МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія централізованого управління доступом віддаленних систем»

Завідувач кафедри **Довбиш А.С.**
Керівник роботи **Проценко О.Б.**
Студент гр. ІН.мз-01с **Горбась І.В.**

СУМИ 2021

Сумський державний університет
(назва вузу)

Факультет *Еліт* Кафедра *Комп'ютерних наук*
Спеціальність *«122 -Комп'ютерні науки»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Горбасю Івану Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)
Інформаційна технологія централізованого управління доступом віддалених систем

затверджую наказом по інституту від "____" _____ 20 ___ р. №_____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми. Постановка задачі. 2) Вибір методів та інструментів для вирішення задачі. 3) Проектування компонентів додатку 4) Розробка програмного додатку

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
	Аналіз проблеми. Постановка задачі		
	Вибір методів та інструментів для вирішення задачі		
	Проектування компонентів додатку		
	Розробка програмного додатку		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 66 стор., 7 рис., 4 додатки, 15 джерел.

Об’єкт дослідження — фільтрація доступів до локальної мережі та мережі інтернет.

Мета роботи — розробка централізованої системи управління доступами до мережі, використовуючи технології розробки монолітних програмних додатків за REST архітектурою і написання bash-сервісу для роботи у якості клієнта.

Методи дослідження — метод аналітичного огляду, метод порівняння та аналогій, метод моделювання, методи розробки, що базуються на мові програмування javascript та скриптовій мові bash.

Результати — отримано гнучку систему, що дає змогу: до управління усіма пристроями у віддаленій мережі, що отримують IP-адресу за DHCP; збирати та зберігати дані щодо пристроїв користувачів у одному місці, а також мати можливість швидко керувати їх доступами до локальної мережі і мережі інтернет через зручний UI; залишати коментар, щодо певного пристрою і точно знати його володаря; делегувати відповідальність за надання/заборону доступів до мережі особам без вищої технічної освіти.

WEB APPLICATION, MODULAR MONOLITH, REST, NODE.JS, EXPRESS.JS, VUE.JS, MONGODB, OPENWRT, BASH, SERVICES, HTTP

ЗМІСТ

ВСТУП

Інтернет-безпека – це безпека дій та транзакцій, що здійснюються в Інтернеті. Проводячи значний час у мережі, можна зіткнутися з такими загрозами інтернет-безпеці як злом, віруси та зловмисні програми або, наприклад, крадіжка персональних даних. А шляхи їх розповсюдження можуть бути доволі різними.

Здійснити атаку, з метою заволодіння жаданою інформацією, зловмисник може скомпрометувавши пошту жертви за допомогою фішингових листів від колеги, скористатися прийомами соціальної інженерії у тандемі з таргетованою рекламою. Це далеко не усі місця з підвищеною небезпекою, де кожному користувачу мережі інтернет потрібно бути насторожі.

Більш того нанести атаку можна і з локальної мережі. Для цього зловмисник скористається поточними уразливостями операційних систем користувачів чи десктопних додатків, для підвищення привілеїв у поточній інфраструктурі. Або ж вдасться проникнути до корпоративної мережі внаслідок недостатньої системи захисту точки доступу WIFI.

Тому у зв’язку з постійною актуальністю проблеми безпеки у мережі, робота буде присвячена проблемі захисту безпеки користувачів, їх пристроїв, серверного та мережевого обладнання у локальних (LAN) та віртуальних локальних (VLAN) мережах. Проблема полягає у недостатній ефективності механізму запобігання проникнень з MAC-фільтром. Гнучкості списків доступів і правил firewall, але неможливості їх віддаленого налаштування. Відсутності посередника між складною системою доступів та користувачем без спеціальної технічної освіти.

До цього широкого списку проблем можна додати і складний процес валідації вхідних даних, де і досвідчений системний інженер здатен припустися помилки. Проблему автоматизації та централізованості складної інфраструктури з великої кількості кінцевих мережевих пристроїв.

Метою роботи є розробка системи управління доступами на віддалених системах. Розробка має забезпечити постійну роботу механізму обмеження доступу на кінцевому мережевому пристрої, додавши ряд правил в iptables за MAC-адресою пристрою. Централізувати місце збереження метаданих щодо пристрою, а саме його: типу, власника, дати додання й останніх змін. Та створити зручний користувацький інтерфейс по роботі із девайсами, що дозволить повною мірою делегувати відповідальність за певною віддаленою філією на вповноважену особу.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД СИСТЕМ ПРИЗНАЧЕНИХ ДЛЯ ПОБУДОВИ МЕРЕЖЕВОЇ ТА ЛОКАЛЬНОЇ СИСТЕМ БЕЗПЕКИ

1.1 Огляд проблемної області

Процес побудови локальної мережі — це лише третина усієї роботи. Далі проведення відповідних налаштувань, і нарешті – забезпечення безпеки локальної мережі. Тому кожному підприємству необхідно зберегти приватність відомостей, що передаються різними каналами або знаходяться на зберіганні всередині самої корпоративної системи. З плиином часу зростають вимоги як до швидкості передачі інформації (зменшення часу доступу до мережевих ресурсів, що знаходяться в різних географічних поясах), так і до надійності та захисту даних, що передаються. Модульність побудови апаратно-програмного забезпечення, новітні технології розвитку мережевих технологій і вирішують ці завдання.

Корпоративною мережею називається мережа, що охоплює велику кількість комп'ютерів і розташована в межах одного підприємства.[1]

У зв'язку з тим, що сучасні підприємства та їхні філії можуть територіально охоплювати різні міста, країни і навіть континенти, їх корпоративні мережі складаються з десятків і сотень локальних мереж, що включають десятки тисяч комп'ютерів і сотні серверів.

Задля безпеки загального потрібно почати з безпеки конкретних кінцевих вузлів. Комп'ютерна мережа, розгорнута всередині будівлі і працює під єдиною адміністративною системою, зазвичай називається локальною обчислювальною мережею (LAN). Зазвичай LAN охоплює офіси організації, школи, коледжі чи університети. Кількість систем, підключених до локальної мережі, може змінюватись від двох до 16 мільйонів. LAN забезпечує корисний спосіб спільного використання ресурсів між кінцевими користувачами. Такі ресурси, як принтери, сканери та Інтернет, легко поширюються серед комп'ютерів.

Локальна мережа потребує безпеки, бо містить низку життєво необхідних вузлів для підтримки мережі у функціонуючому вигляді. Це можуть бути сервери, системи відеоспостереженні, системи керування управління доступами, мережеві телефони та принтери, точки доступа, персональні комп’ютери кінцевих користувачів, а також інше мережеве обладнання.

Існує кілька різновидів серверів у локальній мережі, зокрема:

- Поштовий сервер - управляє передачею електронних повідомлень між користувачами мережі; скомпрометований поштова скринька може призвести до значних проблем як, наприклад, використання у якості спам відправника, викриття конфіденційної інформації щодо співробітників компанії, структури компанії або політики безпеки, відомості щодо облікових записів у інших мережевих ресурсах. Що вже казати про злом цілого поштового сервера.

- Файловий сервер — керує створенням та використанням інформаційних ресурсів локальної мережі, включаючи доступ до її баз даних та окремих файлів, а також їх захист. Наприклад, при роботі з тестовим редактором файл зберігається на файловому сервері, а завантажуватися буде в пам'ять вашого комп'ютера; викриття особистих файлів, службових документів, правової та юридичної інформації – це все те, що може трапитися у разі злому локального мережевого сховища.

- Сервери додатків - управляє роботою локальної мережі при виконанні будь-яких прикладних завдань. Прикладами такого роду завдань можуть бути: забезпечення зв'язку з іншими локальними та/або телекомунікаційними системами, колективне використання друкувальних пристроїв тощо; програмні мережеві АТС, системи відеоспостереження та системи керування управління доступами – це все чудові приклади серверів додатків, їх працездатність значною мірою впливає на роботу підприємства, тому їх вихід з ладу може призвести до значних втрат.

- Проксі-сервери - може зберігати інформацію, що часто запитується, в кеш-пам'яті на локальному диску, швидко доставляючи її користувачам без повторного звернення до Інтернету; крім того, проксі-сервери призначені також для більш комфортної фільтрації трафіку з локальної мережі в інтернет та навпаки. Злом проксі означатиме вільне проходження трафіку, без можливості його фільтрації брандмауером, а значить будь-що може бути завантажено на користувацькі комп’ютери.

- Принт-сервери – дозволяють усім підключеним до мережі комп'ютерам видруковувати документи на одному або кількох спільних принтерах. У цьому випадку відпадає необхідність комплектувати кожен комп'ютер власним принтером. Крім того, беручи на себе всі турботи про виведення документів на друк, принт-сервер звільняє комп'ютери для іншої роботи. Злом принт-серверів, або мережевих принтерів може бути причиною витоку конфіденційної інформації, за допомогою можливості перехоплення, відправлених на друк документів. Крім цього зловмисник може використовувати уражений принт-сервер у якості машини у віддаленій мережі через яку можна перенаправляти запити.

1.2 Огляд подібних рішень

Локальні мережі, розділені на логічні сегменти, називають комутованими локальними мережами.

Комутатор - це пристрій, що функціонує на другому/третьому рівні еталонної моделі ISO/OSI та призначений для об'єднання сегментів мережі, що працюють на основі одного протоколу канального/мережевого рівня. Комутатор спрямовує трафік лише через один порт, необхідний для досягнення місця призначення. Нище приведена схема з класифікації комутаторів рис.1.1.

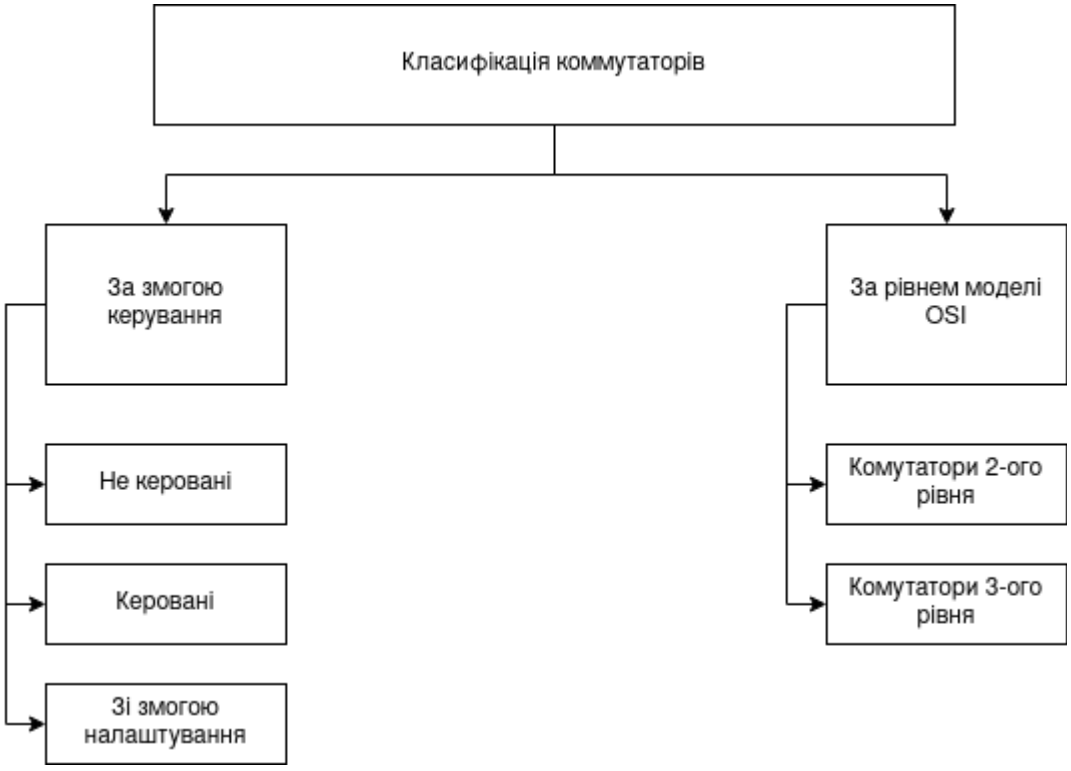

Рисунок 1.1 - Класифікація комутаторів

Комутатор рівня 2 призначений для з'єднання кількох пристроїв локальної обчислювальної мережі (LAN) або кількох сегментів цієї мережі. Обробляє та реєструє MAC–адреси кадрів, що надходять, здійснює фізичну адресацію та управління потоком даних.

Комутатори 3 рівні фактично є маршрутизаторами, які реалізують механізми маршрутизації з використанням протоколів маршрутизації за допомогою спеціалізованих апаратних засобів. Використовуються в локальних мережах для забезпечення швидкодійної передачі). Маршрутизатори використовуються при організації зовнішнього зв'язку.

Для управління доступами в локальній мережі використовують такі технології:

1. *Обмеження кількості керуючих комп'ютерів*

Сучасні комутатори дозволяють задати конкретні комп'ютери (IP-адреси), з яких відбуватиметься налаштування даних комутаторів за Web- або

Telnet-інтерфейсами або через протокол SNMP. Спроба інших комп'ютерів підключитися до комутаторів для керування ними буде відхилена.

2. *Налаштування безпеки індивідуального порту*

Ця функція дозволяє:

- заблокувати подальше оновлення таблиці комутатора - якщо конфігурація мережі більше не зміняться. Комутатор відкидатиме всі пакети, які надходять із невідомих адрес. Ви можете вибрати дію для конкретних портів.
- встановити максимальну кількість MAC-адрес для прив'язки до конкретного порту.

3. *Функція Port Security*

Port Security - функція комутатора, що дозволяє вказати MAC-адреси хостів, яким можна передавати дані через порт. Після цього порт не передає пакети, якщо MAC-адреса відправника не вказана як дозволена. Крім того, можна вказувати не конкретні MAC-адреси, дозволені на порті комутатора, а обмежити кількість MAC-адрес, яким можна передавати трафік через порт.

Використовується для запобігання:

- несанкціонованої зміни MAC-адреси мережного пристрою або підключення до мережі;
- атак, вкладених у переповнення таблиці комутації.

4. *Фільтрування MAC-адрес*

Додатково можна налаштувати комутатор так, щоб він не приймав пакети з певною MAC-адресою (як одержувача, так і відправника). І тому служить таблиця фільтрації трафіку (Filtering Table). Фактично ця таблиця є «чорним списком». Після отримання пакета комутатор зчитує обидві апаратні адреси, як одержувача, так і відправника. Якщо хоча б один з них міститься в таблиці фільтрації, пакет відкидається.

5. *Технологія фільтрації IP-MAC Binding*

Основне призначення цієї технології – це обмежити доступ до комутатора певної кількості комп'ютерів. Для цього індивідуально для кожного бажаного порту створюється таблиця відповідності IP- та MAC-адрес. Далі комутатор пропускатиме лише пакети із зазначеними MAC-адресами і тільки в тому випадку, якщо істинно відповідність IP- та MAC-адреси.

Функція IP-MAC-Port Binding (IMPB), реалізована в комутаторах D-link, дозволяє контролювати доступ комп'ютерів у мережу на основі їх IP- та MAC-адрес, а також порту підключення.

Істотні відмінності цієї технології від технології фільтрації MAC-адрес:

- фільтрація працює всіх портах, а даної технології можна налаштовувати кожен порт окремо;
- фільтрація містить «чорний» список, тобто працює за принципом: не пропускати пакети, MAC-адреси яких містяться в таблиці фільтрації. Ця технологія, навпаки, містить «білий» список, тобто пропускає лише ті пакети, MAC-адреси яких містяться у створених списках;
- на відміну від фільтрації технологія IP-MAC Binding перевіряє не тільки MAC адресу джерела пакета, але і його IP-адресу.

6. *Списки контролю доступу (Access Control Lists)*

Списки контролю доступу забезпечують обмеження проходження трафіку через комутатор. Фактично технологія ACL реалізує на комутаторах 3 рівні повноцінний фільтр пакетів. Прийом пакетів або відмова в прийомі ґрунтується на певних ознаках, які містять пакет:

- IP- та MAC-адреси джерела та приймача;
- номер VLAN;
- номер порту TCP чи UDP;
- тип ICMP-повідомлення.

ACL є послідовністю умов перевірки параметрів пакетів даних. Коли повідомлення надходять на вхідний порт, комутатор перевіряє параметри пакетів даних на збіги з критеріями фільтрації, визначеними ACL, і виконує над пакетами даних одну з дій: Permit (Дозволити) або Deny (Заборонити). Критерії фільтрації можуть бути визначені на основі інформації, що міститься в пакеті.

Ключовим поняттям у цій технології є поняття профілю доступу – це набір ознак, що містить пакет, з певними значеннями. Кожен профіль доступ має свій унікальний номер у межах комутатора.

7. Сегментація трафіку (Traffic Segmentation)

Сегментація трафіку служить для розмежування портів на канальному рівні. Ця функція дозволяє настроювати порти або групу портів таким чином, щоб вони були ізольовані один від одного, але водночас мали доступ до портів, що розділяються, які використовуються для підключення серверів або магістралі мережі провайдера. Дуже важливою є можливість одночасного використання цієї функції з віртуальними мережами (VLAN), що дозволяє проводити подальше розмежування прав.

Ця технологія схожа з технологією VLAN, але є більш обмеженою за функціональністю. До переваг технології можна віднести:

- простота налаштування;
- нижче завантаження процесора комутатора в порівнянні з VLAN

Протокол IEEE 802.1x

Протокол IEEE 802.1x є механізмом безпеки, що забезпечує автентифікацію та авторизацію користувачів і тим самим обмежує доступ дротових або бездротових пристроїв до локальної мережі. Робота протоколу базується на клієнт-серверній моделі контролю доступу.

Рисунок 1.2 - Схема IEEE 802.1 x протоколу з використанням Radius

Як сервер аутентифікації використовується RADIUS-сервер. При цьому весь процес аутентифікації користувача проводиться у провідних мережах на основі протоколу EAPOL (Extensible Authentication Protocol over LAN), бездротових – на основі протоколу EAPOW (Extensible Authentication Protocol over Wireless). Рисунок 1.2. Доки клієнт не буде автентифікований, протокол IEEE 802.1x пропускатиме через мережевий порт тільки трафік протоколу EAPOL. Після успішної аутентифікації звичайний трафік пропускатиметься через порт. Робота протоколу IEEE 802.1x ґрунтується на трьох компонентах (рисунок 1.2).

У якості ролей пристроїв виступають: «Заявник», «Аутентифікатор» та «Сервер аутентифікації».

Заявник – це робоча станція, яка запитує доступ до локальної мережі та сервісів комутатора та відповідає на запити комутатора. На робочій станції має

бути встановлено клієнтське програмне забезпечення, що реалізує протокол 802.1x (в родині ОС Microsoft Windows, починаючи з версії XP - це програмне забезпечення є вбудованим).

Сервер аутентифікації виконує фактичну аутентифікацію клієнта, перевіряючи справжність клієнта та інформуючи комутатор, надавати чи ні клієнту доступом до локальної мережі. Комутатор (також називається аутентифікатор) управляє фізичним доступом до мережі, виходячи з статусу аутентифікації клієнта.

Комутатор(Аутентифікатор) працює як посередник між клієнтом та сервером аутентифікації, отримуючи запит на автентифікацію від клієнта, перевіряючи цю інформацію за допомогою сервера аутентифікації, і пересилаючи відповідь клієнту. ПЗ комутатора включає клієнта RADIUS, який відповідає за інкапсуляцію та деінкапсуляцію кадрів EAP та взаємодію із сервером аутентифікації.

Протокол IEEE 802.1x надає два методи контролю доступу до мережі:

1. На основі портів (Port-Based Access Control). При використанні цього методу достатньо, щоб лише один будь-який користувач, підключений до порту комутатора, був авторизований. Тоді порт перейде в авторизований стан і доступ до мережі отримають будь-які користувачі, підключеного до цього порту.

2. На основі MAC-адрес (MAC-Based Access Control). При використанні даного методу під час аутентифікації також враховується MAC-адреса клієнта, підключеного до порту, і порт авторизується тільки для клієнта з конкретною MAC-адресою.

1.3 Постановка задачі

Певно що усі перечислені технології виконують свою задачу. Але що якщо з’являється потреба в управлінні налаштунками доступів мережевого пристрою (комутатора або маршрутизатора) швидко, виконуючи низку однотипних задач на великій кількості віддалених пристроїв, або взагалі делегувати відповідальність за налаштунки прав для пристроїв на особу без спеціальної технічної освіти.

Метою роботи є готове програмне забезпечення (ПЗ), яке має вирішувати ряд поставлених задач:

- розробка інформаційної системи, що легко розширюється та доповнюється;
- гнучке управління доступами для усіх пристроїв у локальній мережі, що отримують IP-адресу за DHCP за маком пристрою;
- централізована система для управління великою кількістю мережевих пристроїв одночасно;
- єдина база з MAC-адресами пристроїв, можливість залишити коментар щодо типу, посади та ПІБ власника пристрою;
- можливість управління списками дозволених/заборонених пристроїв особою без спеціальної технічної освіти.

2 ВИБІР МЕТОДІВ ТА ІНСТРУМЕНТІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ

2.1 Вибір архітектури для розробки

Для розробки централізованої системи для управління доступами віддаленних систем, потрібно почати з проектування. Перше на що варто звернути увагу на цьому етапі — який тип архітектури мусить мати веб-додаток. Варіантами рішення для вибору архітектури додатку у сучасному додатку можуть бути «Модульний моноліт» або «Мікросервісна архітектура».

Моноліт

Найбільш поширений приклад, який спадає на думку при обговоренні монолітів, — система, в якій весь код розгортається як один процес, як показано на рис. 2.1. Можна мати кілька екземплярів цього процесу задля забезпечення робастності або масштабування, але в основному весь код упакований в один єдиний процес. Насправді як такі ці однопроцесні системи можуть бути простими розподіленими системами, оскільки вони майже завжди, зрештою, займаються читанням даних з бази даних або їх збереженням у ній. [4].

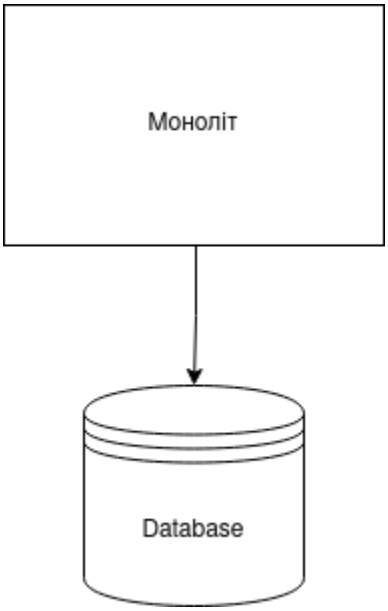

Рисунок 2.1 - Принцип взаємодії ПЗ з монолітною архітектурою з базою даних

Модульний моноліт є варіацією як підмножина однопроцесного моноліту: один процес складається з окремих модулів, над кожним з яких можна працювати незалежно, але вони, як і раніше, повинні об'єднуватися для розгортання, як показано на рис. 2.2.

Для багатьох організацій модульний моноліт - відмінний варіант вибору. Якщо контури модулів чітко визначені, він забезпечує високий рівень паралельності роботи, але оминає труднощі більш розподіленої архітектури, заснованої на мікрослужбах, поруч із набагато простими питаннями розгортання. [13]

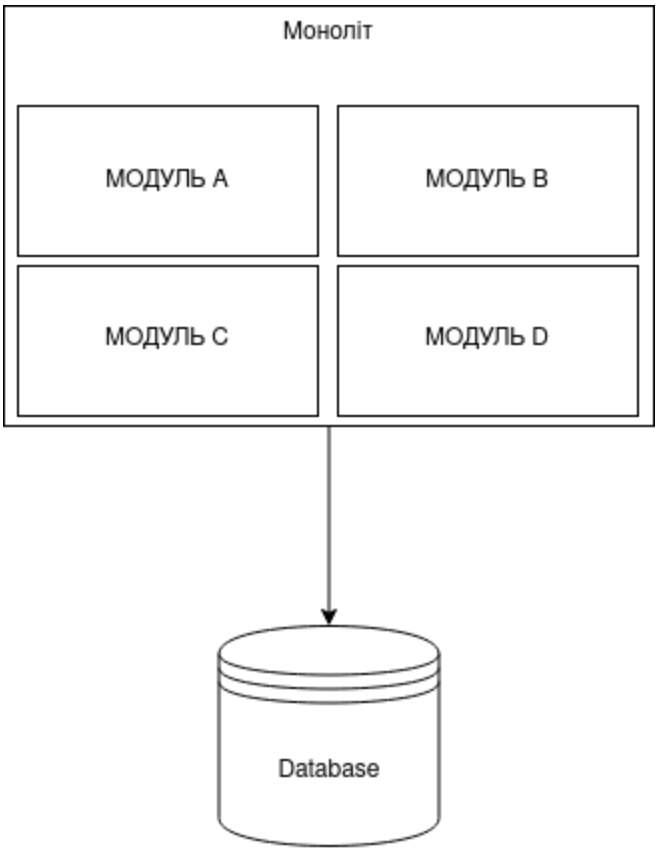

Рисунок 2.2 - Принцип взаємодії ПЗ з модульно-монолітною архітектурою з базою даних

Переваги:

- Швидко та просто розробляти.
- Легше налагоджувати (одна транзакція, один процес).
- Узгодженість даних (одна база).
- Не потрібна суперкваліфікована команда.
- Дешево.
- Легко деплоїти.

Недоліки:

- Ентропія – згодом проект перетворюється на legacy
- Велика «грудка бруду».
- Спагетті-код

Мікросервіси

Мікрослужби — це служби, що незалежно розгортаються, моделюються навколо бізнес-доменів. Вони спілкуються один з одним через мережі та пропонують на вибір цілу низку варіантів архітектури для вирішення завдань, з якими можна зіткнутися. Звідси випливає, що архітектура на основі мікрослужб базується на численних мікрослужбах.[4].

З технологічної точки зору мікрослужби виставляють назовні можливості бізнесу, які вони інкапсулюють через одну або кілька кінцевих точок у мережі. Мікрослужби взаємодіють один з одним через ці мережі, що робить їх різновидом розподіленої системи. Вони також інкапсулюють зберігання та вилучення даних, надаючи дані через чітко визначені інтерфейси. І тому бази даних приховані в межах контуру служби.

Одною з основних особливостей при роботі з мікросервісами це те, що мікрослужби не повинні використовувати бази даних спільно. Якщо одна служба хоче звернутися до даних, що зберігаються в іншій службі, то вона повинна піти і запросити цю службу необхідні їй дані. Такий підхід дає тій службі можливість вирішувати, що є спільним, що прихованим. Він також дозволяє службі відображати деталі внутрішньої імплементації, які можуть змінитися з різних довільних причин, більш чіткий публічний контракт, що забезпечує стабільні інтерфейси між службами. [3]

Переваги:

- Вирішення проблем моноліту.
- Фізична ізоляція мікросервісів.
- Незалежна розробка мікросервісів.
- Незалежне масштабування.
- Незалежний деплой

Недоліки:

- Часткова відмова.
- Ваще у написанні та при роботі з іншими командами.
- Труднощі при тестуванні.

- Ваще у експлуатації.
- Більше обчислювальних потужностей або фізичних машин

Зважаючи на що модульний підхід у розробці ваще імплементувати, довше, а його переваги перед монолітним, у даному проєкті, не суттєво важливі, то для реалізації системи буде обрано ***монолітний підхід*** розробки.

2.2 Вибір способу відправки даних по протоколу HTTP

WEB заснований на дуже простій клієнт-серверній архітектурі, яку можна узагальнити наступним чином: клієнт (зазвичай веб-браузер) відправляє запит на сервер (в основному веб-сервер, такий як Apache, Nginx, IIS, Tomcat і т.д.) , використовуючи протокол HTTP. Сервер відповідає на запит, використовуючи той самий протокол.[5].

Серед сучасних способів у передачі даних з клієнта на сервер та навпаки виділяють різні підходи. [2]

gRPC

Віддалений виклик процедури (Remote Procedure Call) - це специфікація, яка дозволяє віддалено виконувати функцію в іншому контексті. RPC розширює поняття локального виклику процедури, але поміщає його у контекст HTTP API.

Клієнт викликає віддалену процедуру, серіалізує параметри та додаткову інформацію у повідомленні та надсилає це повідомлення на сервер. Отримавши повідомлення, сервер десеріалізує його вміст, виконує запитану операцію та надсилає результат назад клієнту. Стаб сервера та стаб клієнта беруть на себе серіалізацію та десеріалізацію параметрів.

Завдяки підтримці балансування навантаження, трасування, перевірки працездатності та автентифікації gRPC добре підходить для мікросервісів.

Переваги:

- Простота у використанні.
- Легкість у додаванні нових ендпоінтів.
- Висока ефективність, за рахунок оптимізації мережевого рівня.

Недоліки:

- Низький рівень виявності.
- Проблема створення нових функцій на базі запитів.

REST API

RESTful API — це API, який базується на принципах REST (REpresentational State Transfer). А RESTful API надає програмістам зручні функції програмування для передачі даних назад і вперед між веб-ресурсами, що знаходяться в різних місцях мережі.

Важливо знати, що запит складається з чотирьох речей:

- The endpoint.
- The method.
- The headers.
- The data (or body).

Заголовки та параметри також важливі в методах HTTP HTTP-запиту RESTful API, оскільки вони містять важливу інформацію про ідентифікатор щодо метаданих запиту, авторизації, єдиного ідентифікатора ресурсу (URI), кешування, файлів cookie та більше [6]. Існують заголовки запиту та заголовки відповіді, кожен із власною інформацією з’єднання HTTP та кодами стану.

Для того, щоб API вважався RESTful, він повинен відповідати наступним критеріям:

1. Архітектура клієнт-сервер, що складається з клієнтів, серверів і ресурсів, із запитами, які керуються через HTTP.

2. Зв’язок клієнт-сервер без стану, що означає, що інформація про клієнта не зберігається між запитами на отримання, і кожен запит є окремим і не підключеним.
3. Дані з кешуванням, які спрощують взаємодію клієнт-сервер.
4. Єдиний інтерфейс між компонентами, щоб інформація передавалася у стандартній формі. Для цього потрібно

◦ запитувані ресурси є ідентифікованими та відокремленими від представлень, надісланих клієнту.

◦ клієнт може маніпулювати ресурсами за допомогою представлення, яке вони отримують, оскільки подання містить достатньо інформації для цього.

◦ самоописні повідомлення, що повертаються клієнту, мають достатньо інформації, щоб описати, як клієнт повинен її обробити.гіпертекст/гіпермедіа доступний, що означає, що після доступу до ресурсу клієнт повинен мати можливість використовувати гіперпосилання, щоб знайти всі інші доступні на даний момент дії, які він може виконати.

5. Багатошарова система, яка організовує кожен тип серверів (відповідальних за безпеку, балансування навантаження тощо), передбачала отримання запитуваної інформації в ієрархії, невидимі для клієнта.
6. Code-on-demand (опціонально): можливість надсилати виконуваний код із сервера клієнту за запитом, що розширює функціональність клієнта.

Переваги:

- Продуктивність.
- Масштабованість.
- Гнучкість до змін.
- Відмовостійкість.
- Простота підтримки.
- Кешування.

Недоліки:

- Немає «правильної» моделі при практичній реалізації.
- Перевантаження або нестача даних у відповіді.

GraphQL

GraphQL — це мова запитів через API. Це також середовище виконання для виконання запитів з даними. Служба GraphQL не залежить від транспорту, але зазвичай обслуговується через HTTP.[7]. Синтаксисом GraphQL описується, як зробити точний запит даних. Реалізація GraphQL варта того, якщо задіяна модель даних програми з великою кількістю складних сутностей, що посилаються одна на одну.

GraphQL починається з побудови схеми, яка є описом всіх запитів, які можливо зробити в API GraphQL, і всіх типів даних, які вони повертають.

Клієнт, що має схему перед відправкою запиту, може перевірити свій запит і переконатися, що сервер зможе відповісти на нього. Дістався бекенда, операція GraphQL інтерпретується по всій схемі і дозволяється за допомогою даних для фронтенду. Відправивши один масивний запит на сервер, API повертає відповідь у форматі JSON з тими даними, які ми запросили.

Недоліки продуктивності є одним із недоліків, на який часто посилаються розробники. GraphQL не має вбудованих функцій кешування, і запит корисного навантаження може призвести до додаткових затримок.

Переваги:

- Типізована схема запитів.
- Відсутність версійності, що забезпечує безперервний доступ до нових функцій та сприяє більш чистому серверному коду.
- Детальна обробка помилок.
- Гнучкість.

Недоліки:

- Продуктивність.
- Відсутність власної технології кешування.

Сучасний підхід до побудови запитів за схемами і жорсткою типізацією GraphQL, за своєю сутністю підходить більше у якості інструмента розробника складних систем та мікросервісів. GraphQL також чудово зможе підійти для розробки для мобільних пристроїв, де буде важлива корисне навантаження на мережу.

У випадку з розробкою системи для віддаленого управління доступами, оновлення інформації на віддалених мережевих пристроях має відбуватися з певною періодичніст. Щоб не навантажувати кожного разу базу однотиповими запитами, кращим з варіантів буде реалізація процессу кешування. Тому у якості способу відправки даних мережею буде обрано ***REST***.

2.3 Вибір фреймворку для роботи з серверною та клієнтськими частинами. Вибір СУБД

Обрати мову програмування досить легко, бо є досить великий спектр сучасних мов програмування, що розв’язують поставлену задачу.

Backend

1. Node (Express)

Node.js – це програмне середовище, призначене для запуску та виконання JavaScript-програм поза браузером. Завдяки цьому середовищу JavaScript перетворився з вузькоспеціалізованої мови на мову загального призначення. І зараз JavaScript використовується не тільки для створення frontend-частини веб-застосунків, але й для розробки серверної частини, мобільних і десктопних програм, і навіть програм для мікроконтролерів [9].

Node.js у своїй основі використовує той же двигун - V8, який виконує JS-код у браузері Chrome. Цей двигун розробляється та підтримується компанією Google. Основне завдання двигуна - компіляція JavaScript-коду в машинний код. Також, до складу Node.js входить бібліотека, написана на C, яка

називається libuv. Вона дозволяє працювати з операційною системою — читати та записувати файли, надсилати та приймати дані по мережі тощо. Ця бібліотека надає можливість використовувати асинхронні операції введення або виведення [8].

Переваги Express:

- Швидка розробка з використанням Node.js.
- Численне ком'юніті, тому велика кількість існуючих бібліотек.
- Інтегрується з релятивними та не релятивними БД.
- Підтримка template engines.
- Система фільтрів (middleware).

Недоліки Express:

- Недостатньо структурована.
- Не надає рішень у сфері безпеки.

2. Ruby (Ruby on Rails)

Ruby on Rails - один із найпопулярніших фреймфорків для розробки сайтів та веб-додатків. Мова програмування Ruby разом із фреймворком Rails значно спрощує процес розробки.

Що робить Rails таким чудовим? Перш за все, Ruby on Rails на 100 відсотків є відкритим вихідним кодом, доступним під дозволеною ліцензією MIT, і, як наслідок, він також нічого не коштує для завантаження та використання. Rails також багато в чому завдячує своїм успіхом своєму елегантному та компактному дизайну; Використовуючи пластичність основної мови Ruby, Rails ефективно створює мову, специфічну для домену, для написання веб-додатків. У результаті багато поширених завдань веб-програмування, таких як генерація HTML, створення моделей даних і маршрутизація URL-адрес, легко виконуються за допомогою Rails, а отриманий код програми є стислим і читабельним.

Rails також швидко адаптується до нових розробок у веб-технологіях та дизайні фреймворків [11].

Переваги RoR:

- Обробка вхідних даних.
- Gems Authlogic та Devise.
- Потоки і асинхронність.
- Численне ком’юніті, тому велика кількість існуючих бібліотек.

Недоліки RoR:

- Ресурсоємкий у разі постійного з’єднання для кожного з користувачів.
- Не підходить для малих за розміром проєктів.

3. Python (Django)

Django — це сучасні стандарти веб-розробки: схема «модель-представлення-контролер», використання міграцій для внесення змін до бази даних та принцип «DRY» [12].

Для створення веб-сайту з Django, достатньо його одного. Типові завдання, на зразок з'єднання з базою даних, обробки даних, отриманих від користувача, збереження вивантажених користувачем файлів, він виконує самостійно. А ще він надає велику кількість готових бібліотек для роботи з: графічними об’єктами; аутентифікацією через соціальні мережі; капчею; та збором сміття.

Переваги Django:

- Інтегрується з релятивними та не релятивними БД.
- ORM, API доступу до БД із підтримкою транзакцій.
- Система кешування.
- Інтернаціоналізація.
- generic views - шаблони функцій контролерів.
- Система фільтрів (middleware).
- Вбудована автоматична документація

Недоліки Django:

- Погано показує себе при роботі з великими проєктами, або навпаки дуже малими, що не передбачає роботи з БД, файловою структурою т.д.
- Надлишкова автоматизація.

4. PHP (Laravel)

PHP-фреймворк - це програмна платформа для створення веб-додатків високої якості, при цьому заощаджуючи значний час на саму розробку.

Laravel – один з найпопулярніших PHP-фреймворків для написання веб-додатків, створений на основі Symfony, який використовує архітектурну модель Model View Controller (MVC) з відкритим вихідним кодом. Laravel є найсильнішим суперником в екосистемі PHP просто тому, що він включає функції, необхідні для створення сучасних, підтримуваних, розподілених веб-додатків в реальному часі. Крім того, у нього є велика відеотека Laracasts, що містить понад 900 посібників.

Головними особливостями фреймворка є швидкість розробки та виконання, вирішує питання безпеки додатку, легко маштабується та підтримує великий функціонал «з коробки».

Переваги Laravel:

- MVC.
- Eloquent ORM для роботи з БД.
- Підтримка шаблонізаторів.
- PHP Unit-tests.
- Широкий спектр доступних модулів.
- Влаштовані функції безпеки.

Недоліки Laravel:

- Відстутність влаштованої підтримки.
- Не підходить для великих проєктів.

Оптимальним рішенням для вибору backend-фрейморка є ***Express.js***. Проєкт на express, легко маштабується, підтримує велику кількість готових рішень(бібліотек), а також здатен реалізувати REST підхід.

Frontend

Серед усіх сучасних фронтенд фреймворків найрозповсюдженішими є фреймворки написані мовою JavaScript. Google відповідає за розробку та підтримку ***Angular*** з міжнародною командою розробників. У випадку з ***React***, то міжнародна група, яка розробляє та підтримує, - це Facebook. А у разі з ***Vue*** це щось інше, оскільки він не має міжнародної компанії для його розробки та обслуговування, а, швидше, через команду співробітників по всьому світу, головним з яких є Еван Ю як його творець.

1. Angular.js

Angular - це фреймворк з багатьма вже вбудованими функціями, такими як: перевірка форми, відправка HTTP-запитів, маршрутизація, керування станом та багато іншого.

Через це багато хто називає Angular платформою, а не фреймворком через велику кількість вбудованих функцій, які він має. І не тільки через функції або можливості, які можна використовувати на рівні розробки, але і через його екосистеми, в якій його ядро (яке керується командою розробки та обслуговування Google) має додаткові інструменти, такі як: командний рядок, інтерфейси для управління та створення проектів, легко додати підтримку прогресивних веб-сторінок (PWA) і т. д.

Переваги:

- Реактивність.
- Простота у тестуванні.
- Модульність.
- Велика кількість готових функцій «з коробки».

- RXJS.
- Typescript.

Недоліки:

- Важкість у вивченні.
- Проблеми із сумісністю.

2. React

На відміну від Angular, React протилежний йому, він орієнтований на максимально мінімалістський підхід з акцентом на розробку інтерфейсу користувача, тому він вважається швидше бібліотекою, ніж самим фреймворком. Фактично React називає себе бібліотекою. Хоча він також має докладну документацію щодо використання його функцій, React має набагато менше вбудованих функцій, ніж Angular.

Таким чином, при розробці проекту ви зрештою використовуєте безліч сторонніх пакетів або залежностей, будь то для маршрутизації, обробки стану, відправки HTTP-запитів і т.д.

Це може мати плюси і мінуси, тому що багато хто віддає перевагу мінімалістичному React, а інші вважають, що у фреймворку вже є все для початку роботи. Незважаючи на це, спільнота React є широкою, тому, якщо виникають проблеми, ймовірно, існує пакет або залежність, які допоможуть їх вирішити.

Переваги:

- Легкість у вивченні.
- Virtual DOM.
- Легкість під час міграцій.
- Швидкість у роботі.

Недоліки:

- Мала кількість готових функцій «з коробки».
- Недостатня типізація.

3. Vue.js

Можна сказати, що Vue знаходиться між Angular та React, оскільки це фреймворк, який пропонує деякі вбудовані функції, але не так багато, як Angular. В основному він фокусується на вбудованих функціях, необхідних для написання коду, наприклад: маршрутизації та обробки стану, які підтримуються командою Vue, але інші функції, такі як перевірка форми, не інтегровані та залежать від стороннього пакета. Таким чином, це було б більше, ніж React, але менше, ніж Angular щодо вбудованих функцій.

Vue фокусується на коді, і, незважаючи на гарний інтерфейс командного рядка, у них не так багато інструментів, як у Angular, хоч і трохи краще, ніж у React.

Переваги:

- Оптимізованість обробки HTML-блоків.
- Підходить як для одностраничних так і для багатосторінкових додатків.
- Масштабування.
- Легкість і швидкість.
- Середня кількість готових функцій «з коробки».

Недоліки:

- Недостатня типізація.
- Мале ком’юніті і відсутність англомовної документації.

Усі фреймворки цього розділу мають свої переваги та недоліки. Вибір у конкретному випадку не залежить від роботи системи управління доступами, а слугує для створення у зручного UI для роботи з доданими пристроями. Функціонал буде реалізовано на ***Vue.js.***

DB

Під час розвитку IT-індустрії з’явився широкий спектр технологій для комерційних та не комерційних розробників. Серед широкого ряду існуючих баз даних, для конкретного проєкту можна обрати ***MongoDB***. Для цього є декілька підстав.

По-перше, MongoDB відноситься до не реляційних баз даних. Це значить, що вона не працює з таблицями. Структурно, MongoDB використовує JSON-подібні документи та схему бази даних.

По-друге, документи можуть бути перевірені в процесі оновлення або виконання вставок. Функції текстового пошуку покращено у порівнянні з попередніми версіями. Нова здатність часткового індексування може призвести до більш високої продуктивності зменшуючи розмір індексів.

А також, у якості платформи для розробки уже було обрано Node.js, а це свідчитиме про гарну сумісність, оскільки MongoDB була розроблена для асинхронної роботи з об'єктами JSON.

Крім того, з інтеграцією MongoDB у проєкт, можна буде скористатися ODM Mongoose, призначений для гнучкої роботи зі схемами та моделями. У процесі розробки це полегшить тестування компонентів і зможе допомогти розділити логіку обробки вводу і виводу даних.

2.4 Вибір технології для розробки сервісу на стороні маршрутизатора

Ринок мережевих пристроїв пропонує широкий асортимент товарів. Для розв’язання питання централізованого управління доступами повинна виконуватися певна умова. Доступ до внутрішньої структури має бути гнучким, та спроможним до запуску сценаріїв, написаних вручну, тобто бути opensourse.

Такі гіганти як cisco або MikroTik не підійдуть за означенням. Тому, з метою вибору оптимального рішення з попереднім налаштуванням маршрутизатора було обрано розробляти веб-сервіс на будь-якому роутері, що підтримує OpenWRT.

OpenWrt Project - це операційна система Linux, призначена для пристроїв, що вбудовуються. Замість того, щоб намагатися створити одну статичну прошивку, OpenWrt надає файлову систему з керуванням пакетами, що повністю записується. Це звільняє вас від вибору та налаштування програм, що надаються постачальником, і дозволяє налаштовувати пристрій за допомогою пакетів, які підходять для будь-якої програми. Для розробників OpenWrt – це фреймворк для створення програми без необхідності створювати на його основі повну прошивку; для користувачів це означає можливість повного настроювання та використання пристрою різними способами [13].

Отож, із боку віддаленого підрозділу знаходитиметься маршрутизатор з прошивкою OpenWRT, а тобто програмний сценарій можна буде написати на bash. Задача маршрутизатора відправляти запит на отримання поточних даних з серверу по API. Отримавши список пристроїв, яким дозволено/заборонено доступ до локальної мережі та мережі інтернет відбудеться етап обробки даних. За результатами буде створено і записано правила у таблицю iptables. У разі зміни даних попередні записи буде видалено і додано нові.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Розроблення схеми бази даних

Зважаючи на те, що у якості бази даних для проєкту було обрано mongodb – модулі будуть описані за допомогою програмного коду. Їх відношення виглядатимуть таким чином рис.3.1.

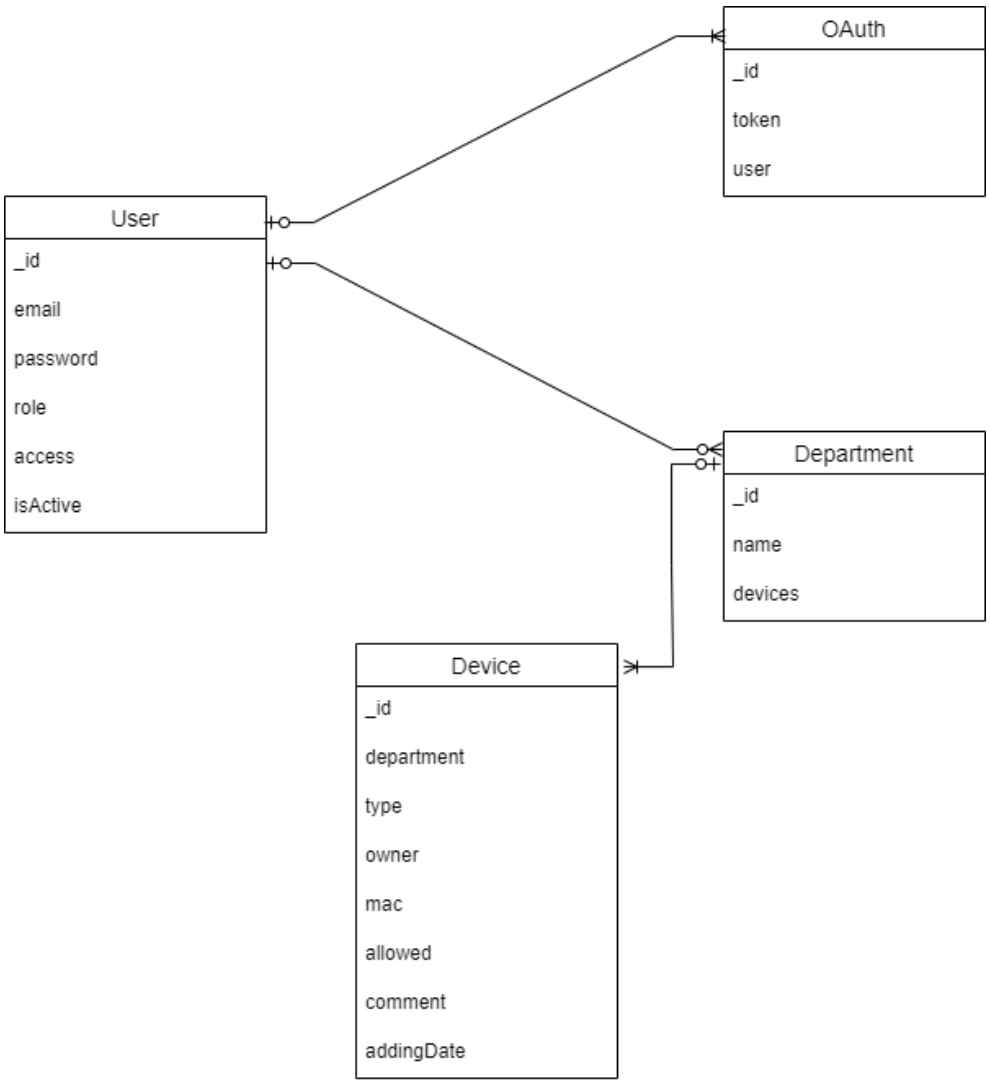

Рисунок 3.1 – Діаграма відношень колекцій між собою

Користувач у полі «access» містить масив унікальних ідентифікаторів з колекції департаменту, що свідчить про наявність доступу у даного користувача до даного департаменту.

Департамент у свою чергу має поле «devices», яке містить масив строкових «_id» з колекції Пристрій. У пристроїв є поле «department» де лежить «_id» департаменту, що вказує на його належність.

Також, виходячи з вимог, що характерні конкретній системі буде створено користувачів с такими ролями:

- “admin” (користувач з повними правами, та має необмежену кількість можливих доступних департаментів. Єдиний хто керує має права для створення нових користувачів, редагування їх прав до департаментів та створення нових департаментів);
- “regional” (користувач має доступ до декількох філіалів та може редагувати їх пристрої);
- “nachrop” (користувач має доступ лише до одного філіалу та може редагувати їх пристрої);
- “router” (користувач із даною роллю на має доступу до UI складової додатку системи, а може лише виконувати запит для отримання усіх доступних пристроїв у департаменті та виконувати запит на додавання нового пристрою).

Наведено фрагменти програмного коду з визначенням моделей на серверній частині буде у «ДОДАТКУ А». Хотілося б зазначити, що при описі моделі користувача виконується анонімна ф-ція, яка реалізовує функціонал створення первинного адміна див. ДОДАТОК А.1.

3.2 Опис програмної реалізації серверної частини

Написання серверної частини реалізована на Node.js з використанням backend фреймворку Express.js. Далі детальніше про flow-користувача у системі.

Зважаючи на вимоги до розроблюваної системи – користувач системи не може самостійно долучитися. Це робиться з метою забезпечення конфіденційності інформації. Тому створити користувача, який матиме доступ до певного департаменту та усіх його пристроїв, може лише інший користувач з відповідними адміністраторськими правами.

Тому flow-створення нового користувача відповідатиме маршруту з фрагменту коду нижче. Спочатку вхідні дані валідуватимуться, далі виконається перевірка чи має створюваний користувач мати право доступу за кількістю дозволених філіалів та його роллю. Далі відбудеться пошук користувача , що виконує запит, за його токеном і перевірить чи може виконатися запит відповідно до ролі запитувавшого юзера. Більш детально щодо маршрутів для роботи з користувачами у «ДОДАТКУ Б»

```
router.post(
        '/register',
        [
          requestDataValidator(userCreateValidator),
          userMiddleware.userAccessListValidator,
          authMiddleware.findUserByToken,
          authMiddleware.hasUserRoleAccess() // без аргументів тільки адмін
        ], authController.register
      );
```

Варто зазначити, що для виконання запитів від імені користувача з роллю «router», логіку було винесено в окремий розділ, оскільки запити на сервер виконуватимуться за «Базовою стратегією»(англ. “Basic strategy”). Для приведеного нижче коду, аутентифікація відбудеться за допомогою basicAuth middleware [10].

```
// POST basicStrategy
router.post(
  '/router',
  [
    requestDataValidator(createDeviceValidator),
    authMiddleware.basicAuth,
    authMiddleware.hasUserRoleAccess(userRoleEnum.ROUTER)
  ],
  deviceController.post
);


// GET all devices by DepartmentId
// basicStrategy
router.get(
  '/router/:departmentId',
  [
    authMiddleware.basicAuth,
    authMiddleware.hasUserRoleAccess(userRoleEnum.ROUTER)
  ],
  deviceController.getDevicesAsRouter
);
```

Щодо функціонала, який має реалізувати інформаційна система, то задля забезпечення конфіденційності даних було розроблено ряд перевірок, що у повній мірі дають можливість ідентифікувати користувача і виявити невалідні дані для запиту. Реалізація функціоналу перевірок базується на функціях посередників (middleware) та значною кількістю валідаторів. Серед них перевірки вхідних даних від клієнтів і даних, що записуватимуться у базу даних.

З метою розробки читабельного і легко розширюваного програмного коду, логіку middleware-ів та валідаторів було винесено у окремі складові та файли. А представлення головних контролерів — спрощено шляхом створення спеціалізованих сервісів, наприклад по роботі з паролями та jwt-токенами.

Список доступних користувачам маршрутів для роботи з департаментами та пристроями буде приведений у «ДОДАТКУ Б».

3.3 Опис програмної реалізації клієнтських частин

Веб-сайт

Внутрішня структура веб-додатку представляє собою ряд логічних переходів між сторінками. Також дає можливість зручно та доступно розмістити всю необхідну інформацію в розділах, підрозділах та категоріях, тому внутрішня структура система має бути повністю закрита для неавторизованих користувачів рис. 3.2.

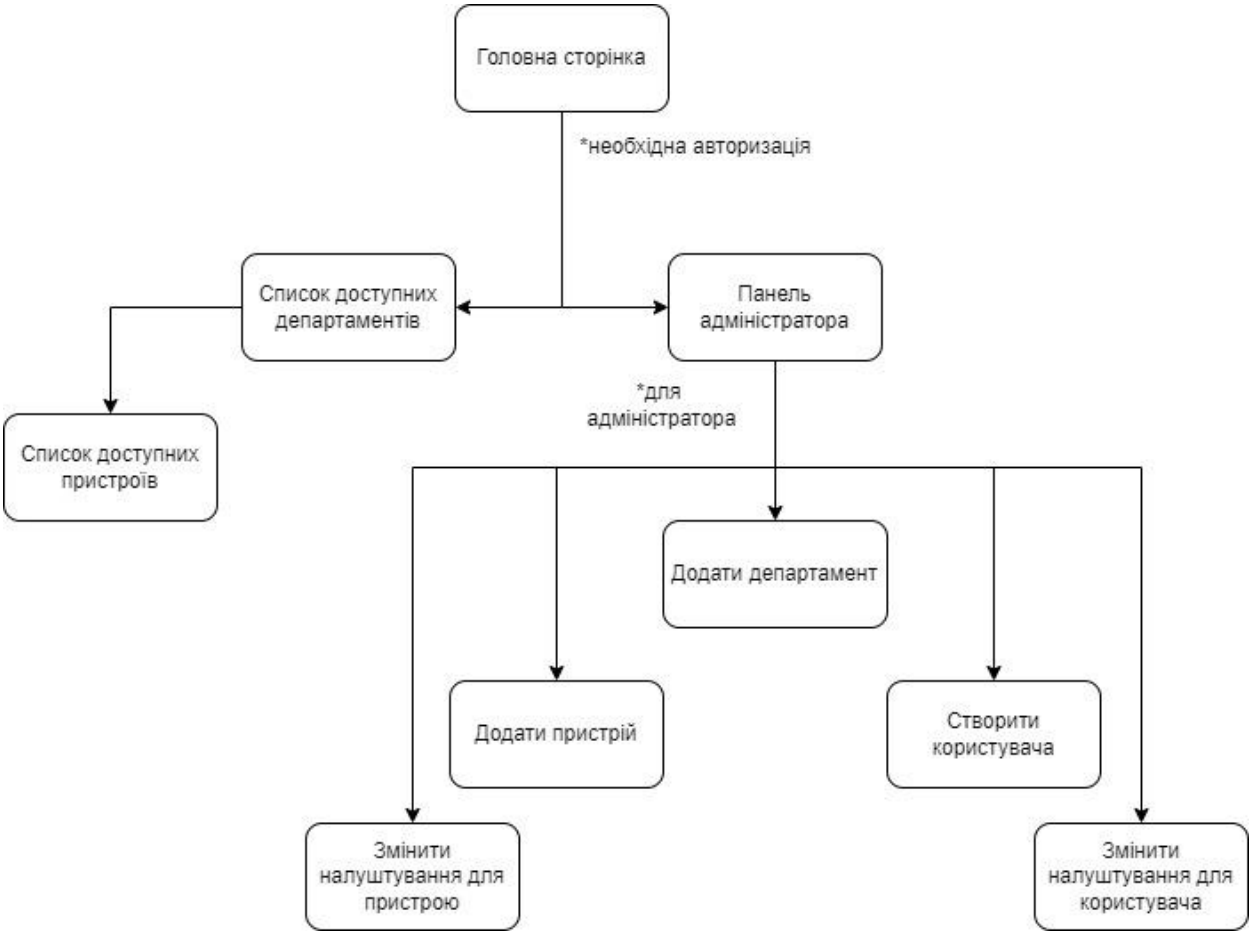

Рисунок 3.2 – Внутрішня структура веб-сайту

Маршрутизація на веб-сайті виконується як на фрагменті коду нижче.

```
const router = new VueRouter({
  mode: 'history',
  routes: [
    {
      path: '/',
      name: 'main',
      component: MainPage
    },
    {
      path: '/login',
      name: 'login',
      component: LoginPage
    },
    {
      path: '/admin',
      name: 'adminPage',
      component: AdminPage
    },
    {
      path: '/createUser',
      name: 'createUser',
      component: RegisterPage
    },
    {
```

```
      path: '/editUser',
      name: 'editUser',
      component: EditUserPage
    },
    {
      path: '/editDevice',
      name: 'editDevice',
      component: EditDevicePage
    },
    {
      path: '/createDevice',
      name: 'createDevice',
      component: CreateDevice
    },
    {
      path: '/createDepartment',
      name: 'createDepartment',
      component: CreateDepartment
    },
    {
      path: '/departments',
      name: 'departmentsLayout',
      component: DepartmentsLayout,
      children: [
        {
          path: '',
          name: 'departments',
          component: AllDepartmentsPage
        },
        {
          path: ':id',
          name: 'departmentPage',
          component: DepartmentPage,
        },
        {
          path: '*/*',
          redirect: { name: 'login' }
        },
    ]
    },
    {
      path: '*',
      name: 'notFound',
      component: NotFound
    },
  ]
})
```

У якості інструмента для розробки було використано Vue.js підхід у розробленні додатку може бути компонентним. Програмну реалізацію компонентів, що використовується на відповідних сторінках приведена у «ДОДАТКУ В».

Сторінки у проєкті є абсолютно незалежними, на відміну від компонентів. Так структура сторінки «DepartmentPage» повністю залежить від внутрішніх компонентів. Структура сторінки на рис. 3.3.

Також у шаблоні кожної сторінки є компонент «Navbar».

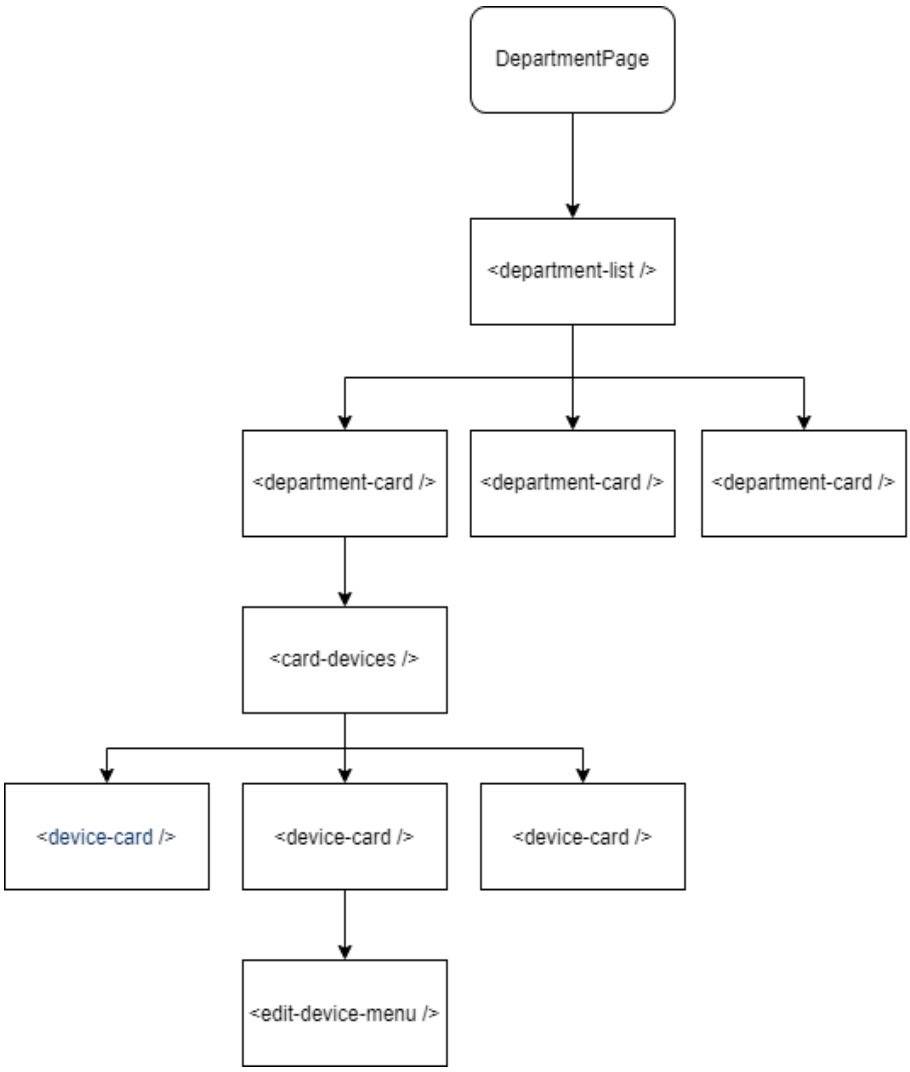

Рисунок 3.3 – Структура сторінки DepartmentPage і залежних компонентів

Bash service

Головною ідеєю системи є робота механізму додання правил на віддалений маршрутизатор за REST, а не базуючись на внутрішній базі відомих пристроїв.

З кожним отриманням пристроєм IP-адреси за dhcp – системи виконуватиме post-скрипт див. ДОДАТОК Г.1. За сценарієм має бути надісланий на ip-адресу серверної частини додатку POST-запит з тілом запита:

```
{
  department: “унікальний ідентифікатор, що відповідає ідентифікатору департа-
мента з бази даних”,
  mac: “мак-адреса «невідомого» пристрою”,
  owner: “керівник віддаленого департамента”,
  type: “не визначено”,
  allowed: false,
  comment: “системна інформація щодо пристрою, яку маршрутизатору надає сам
пристрій”
}
```

Також функціонал OpenWRT дозволяє працювати з iptables не додаючи правила безпосередньо. Це можливо додавши ряд правил у файл файлової системи маршрутизатора /etc/firewall.user.

Разом із запитом скрипт додасть iptables правила, до вищезгаданого файлу, з метою тимчасової заборони доступу до локальної мережі та мережі інтернет.

Сервіс, див. Додаток Г.3, управляє роботою get-скрипта див. Додаток Г.2, та надає зручний інструментарій по роботі з ним користувачу. Get-скрипт з певним інтервалом виконує get-запит на сервер, отримуючи відповідь у форматі json. Для зручного парсингу json було попередньо встановлено та підключено JSON processor –“jq”, за допомогою влаштованого пакетного менеджера “opkg”. На основі отриманих даних маршрутизатор може оновити список «відомих» системі MAC-адресу та оновити правила для заборони.

ВИСНОВКИ

Метою роботи було створити централізовану систему для управління доступами на віддалених філіалах. Для отримання якісного результату було виконано ряд наступних задач:

- розглянуто низку готових рішень, що могли б бути альтернативою створеній системі, і доведено їх недоречність;
- було обрано ряд сучасних технологій для розробки;
- розроблено серверну і клієнтські частини системи, дотримуючись усіх бізнес-вимог;
- розроблено механізм блокування нових пристроїв на кінцевому мережевому обладнанні;
- спроєктовано та реалізовано структуру взаємодії механізму блокування зі стороннім серверним додатком;

За результатами отримано гнучку систему, що дає змогу: до управління усіма пристроями у віддаленій мережі, що отримують IP-адресу за DHCP; збирати та зберігати дані щодо пристроїв користувачів у одному місці, а також мати можливість швидко керувати їх доступами до локальної мережі й мережі інтернет через зручний UI; залишати коментар, щодо певного пристрою і точно знати його володаря; делегувати відповідальність за надання/заборону доступів до мережі особам без вищої технічної освіти.

СПИСОК ЛІТЕРАТУРИ

1. Оліфер В.Г., Оліфер Н.О. Комп'ютерні мережі. Принципи, технології, протоколи. - Пітер: Ріміцан Н. А., 2020 - 131с.

2. Веб-служба [Електронний ресурс]. — Режим доступу: https://ru.Wikipedia.org/wiki/Веб-служба (дата звернення: 20.10.2021). — Назва з екрана.

3. Мікрослужби .NET: Архітектура контейнерних програм .NET [Електронний ресурс]. — Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices (дата звернення: 20.10.2021). — Назва з екрана.

4. Sam Newman. Monolith to Microservices Evolutionary Patterns to Transform Your Monolith. - Gravenstein Highway North, Sebastopol: O'Reilly Media, Incorporated, 2019 — 12с.

5. Надсилання даних форми [Електронний ресурс]. — Режим доступу: https://developer.mozilla.org/ru/docs/Learn/Forms/Sending_and_retrieving_form_data (дата звернення: 20.10.2021). — Назва з екрана.

6. Fernando Doglio. REST API Development with Node.js. - La Paz, Canelones, Urugguay: Apress, 2018 — 39с.

7. Eve Porcello, Alex Banks. Learning GraphQL Declarative Data Fetching for Modern Web Apps. - Gravenstein Highway North, Sebastopol: O'Reilly Media, Incorporated, 2018 - 10с.

8. Shelley Powers. Learning Node. - Gravenstein Highway North, Sebastopol: O'Reilly Media, Incorporated, 2017 – 58с.

9. Alex R. Young, Bradley Meck, Mike Cantelon, Tim Oxley, Marc Harter, TJ Holowaychuk, Nathan Rajlich. Node.js in Action. - NY: Manning, 2017 - 25с.

10. Веб-сайт з офіційною документацією Node.js [Електронний ресурс]. — Режим доступу: https://nodejs.org/en/about/ (дата звернення: 24.11.2020).

11. Michael Hartl. Ruby on Rails Tutorial. - Boston: Addison Wesley Professional, 2020 – 871с.

12. Дронов В.А. Django 3.0. Практика створення веб-сайтів на Python. - Пітер: БХВ-Петербург, 2020 — 120с.

13. Веб-сайт з офіційною документацією OpenWRT [Електронний ресурс]. — Режим доступу: https://openwrt.org/ (дата звернення: 24.11.2020).

14. Sam Newman. Monolith to Microservices Evolutionary Patterns to Transform Your Monolith. - Gravenstein Highway North, Sebastopol: O'Reilly Media, Incorporated, 2019 — 55c.

15. Eve Porcello, Alex Banks. Learning GraphQL Declarative Data Fetching for Modern Web Apps - Gravenstein Highway North, Sebastopol: O'Reilly Media, Incorporated, 2018 - 233c.

ДОДАТКИ

ДОДАТОК А

```
const ErrorHandler = require('../errors/ErrorHandler');

const { passwordService } = require('../services');

const {
  dbTablesEnum, defaultAdminCredentials, httpStatusCodes, userRoleEnum
} = require('../config');

const schema = new Schema({
  email: {
    type: String,
    required: true,
    unique: true
  },

  password: {
    type: String,
    required: true,
    select: false
  },

  role: {
    type: String,
    enum: Object.values(userRoleEnum),
    required: true
  },

  access: [{
    type: Schema.Types.ObjectId,
    ref: dbTablesEnum.DEPARTMENTS
  }],

  isActive: {
    type: Boolean,
    default: true,
    required: true
  }
});

const userModel = model(dbTablesEnum.USERS, schema);
const default_admin = {
  email: defaultAdminCredentials.EMAIL,
  role: userRoleEnum.ADMIN,
  access: [],
  password: defaultAdminCredentials.PASSWORD
};

(async () => {
  try {
    const admin = await userModel.findOne({ email: defaultAdminCreden-
tials.EMAIL });
    if (admin) {
      return;
    }

    const hashedPass = await passwordService.hashPass(default_admin.password);
    await userModel.create({ ...default_admin, password: hashedPass });
  } catch (e) {
    throw new ErrorHandler(
```

```
      httpStatusCodes.Internal_Server_Error,
      'Во время создания профиля default админа произошла ошибка'
    );
  }
})();

module.exports = userModel;
```

Фрагмент А.1 - Створення моделі «User»

```
const { Schema, model } = require('mongoose');

const { dbTablesEnum } = require('../config');

const OAuthSchema = new Schema({
    token: {
        type: String,
        required: true
    },
    user: {
        type: Schema.Types.ObjectId,
        required: true,
        ref: dbTablesEnum.USERS
    }
}, { timestamps: true });

module.exports = model(dbTablesEnum.OAUTH, OAuthSchema);
```

Фрагмент А.2 - Створення моделі «OAuth»

```
const { Schema, model } = require('mongoose');

const { dbTablesEnum } = require('../config');

const schema = new Schema({
  name: {
    type: String,
    unique: true
  },

  devices: [{
    type: Schema.Types.ObjectId,
    ref: dbTablesEnum.DEVICES
  }]
});

module.exports = model(dbTablesEnum.DEPARTMENTS, schema);
```

Фрагмент А.3 - Створення моделі «OAuth»

```
const { Schema, model } = require('mongoose');

const { dbTablesEnum } = require('../config');

const schema = new Schema({
  department: {
    type: Schema.Types.ObjectId,
    ref: dbTablesEnum.DEPARTMENTS,
    required: true
  },
```

```
  type: {
    type: String,
    required: true
  },
  owner: {
    type: String,
    required: true
  },
  mac: {
    type: String,
    lowercase: true,
    trim: true,
    required: true
  },
  allowed: {
    type: Boolean,
    default: false
  },
  comment: {
    type: String
  },
  addingDate: {
    type: Date,
    default: new Date()
  }
});

module.exports = model(dbTablesEnum.DEVICES, schema);
```

Фрагмент А.4 - Створення моделі «Device»

ДОДАТОК Б

```
router.get(
  '/',
  [
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess([
      userRoleEnum.ADMIN,
      userRoleEnum.NACH_ROP,
      userRoleEnum.REGIONAL
    ])
  ],
  departmentController.getAllDepartments
);

router.get(
  '/:departmentId',
  [
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess([
      userRoleEnum.ADMIN,
      userRoleEnum.NACH_ROP,
      userRoleEnum.REGIONAL
    ])
  ],
  departmentController.getDepartmentById
);

router.post(
  '/',
  [
    requestDataValidator(departmentCreateValidator),
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess(),
    departmentMiddleware.userIdListValidator
  ],
  departmentController.createDepartment
);

router.delete(
  '/:departmentId',
  [
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess()
  ],
  departmentController.removeDepartmentById
);
```

Фрагмент Б.1 – маршрути для роботи департаментами

```
router.get(
  '/',
  [
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess(userRoleEnum.ADMIN,
userRoleEnum.REGIONAL, userRoleEnum.NACH_ROP)
  ],
  deviceController.getAll
);

router.get(
  '/:deviceId',
```

```
  [
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess(userRoleEnum.ADMIN,
userRoleEnum.REGIONAL, userRoleEnum.NACH_ROP)
  ],
  deviceController.getDeviceById
);
router.post(
  '/',
  [
    requestDataValidator(createDeviceValidator),
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess(userRoleEnum.ADMIN,
userRoleEnum.REGIONAL, userRoleEnum.NACH_ROP),
    deviceMiddleware.hasUserDepartmentAccess(),
    deviceMiddleware.isMacUnique
  ],
  deviceController.post
);

// change only allowed param to each device
router.patch(
  '/multiple',
  [
    requestDataValidator(patchMultipleDeviceValidator),
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess(userRoleEnum.ADMIN,
userRoleEnum.REGIONAL, userRoleEnum.NACH_ROP),
    deviceMiddleware.hasUserDepartmentAccess('multiple'),
  ],
  deviceController.patchDevicesAllowedParam
);

router.patch(
  '/:deviceId',
  [
    requestDataValidator(patchSingleDeviceValidator),
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess(userRoleEnum.ADMIN,
userRoleEnum.REGIONAL, userRoleEnum.NACH_ROP),
    deviceMiddleware.hasUserDepartmentAccess(),
  ],
  deviceController.patchSingleDevice
);

router.delete(
  '/:deviceId',
  [
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess(userRoleEnum.ADMIN,
userRoleEnum.REGIONAL, userRoleEnum.NACH_ROP),
  ],
  deviceController.delete
);
```

Фрагмент Б.2 – маршрути для роботи з пристроями

```
router.get(
  '/',
  [
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess()
  ],
```

```
  userController.getAll
);

router.patch(
  '/:userId',
  [
    requestDataValidator(userPatchValidator),
    userMiddleware.userAccessListValidator,
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess()
  ],
  userController.patch
);

router.delete(
  '/:userId',
  [
    authMiddleware.findUserByToken,
    authMiddleware.hasUserRoleAccess()
  ],
  userController.deleteUserById
);
```

Фрагмент Б.3 – маршрути для роботи з користувачами

```
findUserByToken: async (req, res, next) => {
    try {
      const token = req.get('Authorization');
      if (!token) {
        next(new ErrorHandler(httpStatusCodes.Bad_Request, 'Запрос без исполь-
зования токена авторизации. Запрещено'));
      }

      await jwtService.verifyUserToken(token);

      const dbToken = await OAuth.findOne({ token });
      if (!dbToken) {
        next(new ErrorHandler(httpStatusCodes.Unauthorized, 'Bad token'));
      }

      const user = await User.findById(dbToken.user);
      if (!user) {
        next(new ErrorHandler(httpStatusCodes.Unauthorized, 'Bad token'));
      }

      if (!user.isActive) {
        next(new ErrorHandler(httpStatusCodes.Unauthorized, 'Пользователь не
активен. Обратитесь к администратору'));
      }

      req.locals = {
        ...req.locals,
        token: dbToken,
        user
      };

      next();
    } catch (e) {
      next(e);
    }
```

```
  },

  hasUserRoleAccess: (allowedAccessForRoles = [userRoleEnum.ADMIN]) => (req,
res, next) => {
    try {
      const { user } = req.locals;

      if (!allowedAccessForRoles.includes(user.role)) {
        next(new ErrorHandler(httpStatusCodes.Unauthorized, 'Недостаточно пол-
номочий'));
      }

      next();
    } catch (e) {
      next(e);
    }
  },

  basicAuth: async (req, res, next) => {
    try {
      if (req.headers.authorization.indexOf('Basic ') === -1) {
        next(new ErrorHandler(httpStatusCodes.Unauthorized, 'Bad token'));
      }

      const base64Credentials = req.get('Authorization').split(' ')[1];
      const credentials = Buffer.from(base64Credentials,
'base64').toString('ascii');
      const [
        email,
        password
      ] = credentials.split(':');

      const user = await User.findOne({ email }).select('+password');
      if (!user) {
        next(new ErrorHandler(httpStatusCodes.Unauthorized, 'Bad token'));
      }

      if (!user.isActive) {
        next(new ErrorHandler(httpStatusCodes.Unauthorized, 'Пользователь не
активен. Обратитесь к администратору'));
      }

      await passwordService.isPassMatches(password, user.password);

      req.locals = {
        ...req.locals,
        user
      };

      next();
    } catch (e) {
      next(e);
    }
  }
```

Фрагмент Б.4 – проміжні ф-ції перевірок(middleware) пов’язані з процесом аутентифікації

```
userIdListValidator: async (req, res, next) => {
    try {
      const { userIdList } = req.body;
```

```
      if (userIdList.length === 0) {
        next();
      }

      const users = await User.find({ _id: { $in: userIdList } });

      for (const user of users) {
        if ((user.role === userRoleEnum.ROUTER || user.role ===
userRoleEnum.NACH_ROP) && user.access.length > 0) {
          next(new ErrorHandler(
            httpStatusCodes.Bad_Request,
            `Пользователю ${user.email} невозможно предоставить доступ. Филиал
не добавлен`
          ));
        }
      }

      next();
    } catch (e) {
      next(e);
    }
  }
```

Фрагмент Б.5 – middleware пов’язані з валідацією даних при створення департаменту

```
hasUserDepartmentAccess: (deviceCount = 'single') => (req, res, next) => {
    const { user } = req.locals;

    if (deviceCount === 'single') {
      const { department } = req.body;

      if (!department || !user.access.includes(department)) {
        throw new ErrorHandler(httpStatusCodes.Bad_Request, 'У пользователя
недостаточно прав');
      }
    } else {
      const devices = req.body;

      devices.forEach((device) => {
        if (!user.access.includes(device.department)) {
          throw new ErrorHandler(httpStatusCodes.Bad_Request, 'У пользователя
недостаточно прав');
        }
      });
    }

    next();
  },

  isMacUnique: async (req, res, next) => {
    try {
      const { mac, department } = req.body;
      const lowercaseMac = mac.toLowerCase();

      const dbDepartment = await Department.findById(department);

      if (!dbDepartment) {
        next(new ErrorHandler(httpStatusCodes.Bad_Request, 'Bad input'));
      }
```

```
      const dbDevices = await Device.find({ _id: { $in: dbDepartment.de-
vices } });
      const result = dbDevices.find((device) => device.mac === lowercaseMac);

      if (result) {
        next(new ErrorHandler(httpStatusCodes.Conflict, 'Устройство с таким же
с MAC-адресом уже существует'));
      }

      next();
    } catch (e) {
      next(e);
    }
  }
```

Фрагмент Б.6 – middleware пов’язані з валідацією даних при роботі з колекцією пристроїв

```
hasUserDepartmentAccess: (deviceCount = 'single') => (req, res, next) => {
    const { user } = req.locals;

    if (deviceCount === 'single') {
      const { department } = req.body;

      if (!department || !user.access.includes(department)) {
        throw new ErrorHandler(httpStatusCodes.Bad_Request, 'У пользователя
недостаточно прав');
      }
    } else {
      const devices = req.body;

      devices.forEach((device) => {
        if (!user.access.includes(device.department)) {
          throw new ErrorHandler(httpStatusCodes.Bad_Request, 'У пользователя
недостаточно прав');
        }
      });
    }

    next();
  },

  isMacUnique: async (req, res, next) => {
    try {
      const { mac, department } = req.body;
      const lowercaseMac = mac.toLowerCase();

      const dbDepartment = await Department.findById(department);

      if (!dbDepartment) {
        next(new ErrorHandler(httpStatusCodes.Bad_Request, 'Bad input'));
      }

      const dbDevices = await Device.find({ _id: { $in: dbDepartment.de-
vices } });
      const result = dbDevices.find((device) => device.mac === lowercaseMac);

      if (result) {
        next(new ErrorHandler(httpStatusCodes.Conflict, 'Устройство с таким же
с MAC-адресом уже существует'));
```

```
      }

      next();
    } catch (e) {
      next(e);
    }
  }
```

Фрагмент Б.7 – middleware пов’язані з валідацією даних під час запиту користувача користувача

```
const Joi = require('joi');

const departmentCreateValidator = Joi.object({
  name: Joi.string()
    .trim()
    .required(),

  devices: Joi.array(),

  userIdList: Joi.array()
});

const departmentPatchValidator = Joi.object({
  devices: Joi.array()
});

const createDeviceValidator = Joi.object({
  department: Joi.string()
    .trim()
    .lowercase()
    .regex(regexp.MONGO_OBJECTID)
    .required(),

  type: Joi.string()
    .trim()
    .allow(...Object.values(deviceTypesEnum))
    .required(),

  owner: Joi.string()
    .trim()
    .required(),

  mac: Joi.string()
    .lowercase()
    .trim()
    .regex(regexp.MAC_ADR_REGEXP)
    .required(),

  allowed: Joi.boolean(),

  comment: Joi.string()
    .trim()
    .allow('')
});

const patchSingleDeviceValidator = Joi.object({
  department: Joi.string()
    .trim()
```

```
    .lowercase()
    .regex(regexp.MONGO_OBJECTID)
    .required(),

  type: Joi.string()
    .trim()
    .allow(...Object.values(deviceTypesEnum)),

  owner: Joi.string()
    .trim(),

  allowed: Joi.boolean(),

  comment: Joi.string()
    .trim()
    .allow('')
});

const _singleDevice = Joi.object({
  _id: Joi.string()
    .trim()
    .required(),

  department: Joi.string()
    .trim()
    .lowercase()
    .regex(regexp.MONGO_OBJECTID)
    .required(),

  allowed: Joi.boolean()
    .required()
});

const patchMultipleDeviceValidator = Joi.array().items(_singleDevice);

const userCreateValidator = Joi.object({
  email: Joi.string()
    .email({
      tlds: {
        allow: [
          'com',
          'net',
          'ua'
        ]
      }
    })
    .required()
    .lowercase()
    .trim(),

  role: Joi.string()
    .allow(...Object.values(userRoleEnum))
    .required(),

  access: Joi.array()
    .items(
      Joi.string()
        .regex(regexp.MONGO_OBJECTID)
    ),

  password: Joi.string()
    .trim()
    .required()
    .regex(regexp.PASS_REGEXP)
```

```
});

const userPatchValidator = Joi.object({
  role: Joi.string()
    .allow(...Object.values(userRoleEnum)),

  access: Joi.array()
    .items(
      Joi.string()
        .regex(regexp.MONGO_OBJECTID)
    ),

  password: Joi.string()
    .trim()
    .regex(regexp.PASS_REGEXP),

  isActive: Joi.boolean()
});
```

Фрагмент Б.8 – валідатори вхідних даних відповідно до моделей баз даних «User» «Device» «Department»

ДОДАТОК В

```
<template>
  <div class="department-card" style="min-width: 425px;">
    <div class="card text-center">
      <img class="card-img-top" src="../assets/images/logo.png" alt="Card image
cap">
      <div class="card-body">
        <h5 class="card-title">{{ department.name }}</h5>
        <p       class="card-devices">Устройств:       {{       (department.de-
vices).length }}</p>
        <router-link :to="$router.resolve({name: 'departmentPage', params: {id:
department._id}}).href"
          class="more-info"
        >К устройствам
        </router-link>
      </div>
    </div>
  </div>
</template>
```

Фрагмент В.1 – Компонент картки департаменту у списку департаментів

```
<department-card />
```

```
<template>
  <div class="department-list">
    <div class="row justify-content-center">
      <department-card
        class="department-card"
        v-for="department in departments"
        :key="department._id"
        :department="department">

      </department-card>
    </div>

  </div>

</template>
```

Фрагмент В.2 – Компонент списку департаментів на сторінці DepartmentPage

```
<card-devices />
```

```
<template>
  <div class="device-card">
    <div class="card box-shadow mx-auto my-5" style="width: 18rem">
      <div
        class="card-header"

        :style="{ background: device.allowed ? '#3c6' : '#CD5C5C' }"
      >
        <div class="logo-container">
          <img :src="getLogoPath" class="card-img-top" alt="logo" />
        </div>
        <!-- <router-link :to="$router.resolve({ name: 'main' }).href">
          <div class="wrapper pencil">
            <img src="../assets/images/pencil-removebg-preview.png" />
          </div>
        </router-link> -->
        <a type="button" class="logo-container" @click="showModal">
```

```
          <div class="wrapper pencil">
            <img src="../assets/images/pencil-removebg-preview.png" />
          </div>
        </a>
      </div>
      <div class="card-body">
        <div class="card-title">
          <h5>{{ device.type }}</h5>
        </div>
        <div class="content">
          <p class="card-text"><b>Владелец:</b> {{ device.owner }}</p>
          <p class="card-text"><b>Мак адресс:</b> {{ device.mac }}</p>
          <p class="card-text">
            <b>Дата добавления:</b> {{ new Date(device.addingDate) }}
          </p>
          <div v-if="device.comment">
            <hr />
            <p class="text-secondary">{{ device.comment }}</p>
          </div>
        </div>
      </div>
      <div class="card-footer">
        <button
          class="btn btn-success mr-3"
          v-on:click="allowDevice(device._id)"
          :disabled="device.allowed"
        >
          Разрешить
        </button>
        <button
          class="btn btn-danger"
          v-on:click="disallowDevice(device._id)"
          :disabled="!device.allowed"
        >
          Запретить
        </button>
      </div>
    </div>
  </div>
</template>
```

Фрагмент В.3 – Компонент картки пристрою

```
<device-card />
```

```
<template>
  <div class="device-card">
    <div class="card box-shadow mx-auto my-5" style="width: 18rem">
      <div
        class="card-header"

        :style="{ background: device.allowed ? '#3c6' : '#CD5C5C' }"
      >
        <div class="logo-container">
          <img :src="getLogoPath" class="card-img-top" alt="logo" />
        </div>
        <a type="button" class="logo-container" @click="showModal">
          <div class="wrapper pencil">
            <img src="../assets/images/pencil-removebg-preview.png" />
          </div>
        </a>
      </div>
      <div class="card-body">
```

```
        <div class="card-title">
          <h5>{{ device.type }}</h5>
        </div>
        <div class="content">
          <p class="card-text"><b>Владелец:</b> {{ device.owner }}</p>
          <p class="card-text"><b>Мак адресс:</b> {{ device.mac }}</p>
          <p class="card-text">
            <b>Дата добавления:</b> {{ new Date(device.addingDate) }}
          </p>
          <div v-if="device.comment">
            <hr />
            <p class="text-secondary">{{ device.comment }}</p>
          </div>
        </div>
      </div>
      <div class="card-footer">
        <button
          class="btn btn-success mr-3"
          v-on:click="allowDevice(device._id)"
          :disabled="device.allowed"
        >
          Разрешить
        </button>
        <button
          class="btn btn-danger"
          v-on:click="disallowDevice(device._id)"
          :disabled="!device.allowed"
        >
          Запретить
        </button>
      </div>
    </div>
  </div>
</template>
```

Фрагмент В.4 – Компонент картки пристрою

```
<device-card />
```

```
<div v-if="devices.length">
        <div class="row justify-content-start">
          <div v-for="device in devices" :key="device">
            <div class="col-sm-4 ml-5 mr-4">
              <device-card :device="device" @showModal="showModal" />
              <modal :name="device._id" :width="450" :height="'auto'">
                <edit-device-menu :device="device" />
              </modal>
            </div>
          </div>
        </div>
        <button
          class="btn btn-primary mb-5"
          type="submit"
          v-on:click="applyChanges()"
        >
          Сохранить изменения
        </button>
      </div>
      <div v-else class="text-center">
        <div class="spinner-border" role="status">
          <span class="sr-only">Loading...</span>
        </div>
      </div>
```

Фрагмент В.5 – Компонент списку карток пристроїв з можливістю відкрити модальне вікно для редагування на картці пристрою

```
<device-list />
```

```
<div v-if="devices.length">
        <div class="row justify-content-start">
          <div v-for="device in devices" :key="device">
            <div class="col-sm-4 ml-5 mr-4">
              <device-card :device="device" @showModal="showModal" />
              <modal :name="device._id" :width="450" :height="'auto'">
                <edit-device-menu :device="device" />
              </modal>
            </div>
          </div>
        </div>
        <button
          class="btn btn-primary mb-5"
          type="submit"
          v-on:click="applyChanges()"
        >
          Сохранить изменения
        </button>
      </div>
      <div v-else class="text-center">
        <div class="spinner-border" role="status">
          <span class="sr-only">Loading...</span>
        </div>
      </div>
```

Фрагмент В.6 – Компонент модального вікна для редагування пристрою

```
<edit-device-menu />
```

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <router-link
    class="navbar-brand"
    :to="$router.resolve({name: 'main'}).href">Главная
  </router-link >
  <button
    class="navbar-toggler"
    type="button"
    data-toggle="collapse"
    data-target="#navbarSupportedContent"
    aria-controls="navbarSupportedContent"
    aria-expanded="false"
    aria-label="Toggle navigation"
  >
      <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto" v-if="auth">
      <li class="nav-item">
        <router-link
        class="nav-link"
        aria-current="page"
        :to="$router.resolve({name: 'departmentsLayout'}).href"
        >Мои филиалы
```

```
        </router-link>
      </li>
      <li class="nav-item" v-if="role ==='admin'">
        <router-link
        class="nav-link"
        aria-current="page"
        :to="$router.resolve({name: 'adminPage'}).href"
        >Админ панель
        </router-link>
      </li>
    </ul>


  </div>
  <div class="is-logged">
      <form class="form-inline my-2 my-lg-0" v-if="!auth">
        <router-link
          class="btn btn-outline-success my-2 my-sm-0"
          type="submit"
          :to="$router.resolve({name: 'login'}).href"
          >Войти
        </router-link >
      </form>
      <form class="form-inline my-2 my-lg-0" v-else>
        <div class="nav-item dropdown">
          <button
          class="btn btn-default dropdown-toggle"
          type="button"
          data-toggle="dropdown"
          >Привет, {{ email }}
          <span class="caret"></span>
          </button>
          <div     class="dropdown-menu     dropdown-menu-right"     aria-la-
belledby="navbarDropdown">
            <button
              class="btn nav-link"
              type="submit"
              @click="handleLogout"
              >Выйти
            </button >
          </div>
        </div>
      </form>
    </div>
</nav>
```

Фрагмент В.7 – Компонент navbar

ДОДАТОК Г

```
#!/bin/sh
# script to detect new dhcp lease

# this will be called by dnsmasq everytime a new device is connected
# with the following arguments
# $1 = add | old
# $2 = mac address
# $3 = ip address
# $4 = device name

############### SETUP STAGE ##############

known_mac_addr="/etc/macfilter_client/known_mac_addr"
IP='192.168.9.163'
PORT=3001

#Database department ID
DEPARTMENT_ID='60daba810749b827e0cb8448'
AUTH_TOKEN='Basic a2FyYWt1dHNAZmFyYmV4LmNvbS51YToxMjM='
NACH_ROP='Karakuts'

############### SETUP END ###############

# check if the mac is in known devices list
grep -iq "$2" "$known_mac_addr"
unknown_mac_addr=$?

echo "$1"

if [ "$1" == "add" ] && [ "$unknown_mac_addr" -ne 0 ]; then
  msg="New device on `uci get system.@system[0].hostname` $*"
  echo `date` $msg >> $known_mac_addr

  echo "adding"
  curl --location --request POST "http://$IP:$PORT/api/devices/router" \
  --header 'Content-Type: application/json' \
  --header "Authorization: Basic $AUTH_TOKEN" \
  --data-raw '{
    "department": "'"$DEPARTMENT_ID"'",
    "type": "Other",
    "owner": "'"$NACH_ROP"'",
    "mac": "'"$2"'",
    "allowed": false,
    "comment": "'"$4"'"
  }'>>log.txt

  # adding rules to iptables once
  grep -iq "$2" /etc/firewall.user
  is_rule_already_added=$?

  if [ "$is_rule_already_added" -ne 0 ]; then
    echo "iptables -A INPUT -m mac --mac-source $2 -j DROP" >> /etc/fire-
wall.user
    echo "iptables -A FORWARD -m mac --mac-source $2 -j DROP" >> /etc/fire-
wall.user
  fi

  /etc/init.d/firewall restart
fi
```

Фрагмент Г.1 – post-скрипт

```
#!/bin/sh
. /usr/share/libubox/jshn.sh

############### SETUP STAGE ##############

TIMEOUT=30
known_mac_addr="/etc/macfilter_client/known_mac_addr"
log="/etc/macfilter_client/log.txt"
IP='192.168.9.163'
PORT=3000

#Database department ID
DEPARTMENT_ID='60daba810749b827e0cb8448'
AUTH_TOKEN='Basic a2FyYWt1dHNAZmFyYmV4LmNvbS51YToxMjM='

############### SETUP END ###############

while true
do
      response=$(curl --location --request GET "http://$IP:$PORT/api/de-
vices/router/$DEPARTMENT_ID" \
      --header "Authorization: Basic $AUTH_TOKEN" | jq -r '.devices')

      if [ -f /etc/firewall.user ]; then
            rm /etc/firewall.user
      fi

      count=$(echo $response | grep -i -o "mac" | wc -l )

      new_known_mac_addr=""

      i=0; while [ $i -lt $count ]; do

                  test=$(echo $response | jq '.['$i'] | {mac, allowed}')

                  json_load "$test"

                  json_select

                  json_get_var mac mac

                  json_get_var allowed allowed

                  # check if the mac is in known devices list

                  if ! [ -f $known_mac_addr ]; then

                        touch $known_mac_addr

                        echo -e "Init data from web:\n $(echo $response | jq
'.[] | mac')" >> $known_mac_addr

                  fi

                  grep -iq "$mac" "$known_mac_addr"

                  unknown_mac_addr=$?

                  if [ "$unknown_mac_addr" -ne 0 ]; then

                        msg="Device was added from web. MAC: $mac"
```

```
                        echo `date` $msg >> $known_mac_addr

                fi

                new_known_mac_addr=$(echo -e "$new_known_mac_addr\n $(cat
"$known_mac_addr" | grep $mac)" )

                echo "`date` Device with mac: $mac - allowed: $allowed" >>
$log

                if [ $allowed -eq 0 ]; then

                                echo "iptables -A INPUT -m mac --mac-source $mac
-j DROP" >> /etc/firewall.user

                                echo "iptables -A FORWARD -m mac --mac-source
$mac -j DROP" >> /etc/firewall.user
                fi

                i=$(($i + 1))
      done

      echo -e "$new_known_mac_addr" > "$known_mac_addr"
      /etc/init.d/firewall restart
      sleep $TIMEOUT
      echo -e "\n" >> $log
done
```

Фрагмент Г.2 – get-скрипт

```
#!/bin/sh /etc/rc.common

START=10
STOP=15

name="macfilter_client"
pid_file="/var/run/$name.pid"
dir="/etc/macfilter_client"
app="sh get_script.sh &>/dev/null &"

stdout_log="/var/log/$name.log"
stderr_log="/var/log/$name.err"

is_working(){
        if [ -f $pid_file ]; then
                echo true
        else
                echo false
        fi
}

start(){
  if [ -d $dir ]; then
      if ! `is_working` ; then

            echo "name $name"
                cd $dir
                $app >> "$stdout_log" 2>> "$stderr_log" &
                echo `pgrep -f get_script.sh` > "$pid_file"
                echo `pgrep -f sleep` >> "$pid_file"
            echo "Started with pid $(pgrep -f get_script.sh)"
                echo "Working..."
```

```
        else
            echo "name $name"
                echo "App working now"

        fi
  fi
}
stop(){
  if `is_working` ; then

                kill $(cat $pid_file)

                rm $pid_file

                echo "Closed!"
        else
                echo "App is not working now!"

        fi

}
```

Фрагмент Г.3 – macfilter_client управляючий сервіс