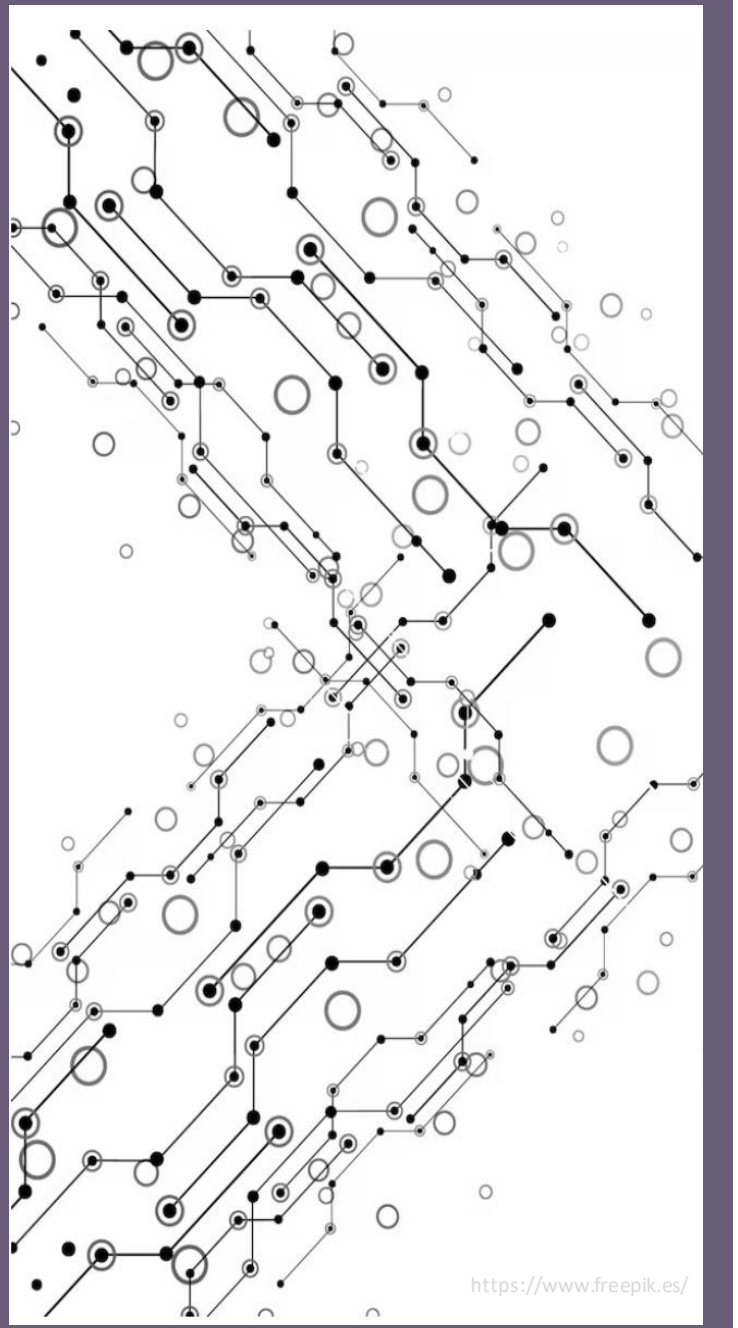
РОЗРОБКА КОМПОНЕНТУ

В результаті розробки:

- ❑ **створено компонент «Колектор» та протестовано його роботу;**
- ❑ **розроблено код, який реалізує підключення компоненту «Колектор» до потенційних девайсів.**

При розробці коду були використані підходи ООП та виконано:

- ➢ робота з БД (підключення, виконання запитів, обробку даних),
- ➢ робота з підключенням SSH до пристроїв,
- ➢ робота з **AWS SDK boto3** для підключення та роботи з сервісами AWS,
- ➢ бібліотека **asyncio** для асинхронного виконання завдань,
- ➢ робота з аргументами командного рядка.



ВИСНОВОК

Під час написання даної роботи було виконано наступні завдання:

- визначено основні завдання СККККМ та обґрунтовано важливість використання даних систем для підвищення надійності роботи мережі;
- проведено порівняльний аналіз різних архітектурних шаблонів для використання у розробці СККККМ;
- розглянуто різні платформи для побудови СККККМ – їх переваги та недоліки у контексті задач системи;
- розглянуто існуючі продукти для реалізації компонентів СККККМ системи;
- проведено структурну розробку власної СККККМ системи та описано функціонування та особливості кожного змодельованого компоненту системи;
- розроблено та протестовано один із компонентів системи – Колектор.

Досягнуто таких результатів:

- розроблено структурну схему СККККМ, яка задовольняє основним стандартним вимогам до неї: надійність контролю, ефективний моніторинг стану пристроїв та їх конфігурацій, структуроване зберігання даних;
- проведено практичну розробку одного з компонентів структурної схеми СККККМ.

Розроблені структурна схема СККККМ та компонент можуть бути застосовані на підприємствах для централізованого управління конфігураціями використовуваних девайсів та оптимізації роботи системних інженерів.

ДЯКУЮ ЗА УВАГУ