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

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія забезпечення централізованого контролю за використанням USB-накопичувачів в корпоративній мережі»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Барченко Н.Л.**

Студента групи ІН.м-02 **Татарінов В.Р.**

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

Татарінову Вадиму Руслановичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія забезпечення централізованого контролю за використанням USB-накопичувачів в корпоративній мережі*

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) Огляд технологій, що дозволять реалізувати поставлену задачу; 2) Постановка завдання й формування завданння дослідження; 3) Огляд технологій, що використовуються під час розробки додатків на ОС Windows, ASP.NET Core, React; 4) Моделювання та проектування; 5) Програмна реалізація; 6) Аналіз результатів.

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
1.	Огляд наявних рішень		
2.	Постановка задачі та формування завдань дослідження.		
3.	Проектування та моделювання		
4.	Програмна реалізація		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 117 стор., 23 рис., таблиці, 1 додаток, 24 літературних джерел.

Об'єкт дослідження — технологія забезпечення централізованого контролю за використанням USB-накопичувачів в корпоративній мережі.

Мета роботи — розробка технології, яка забезпечить централізований контролю за використанням USB-накопичувачів в корпоративній мережі.

Результати — було проведено аналіз предметної області, який складався з огляду літератури та аналізу наявних рішень. Досліджено наявні способи забезпечення мережевої взаємодії, методи побудови веб-додатків та способи взаємодії з операційною системою Windows. Проведено аналіз засобів програмної реалізації, які дозволять розробити технологію централізованого контролю використання USB-накопичувачів в корпоративній мережі. Після аналізу предметної області та засобів реалізації було проведено проектування та моделювання технології під час, якої була розроблена діаграма варіантів використання, створений прототип веб-інтерфейсу та спроектована структура бази даних. На завершальному етапі було проведено програмну реалізацію. Результатом роботи є розроблена технологія, яка забезпечує централізований контроль за використанням USB-накопичувачів в корпоративній мережі.

ТЕХНОЛОГІЯ КОНТРОЛЮ ВИКОРИСТАННЯ USB ПРИСТРОЇВ, ASP.NET CORE, C#, JAVASCRIPT, REACT, ENTITY FRAMEWORK CORE, POSTGRESQL, WMI, TCP

ЗМІСТ

ВСТУП

На сьогоднішній день USB-накопичувачі набули широкої популярності завдяки невеликому розміру, значному обсягу пам’яті, доступній ціні та портативності, що дозволяє зручно зберігати та переносити файли з одного комп’ютера на інший. Однак ці характеристики роблять їх привабливими для зловмисників[1]. При нехтуванні заходами безпеки, що направлені на захист комп’ютерів від небажаних підключень USB-пристроїв, компанії ризикують зазнати суттєвих збитків. Зловмисники можуть використовувати USB-накопичувачі для зараження інших комп'ютерів шкідливими програмами або поцупити конфіденційну інформацію[2]. Останнім часом набули розповсюдження атаки через начебто загублені пристрої. Підібравши пристрій та під’єднавши його до комп’ютера, працівники компанії навіть не здогадуються, що стали співучасниками кібератаки[3]. Також нерідко працівники навмисно вдаються до злочинних дій заради помсти або наживи. Тому безпека USB-портів потребує значної уваги.

Найефективніший спосіб захиститися від підключень небажаних накопичувачів - повне відключення усіх USB-портів. Нажаль, дані міри безпеки зроблять взаємодію користувача з комп’ютером неможливою[4]. Звідси виникає необхідність у використанні програмного забезпечення, що не уможливіть підключення небажаних пристроїв та дозволить відслідкувати історію використання дозволених USB-накопичувачів. Використання подібного програмного забезпечення значно підсилить рівень безпеки в корпоративній мережі та не буде заважати нормальній роботі користувачів.

Метою даної роботи є розробка система централізованого контролю та управління використання USB-накопичувачів, що дозволить значно підвищити рівень безпеки в корпоративній мережі.

Новизна роботи полягає в вирішенні наступних недоліків, що має переважна частина наявних рішень:

- безкоштовні рішення не мають потрібного функціоналу, що дозволить централізовано управляти безпекою USB-портів;
- більшість рішень мають громіздкий та незрозумілий інтерфейс;
- деякі існуючих рішень працюють в хмарах без можливості розгорнутися на власних серверах;
- переважна частина платних рішень мають зайвий функціонал, що додає системі зайвої складності;
- більшість платних рішень мають занадто високу вартість для невеликих компаній.

На основі виявлених недоліків, які мають переважна частина наявних рішень, в даній роботі буде розроблено систему, що дозволить:

- задавати списки дозволених пристроїв на основі яких буде прийматися рішення;
- переглядати список підключених USB-накопичувачів, які використовуються в корпоративній мережі;
- можливість блокувати USB-накопичувачі, що використовуються на комп’ютері;
- відслідковувати історію підключень носіїв інформації;
- централізовано управляти безпекою, за допомогою інтуїтивно зрозумілого веб-застосунку;
- розгортати систему на власних серверах;

Для досягнення поставленої мети треба вирішити наступні задачі:

- провести аналіз предметної області;
- обрати засоби та методи для вирішення задачі;
- виконати проектування системи;
- здійснити програмну реалізацію.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Літературний огляд

1.1.1. Огляд технологій мережевої взаємодії

Протокол TCP – це стандарт зв'язку гарантованого доставлення повідомлень через мережу Інтернет. Він дозволяє надсилати повідомлення забезпечуючи успішну доставку. TCP є одним основних стандартів, що визначають правила Інтернету, і входить до стандартів, визначених Інженерною групою з розробки Інтернету (IETF)[5]. Цей протокол є основним, бо на його пазі працюють додатки прикладного рівня.

TCP організовує дані так, щоб їх можна було передавати між сервером і клієнтом, гарантуючи цілісність даних. Перш ніж передавати дані TCP ініціалізує з'єднання між джерелом та отримувачем, який мусить забезпечувати працездатність до початку передачі повідомлень. Потім він ділить великі повідомлення на більш менші пакети, гарантуючи при цьому непорушність даних протягом усього процесу[5]. В результаті всі протоколи, які належать до високого рівня та вимагають передачу даних, використовують протокол TCP. Основними прикладами можуть слугувати такі протоколи, як (FTP), Secure Shell (SSH) і Telnet[6]. Його також використовують, щоб відправляти та отримувати повідомлення через електронну пошту за допомогою таких протоколів як IMAP та (POP). Найбільш розповсюджений веб-протокол HTTTP також базується на даному протоколі.

Альтернативним варіантом до протоколу TCP є протокол User Datagram Protocol (UDP), який призначений для встановлення з’єднань з малою затримкою між програмами та скорочення часу передачі[5]. TCP може бути занадто повільним для побудови деяких додатків, оскільки він

намагається відновити відсутні або пошкоджені пакети та гарантує доставку даних за допомогою таких засобів керування, як підтвердження, запуск з’єднання та керування потоком. UDP не намагається відновити втрачені пакети або пошкоджені пакети та також не сигналізує про пункт призначення перед доставкою даних, що робить його менш надійним, але більш швидким[5]. Таким чином, це хороший варіант для чутливих до часу ситуацій, таких як пошук системи доменних імен (DNS), передача голосу через Інтернет-протокол (VoIP) і для потокової мультимедіа.

Операційні системи надають різні засоби для маніпулювання своїми TCP-з'єднаннями. Sockets API приховує всі деталі TCP та IP від програміста HTTP[6]. Він вперше був розроблений для операційної системи Unix, але тепер варіанти доступні майже для кожної операційної системи та мови. Таблиця 1.1 показує деякі з основних функцій для програмування сокетів.

Таблиця 1.1 – Загальні функції інтерфейсу TCP-сокетів.

Sockets API	Опис
s = socket(<parameters>)	Створює новий, неприєднаний сокет.
bind(s, <local IP:port>)	Призначає номер локального порту та інтерфейс сокету.
connect(s, <remote IP:port>)	Встановлює TCP-з'єднання з локальним сокетом і віддаленим хостом і портом.
listen(s,...)	Починає прослуховувати порт для під’єднання.
s2 = accept(s)	Очікує, поки хтось встановить з’єднання з локальним портом.
n = read(s,buffer,n)	Намагається прочитати n байтів із сокета в буфер.
n = write(s,buffer,n)	Намагається записати n байтів з буфера в сокет.

Sockets API	Опис
close(s)	Повністю закриває TCP-з'єднання.
shutdown(s,<side>)	Закриває лише вхід або вихід TCP-з’єднання.
getsockopt(s, . . .)	Зчитує значення параметра конфігурації внутрішнього сокета.
setsockopt(s, . . .)	Змінює значення параметра конфігурації внутрішнього сокета.

1.1.2. Методи побудови веб-додатків

Веб-додатки можна створювати різними способами – як односторінкові програми (SPA) і багатосторінкові програми (MPA).

Односторінковий додаток — це веб-додаток, який складається з однієї HTML-сторінки, завантажує нові дані JSON із сервера та переписує сторінку у відповідь на дії користувача[7]. На відміну від MPA, SPA не потрібно завантажувати кожну нову веб-сторінку з нуля — вони просто змінюють наявну сторінку[8]. Цей підхід дозволяє створювати односторінкові програми, які запускаються у браузері, але мають інтерфейс користувача та функціональність, подібну до програм для настільних комп’ютерів.

По суті, односторінкові програми засновані на Ajax — наборі методів веб-розробки для надсилання та обробки запитів із сервера у фоновому режимі без необхідності перезавантажувати цілу сторінку[9]. Ajax дозволив розробникам почати складати складні функції в програми, які поміщаються на одній сторінці.

Рисунок 1.1.1 - Принцип роботи багатосторінкових та односторінкових веб-додатків

SPA стали надзвичайно популярними завдяки активній підтримці з боку Google, яка прийняла цей формат для більшості своїх послуг. Сьогодні SPA в основному використовуються для:

- соціальні мережі (Facebook, Twitter, LinkedIn)
- інформаційні панелі (Jira, Trello)
- веб-аналоги локальних додатків (Google Docs, Slack, Telegram)
- програмне забезпечення як сервісний продукт динамічні проекти, які включають невелику кількість даних внутрішні корпоративні проекти.

Багатосторінковий додаток (або MPA) – це веб-додаток, який складається з кількох веб-сторінок, які завантажуються, коли користувач відвідує різні частини сторінки[10]. Це традиційний шаблон розробки веб-додатків, який підходить для веб-сайтів, які мають справу з великою кількістю вмісту. Кожен запит, який ми надсилаємо на сервер, як-от щоразу, коли ми вводимо нову URL-адресу або натискаємо посилання, призводить до повернення нової сторінки із сервера[11].

Чудовими прикладами багатосторінкових додатків є такі гіганти, як Amazon або eBay. Використовуючи їх, ви завжди отримуєте новий файл для кожного запиту. Як правило, багатосторінкові додатки є складними з

кількома рівнями, посиланнями та різними інтерфейсами. Вміст таких веб-сайтів розбивається на кілька мікро-сайтів, розділів та підрозділів.

1.1.3. Методи отримання подій та даних про стан системи в операційній системі Windows

WMI – це технологія управління, яка впроваджується в операційну систему Windows з часів Windows NT 4.0[12]. Кожна нова версія Windows, так само, як і кожна нова версія багатьох інших продуктів Microsoft, оновлює WMI, надаючи їй нові риси і наділяючи новими можливостями. WMI надає інформацію відповідно до моделі Common Information Model, яка була розроблена промисловою групою Distributed Management task Force за участю Microsoft та інших постачальників. Ця модель зберігає практично будь-яку інформацію в сховищі, яка пов’язана з внутрішньою роботою Windows і тим, що в ній виконується. WMI – це потужний інструмент, який адміністратори використовують для локального та віддаленого керування Windows. Використовуючи WMI, адміністратори можуть запитувати інформацію в системі Windows, як-от встановлені програми, статус служби, файли у файловій системі та багато іншого[13]. WMI дозволяє підприємствам моніторити та збирати інформацію про стан операційної системи та додатків без використання дорогих інструментів.

WMI - це не просто цілісна програма. Вона включає в себе кілька провайдерів, кожен з яких з'єднує WMI-сервіс з конкретним продуктом, технологією, функцією і т.д. WMI провайдер діє майже як драйвер пристрою, забезпечуючи, доступ до різних продуктів Windows[12]. Наприклад, на комп'ютерах з операційною системою Windows Server, на яких встановлений сервіс Windows DNS, WMI провайдер дозволяє WMI сервісу запитувати і створювати ресурсні записи DNS, а також налаштування конфігурації DNS.

Вся інформація, яку отримує WMI-провайдер, реєструється в WMI-сховище, централізованої конфігураційної бази даних, яка вказує WMI, що інформація доступна на тому чи іншому конкретному комп'ютері. Важливо розуміти, що інформація, яку Ви отримуєте за допомогою WMI, насправді не знаходиться в сховищі: сховище містить лише перелік доступної інформації, і WMI витягує її динамічно, за запитом[14]. Сховище складається з просторів імен, які приблизно відповідають окремим продуктам і технологіям. Простори імен розташовані ієрархічно, а значить, вони можуть включати в себе підпростори.

Простір імені верхнього рівня є кореневим. Інші простори можуть містити:

- Root\Cimv2
- Root\MicrosoftDNS
- Root\SecurityCenter
- Root\MicrosoftActiveDirectory

Всередині кожного простору імен WMI виділяється один або декілька класів. Клас - це абстрактне значення компонента управління. Наприклад, простір імен Root\Cimv2 містить клас, який називається Win32_TapeDrive. Цей клас визначає властивості стрічкового накопичувача і існує в просторі імені незалежно від того, підключений даний носій до комп'ютера чи ні.

Класи в просторі імен the Root\Cimv2 майже завжди мають префікс Win32_, навіть на 64-бітних комп'ютерах. Цей префікс не використовується в назвах класів інших просторів імен. В інших просторах імен назви класів взагалі рідко мають будь-якої префікс. До інших класів простору імен Root\Cimv2 відносяться:

- Win32_Keyboard
- Win32_LogicalDisk
- Win32_NetworkAdapterConfiguration

- Win32_NTDomain
- Win32_Account
- Win32_Product
- Win32_Service
- Win32_BIOS
- Win32_Desktop
- Win32_Fan
- Win32_Group

Як видно, простір імен Root\Cimv2 містить класи, що мають відношення до операційної системи Windows і апаратного забезпечення комп'ютера. Це одне з небагатьох просторів імен, що містяться на кожному комп'ютері з Windows, хоча класи всередині цього простору імен розрізняються на серверних і клієнтських комп'ютерах, а також на різних версіях Windows ОС. Класи визначають два важливі елементи: властивості та методи. Кожен клас має як мінімум одну властивість, і подібно до інших об'єктів в Windows PowerShell, властивості класу забезпечують доступ до управлінської та конфігураційної інформації. Деякі класи мають методи, які вимагають певних дій, наприклад, перезавантаження комп'ютера або зміни конфігурації.

Реальне існування класу називають екземпляром. Наприклад, якщо на комп'ютері є чотири логічних диска, то він має чотири екземпляри класу Win32_LogicalDisk. Якщо на комп'ютері немає підключеного стрічкового накопичувача, у нього буде нульовий екземпляр класу Win32_TapeDrive, хоча сам клас буде існувати у вигляді абстрактного значення.

Екземпляри є такими ж об'єктами, як і будь-які інші об'єкти, які використовуються в Windows PowerShell. Екземпляри мають властивості та методи, визначені їх класом, і ви можете працювати з цими властивостями та методами всередині конвеєра оболонки.

У багатьох випадках екземпляри призначені лише для читання, що означає, що є можливість отримати значення властивості, але не змінити його. Особливо це стосується класів в просторі імен Root\Cimv2. Для того, щоб змінити налаштування конфігурації, клас повинен володіти методом, який можна використовувати для здійснення змін. За межами простору імен Root\Cimv2 принцип роботи може бути іншим. У деяких випадках можна змінити значення властивості шляхом прикріплення до нього нової властивості. Прикладом цього може служити простір імен для Internet Information Services (IIS) 6 всередині WMI.

У Windows у будь-який момент часу можуть відбуватися сотні подій. Коли ви використовуєте різні функції Windows, як-от створення файлів, зупинка та запуск служб, встановлення програмного забезпечення або будь-що інше, ймовірно, ініціюється подія WMI[13]. Майже кожна дія, яка виконується в Windows, може бути відкрита за допомогою події WMI. Коли дію виконується в Windows, Windows запускає подію через свою внутрішню інфраструктуру. За замовчуванням ці події побачити не можна. Вони відбуваються у фоновому режимі. Щоб побачити ці події, потрібно підписатися на них.

Створення підписки на подію WMI визначає наступні кроки:

1. Створення WQL-запиту. Так само, як і під час запиту статичних даних, ви повинні створити WQL-запит, який відповідатиме типу події WMI, яку ви хочете бачити. Але, на відміну від запитів до сховища даних, у запиті потрібно використовувати кілька складніших компонентів, наприклад, системні класи та цикли перевірки (про них далі).
2. Створення фільтра подій. Після того як ви створили запит WQL, ви повинні створити фільтр подій. Фільтр подій реєструє запит WQL в CIM.
3. Створення споживача – споживач визначає дію, яку потрібно виконати, коли запит фільтра подій повертає зміну в класі. Наприклад, щоразу,

коли статус служби запускається, зупиняється, створюється або видаляється, споживач ініціює дію.

4. Прив’язування фільтра подій до споживача – клей, який зв’язує запит Windows WMI із споживачем. Прив’язка сповіщає споживача, коли фільтр подій отримує збіг.

Системні класи – це внутрішній клас, який представляє тип зміни, яку зазнає подія. Подія WMI має чотири типи системних класів:

- InstanceModificationEvent – перевіряє будь-які зміни значення властивості для екземпляра в класі.
- InstanceCreationEvent – перевіряє наявність нових екземплярів.
- InstanceDeletionEvent – іперевіряє наявність видалених екземплярів.
- InstanceOperationEvent – цей системний клас перевіряє всі типи подій, зміни, створення та видалення.

1.2. Аналіз наявних рішень

Endpoint Protector – це кросплатформне програмне забезпечення, що пропонує функціонал для блокування, контролю та моніторингу USB-портів. За допомогою веб-інтерфейсу користувач може віддалено управляти безпекою USB-портів. Програма також може надсилати сповіщення безпеки, коли на системі запускаються вірусні програми, задавати політики безпеки, аналізувати типи файлів та віддалено шифрувати носіїв інформації[15].

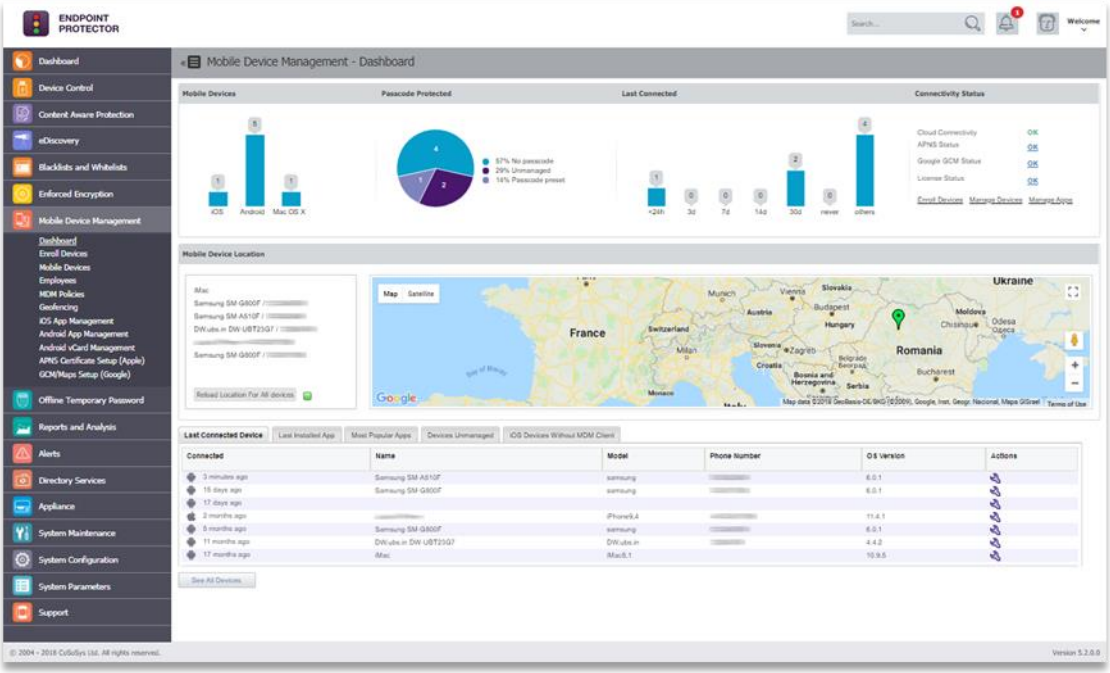

Рисунок 1.2 - Endpoint Protector.

Аналіз недоліків

Система має громіздкий інтерфейс з великою кількістю параметрів для налаштування, що робить її більш складною у використані. При втраті з'єднання між агентом і сервером не відбувається відправлення логів про інциденти на сервер. Сканування запущених програм для виявлення вірусів може значно сповільнювати систему та перешкоджати нормальній роботі

користувачів. Шифрування USB-накопичувачів є додатковою опцією програми, але її можливості програють стандартним утилітам.

Device Control Plus – це комплексне програмне забезпечення для запобігання витоку даних, що дозволяє користувачам централізовано контролювати використання USB та периферійних пристроїв в корпоративній мережі[15]. Програма дає можливість задавати список довірених пристроїв, контролювати доступ на основі ролей та передачу файлів спираючись на їх тип, розмір тощо. Ще одна цікава особливість полягає в тому, що IT-адміністратор, можете надати користувачам тимчасовий доступ для передачі або копіювання файлів на зовнішні USB-пристрої.

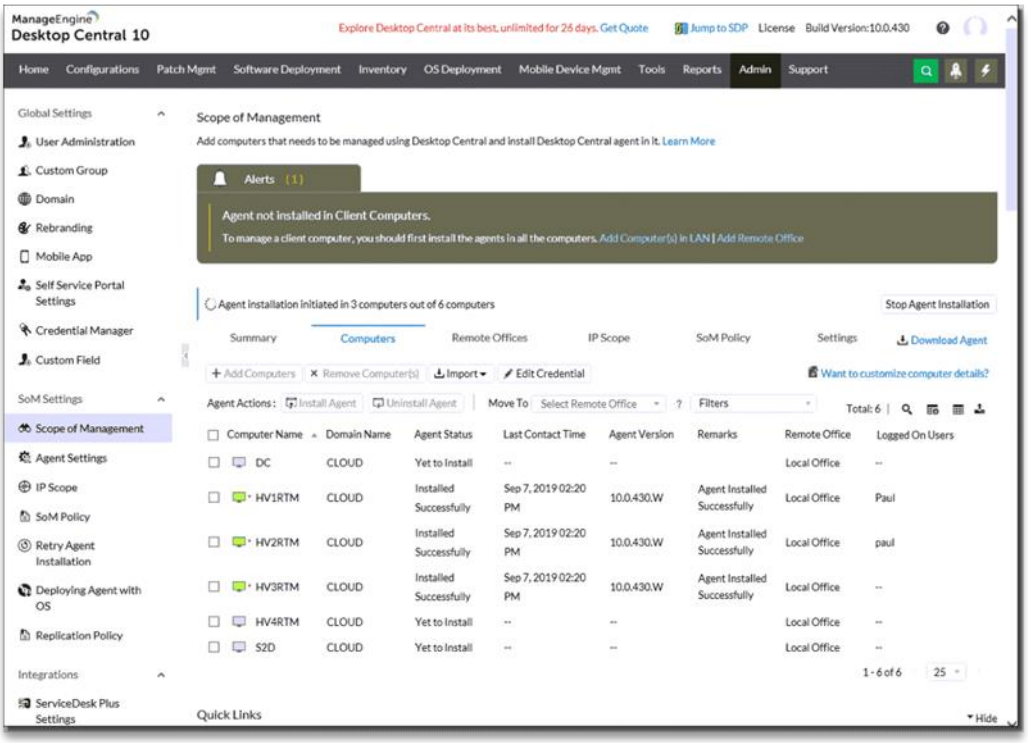

Рисунок 1.3 - Device Control Plus.

Аналіз недоліків

При значній кількості функціоналу програма має погано продуманий інтерфейс. Користувачу складно знайти потрібні йому налаштування без детального вивчення системи.

MacAfee DLP захищає інтелектуальну власність або фінансові дані від несанкціонованого доступу за допомогою політик безпеки. Також має функціонал, що дозволяє управляти процесом копіювання конфіденційних даних на зміні пристрої. Користувач може відслідковувати події безпеки в реальному часі з можливістю генерування звітів для зовнішнього або внутрішнього користування. Програма дозволяє створювати списки дозволених пристроїв на основі яких відбувається блокування небажаних пристроїв[16]. Окрім функцій направлених на недопущення витоку даних через зміні носії інформації, програма захищає від вірусів, спаму та фішингових атак.

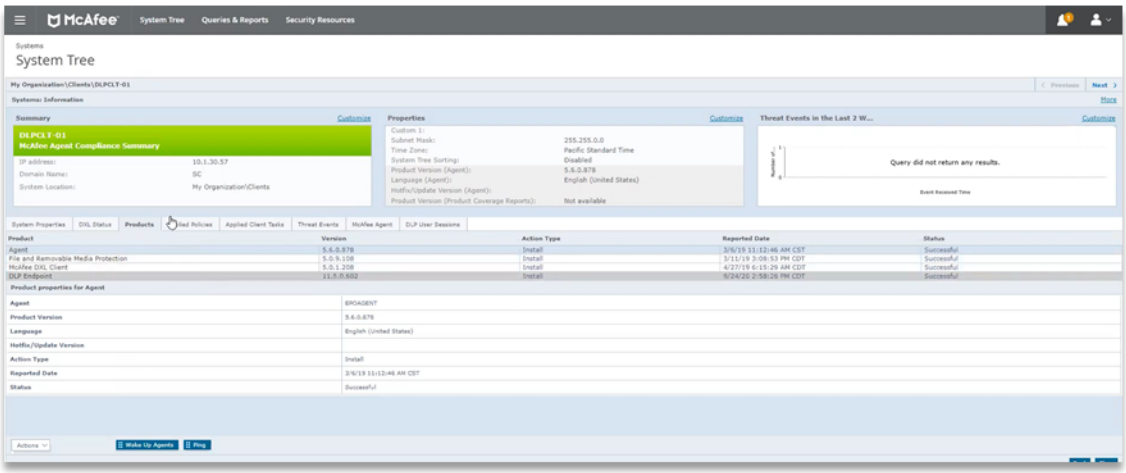

Рисунок 1.4 - MacAfee DLP.

Аналіз недоліків

MacAfee DLP гарний інструмент для попередження витоку інформаії, але воно містить свої недостачі. Системи складна у встановленні, управлінні та обслуговуванні. Дане рішення поєднує у собі не тільки функціонал для контролю використання зовнішніх пристроїв, але і функції антивірусного забезпечення, що змушує користувачів платити за додаткові функції. Інтерфейс погано пристосований для задання правив безпеки. Користувач вимушений робити зайві дії та витрачати додатковий час для вивчення принципів роботи з системою.

DriveLock – це програмне рішення, що дозволяє управляти пристроями та контролювати їх транзакції, що зменшує ризики витоку інформації[16]. Програма здатна запобігати передачу даних через незашифровані носії, має різні можливості, такі як керування пристроями, контроль додатків, аналітика та криміналістична експертиза, машинне навчання, керування Bitlocker, шифрування, керування ідентифікацією та доступом тощо.

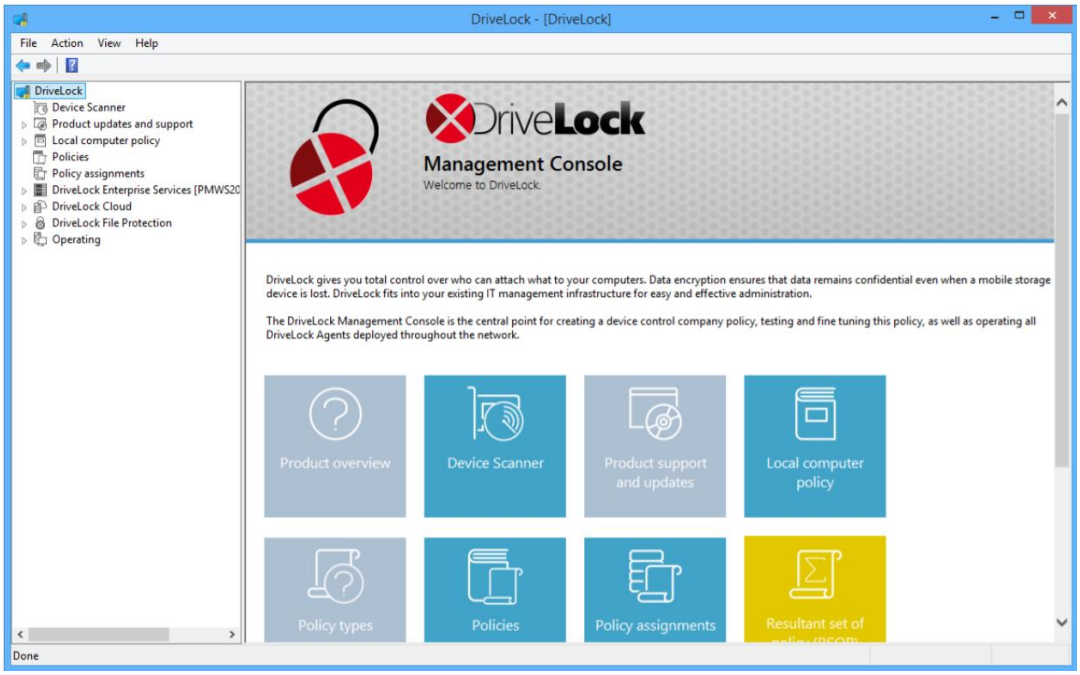

Рисунок 1.5 – DriveLock.

Аналіз недоліків

Недоліком DriveLock є те, що контроль використання USB-пристроїв здійснюється лише шифруванні без можливість задавати списки дозволених пристроїв[15]. Система представляється як хмарний сервіс без можливості розгорнутися локально. Алгоритми машинного навчання, хоча і додають додаткового захисту, але вірогідність хибного спрацювання або пропущення інциденту залишається досить високою. Програма має надлишковий функціонал, що вирішує задачі непов’язані з контролем використання USB-пристроїв.

DeviceLock – це програма для запобігання витоку конфіденційної інформації, що надає користувачам детальний контроль над повним спектром шляхів витоку даних на контекстному рівні[16]. Поряд з функціями контролю доступу до пристроїв, вона містить функції керування мережевими комунікаціями, фільтрації вмісту, виявлення вмісту тощо. Ця програма дозволяє керувати локальною синхронізацією, що дає можливість легко налаштовувати детальний контроль доступу і правил блокування даних між синхронізованими пристроями. За допомогою DeviceLock адміністратори можуть контролювати групу користувачів, які мають доступ до адаптерів USB, FireWire, WiFi і Bluetooth. Також програма дозволяє налаштувати задавати режим «лише читання» для певних пристроїв та контролювати доступ до пристроїв залежно від часу доби та дня тижня.

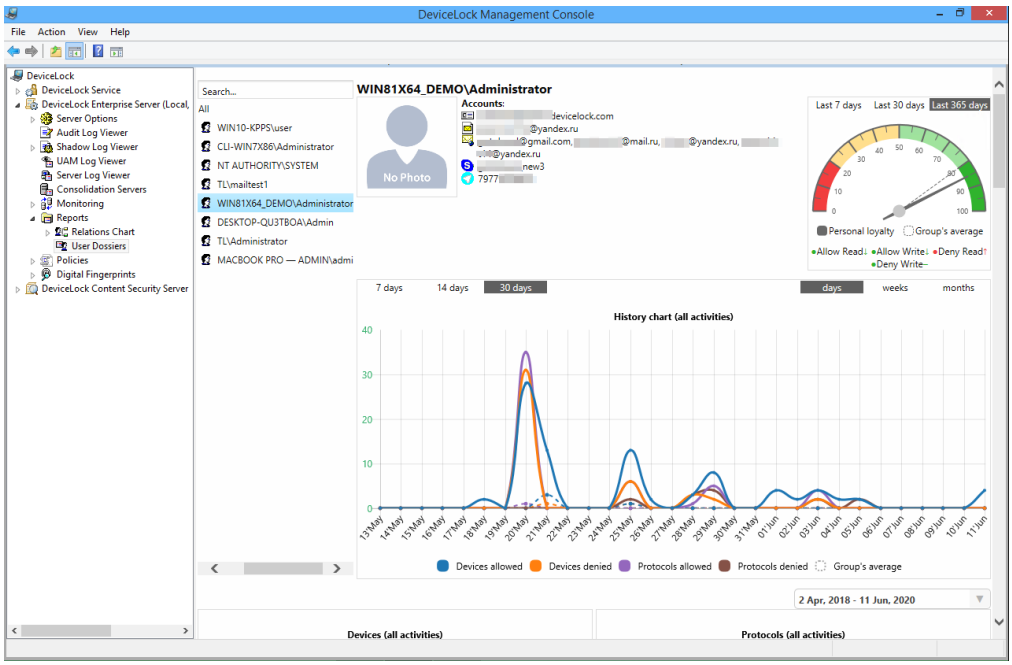

Рисунок 1.6 – DeviceLock.

Аналіз недоліків

Однак дане рішення має ряд недоліків. Програма не має веб-консолі для централізованого управління безпекою, а десктопний додаток потребує встановлення. Система не має можливості задавати списки дозволених

пристроїв. Управління безпекою через групи складніше для розуміння та потребує додаткового часу для опанування.

Ivanti Device Control – рішення для керування пристроями, яке дозволяє легко застосовувати політику безпеки і шифрувати дані на знімних пристроях[17]. Ця програма може захистити кінцеві точки від шкідливого програмного забезпечення, надає функції тимчасового доступу, централізованого управління, забезпечує видимість і контроль над пристроями з доступом до кінцевих точок, таких як USB-накопичувачі, принтери тощо. Програма має можливість задавати списки дозволених пристроїв, контролювати доступ на основі ролей та надавати користувачам тимчасовий або запланований доступ до знімних пристроїв.

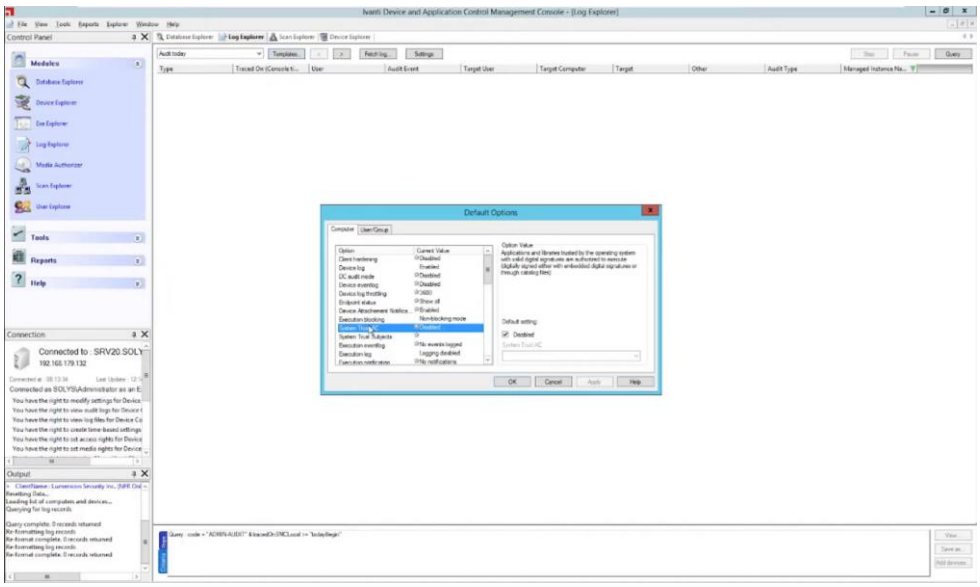

Рисунок 1.7 – DeviceLock.

Аналіз недоліків

Хоча система має не погані характеристик, але вона має свої мінуси. Система не має веб-панелі, яка значно спрощує доступ до налаштувань безпеки, має багато параметрів для налаштувань, не дозволяє відслідковувати події підключення USB-накопичувачів, інтерфейс системи складний та незручний у використанні.

Endpoint Security – це повнофункціональне програмне забезпечення, яке сумісне з операційною системою Windows та має такий функціонал, як журнали активності для проведення аудиту, дозволяє керувати пристроями, шифрувати зміні носії інформації та задавати білі/чорні списки[16]. Метод блокування доступу пристроїв на основі білих/чорних списків попереджує несанкціоноване використання USB-пристроїв, що зменшує вірогідність проникнення зловмисників в корпоративну мережу та розповсюдження шкідливого програмного забезпечення.

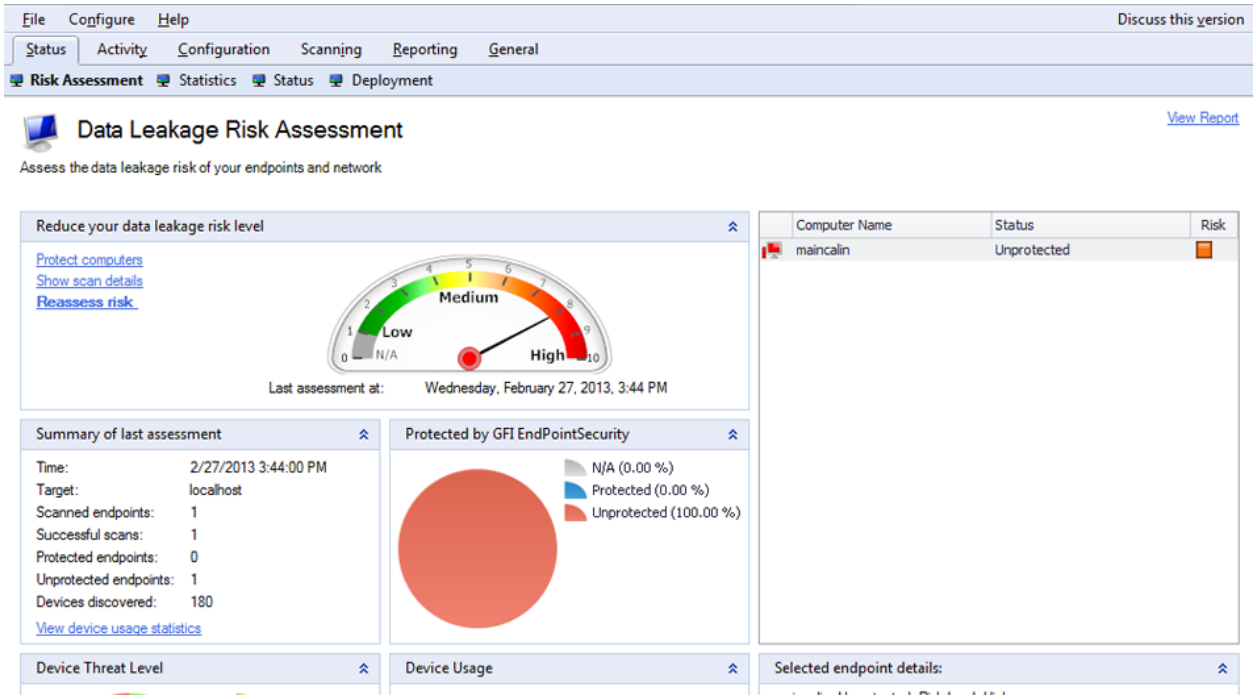

Рисунок 1.8 – Endpoint Security.

Аналіз недоліків

Endpoint Security працює, як хмарний сервіс та не має можливості розгортання на власних ресурсах. Графічний інтерфейс непростий у застосуванні. Він має велику кількість конфігурацій без детального пояснення, складно створювати політики безпеки. Додаткова функція брандмауера працює гірше ніж брандмауер Windows, що робить його використання недоцільним. Цей продукт є платним, але вартість не знаходиться у відкритому доступі.

2. ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР МЕТОДІВ РЕЛІЗАЦІЇ

2.1. Постановка задачі та визначення мети

Метою роботи є створення інформаційної технології, яка дозволить забезпечити централізований контроль та управління використання USB-накопичувачів, що дозволить зменшити ризики втрати інформації.

Технологія повинна давати можливість:

- задавати списки дозволених пристроїв на основі яких буде прийматися рішення;
- переглядати список підключених USB-накопичувачів, які використовуються в корпоративній мережі;
- блокувати USB-накопичувачі, що використовуються на комп’ютері;
- відслідковувати історію підключень носіїв інформації;
- централізованого управляти безпекою, за допомогою інтуїтивно зрозумілого веб-застосунку;
- розгортання на власних серверах;

Для досягнення поставленої мети треба вирішити наступні задачі:

- провести аналіз предметної області;
- обрати засоби та методи для вирішення задачі;
- виконати проектування системи;
- здійснити програмну реалізацію.

2.2. Вибір методів реалізації

На даний момент існує безліч технологій розробки програмного продукту, які мають свої переваги та недоліки при вирішенні поставленої задачі. Тому дуже важливо обрати той інструментарій, який найкраще задовольнить потреби проекту.

Операційна система Windows є найбільш розповсюдженою у світі, тому саме їй найчастіше відають перевагу компанії будь-якого розміру. Отже, вкрай необхідно, щоб фонова служба, яка буде встановлюватися на комп'ютери в корпоративній мережі, була сумісною з операційною системою Windows. Для досягнення цієї мети, найкраще підходить платформа .NET 5, яка була розроблена компанією Microsoft. Платформа .NET 5 має повний інструментарій для створення довготривалих програм, які запускаються під час власних сеансів Windows. Ці служби можуть запускатися автоматично під час завантаження комп'ютера, їх можна призупинити та перезапустити. Також служби можна запускати в контексті безпеки певного облікового запису користувача, який увійшов у систему, або облікового запису комп'ютера за замовчуванням. Платформа .NET 5 підтримує більше однієї мови програмування, компілюючи код в збірку на спільній мові CIL[18]. Основна мова програмування платформи є C#, яка була розроблена компанією Microsoft. C# - це широко розповсюджена, об'єктно-орієнтована, строго типізована мова програмування. Завдяки середовищу виконання Common Language Runtime вона дозволяє писати код набагато простіше порівняно з C та C++, автоматизуючи частину низькорівневих задач[18]. Ця мова добре задокументована, має короткий цикл оновлень та велику спільноту.

Наступною вимогою до системи є те, щоб вона була здатна централізовано управляти безпекою USB-портів та відслідковувати, які

пристрої використовувалися у корпоративній мережі. Для задоволення цієї потреби необхідно розробити SPA та WEB-API.

На сьогоднішній день для розробки SPA найчастіше використовують бібліотеку React. React – це бібліотека, яка розроблена на базі JavaScript з відкритим кодом для створення користувацьких інтерфейсів. React був створений компанією Facebook. Ця бібліотека спирається на концепцію використання компонентів, що дозволяє перевикористовувати код у різних проектах. Для визначення коду інтерфейсу React використовую JSX, що дозволяє писати HTML подібний код разом з компонентами[19]. Також, ця бібліотека має високу продуктивність завдяки концепції віртуального DOM. Віртуальний DOM зберігає копію звичайного DOM. При необхідності оновити елемент інтерфейсу спочатку порівнюється та замінюється віртуальний DOM, і при виявленні розбіжностей оновлює основний DOM з мінімальною кількістю затрат[20]. Отже, такий спосіб взаємодії з елементами набагато ефективніший, ніж напряму змінювати DOM через JavaScript. React використовує однонаправлену передачу даних, тобто з батьківських компонентів до компонентів нащадків. Дані, які були передані не можуть бути змінені напряму компонентом нащадком, але при необхідності можна застосувати callback-функції.

Для створення API платформа .NET 5 містить ASP.NET Core фреймворк[18]. ASP.NET Core — це веб-фреймворк з відкритим вихідним кодом і оптимізований для хмари для розробки сучасних веб-додатків, які можна розробляти та запускати на Windows, Linux та Mac[21]. Він включає фреймворк MVC, який тепер поєднує функції MVC і Web API в єдину платформу веб-програмування. ASP.NET Core є кросплатформним і ідеально працюватиме на Windows, Linux і Mac, а також на всіх пристроях. Система гнучка і дозволяє розробникам вибирати будь-яку ОС для своєї зручності. У ASP.NET Core розробники можуть писати набагато менше операторів[22]. Що, у свою чергу, призведе до спрощення структури коду та меншого

кодування. Для бізнесу це робить розробку додатків більш економічно ефективною. Для розробників це дає більше контролю над процесом і спрощує налагодження. Підтримка ін’єкції залежностей використовується для проектування інтеграції в програму. Завдяки цьому ASP.NET Core надає розробникам веб-додатків покращену можливість тестування та розширення. Більше не потрібно використовувати сторонні фреймворки для інтеграції необхідного дизайну в додаток. Підтримка ін’єкції залежностей використовується для проектування інтеграції в програму.

Завдяки цьому ASP.NET Core надає розробникам веб-додатків покращену можливість тестування та розширення. Їм більше не потрібно використовувати сторонні фреймворки для інтеграції необхідного дизайну в додаток. ASP.NET Core має вбудовані функції, які дозволяють розробникам створювати високобезпечні веб-додатки. Завдяки цій технології легше підтримувати застосування HTTPS, запобігання XSRF/CSRF, аутентифікацію, авторизацію та захист даних[23].

Rapid Development ідеально підходить для проектів, які потрібно випустити за короткий проміжок часу (2-3 місяці). Це модель розробки, де більше уваги приділяється завданням розробки та прототипу, ніж плануванню. Модель гнучка та адаптована до змін, знижує загальний ризик проекту, ручне кодування та помилки. Модель швидкої розробки додатків можна легко використовувати для ASP.NET Core.

Відкритий код означає, що будь-який експерт ASP.NET Core має доступ до коду фреймворку та репозиторіїв на GitHub. Уся спільнота .NET Core може працювати над удосконаленням технології та змінювати її відповідно до потреб розробки веб-програм. Отримавши найкраще від ASP.NET Core, розробники можуть створювати найкращі веб-рішення.

3. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1. Розробка моделі варіантів використання

На етапі моделювання важливо явно виділити систему від оточуючого середовища, визначити всіх акторів, які будуть взаємодіяти з системою та визначити перелік основного функціоналу системи. Для досягнення поставленої мети була використана діаграма варіантів використання, яка зображена на рисунку 3.1.

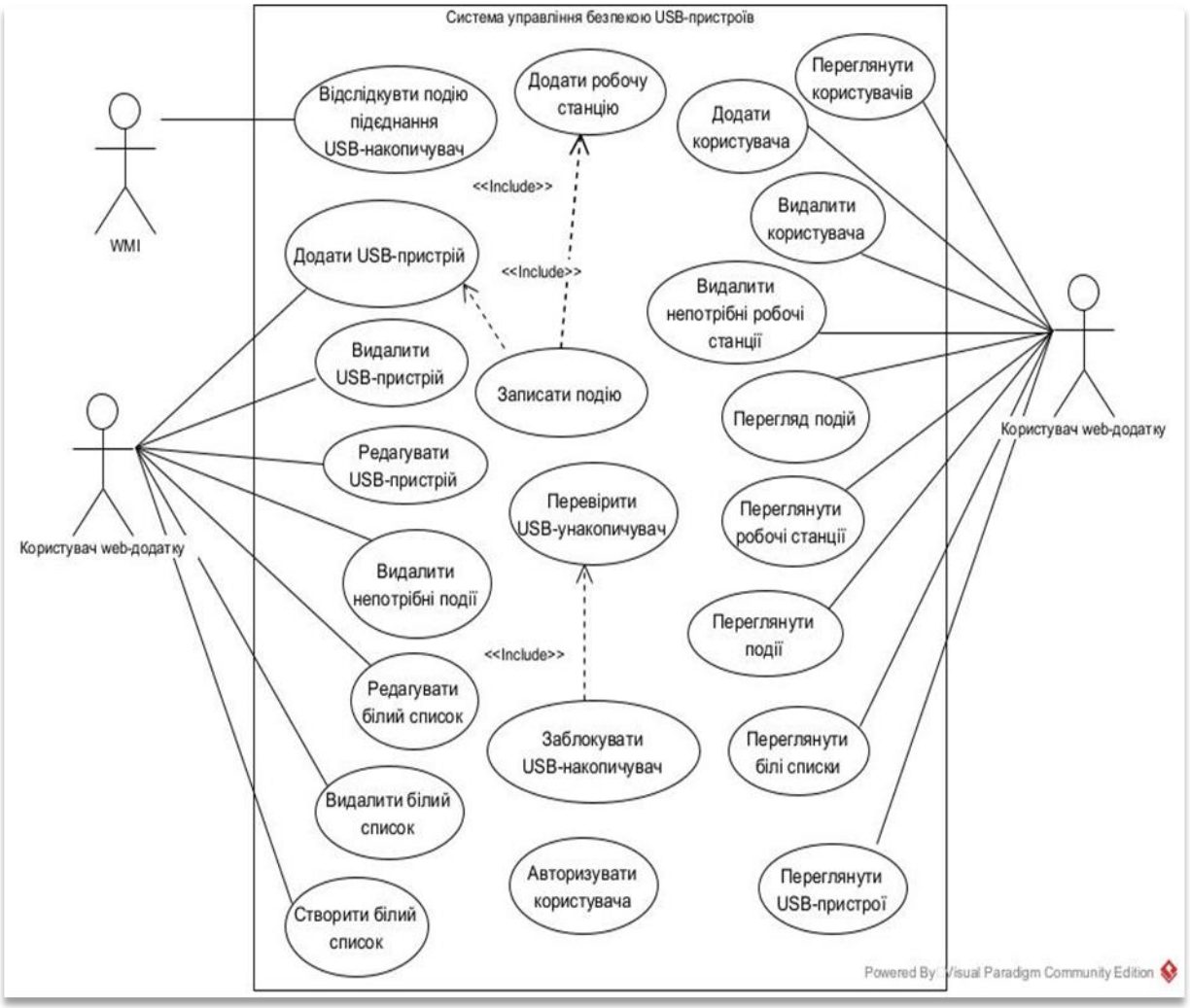

Рисунок 3.1 – Діаграма варіантів використання.

Ця діаграма зображує перелік наявного функціоналу та акторів, що взаємодіють з системою.

3.2. Розробка прототипу інтерфейсу користувача

Прототип – дозволяє визначити основний функціонал інтерфейсу користувача та оцінити дизайн. Прототип також дає можливість продумати ідеї без фактичної програмної реалізації. Завдяки цьому можна значно зменшити час розробки системи та зробити її більш зручною для користувача.

Інтерфейс системи повинен забезпечувати взаємодію між користувачем та робочими станціями. Користувач повинен мати змогу переглянути стан безпеки в мережі та прийняти необхідні заходи для його покращення. На рисунку 3.2 зображений прототип головної сторінки, яка дозволяє користувачу переглянути список робочих станцій в системі, який організовано у вигляді таблиці з основною інформацією.

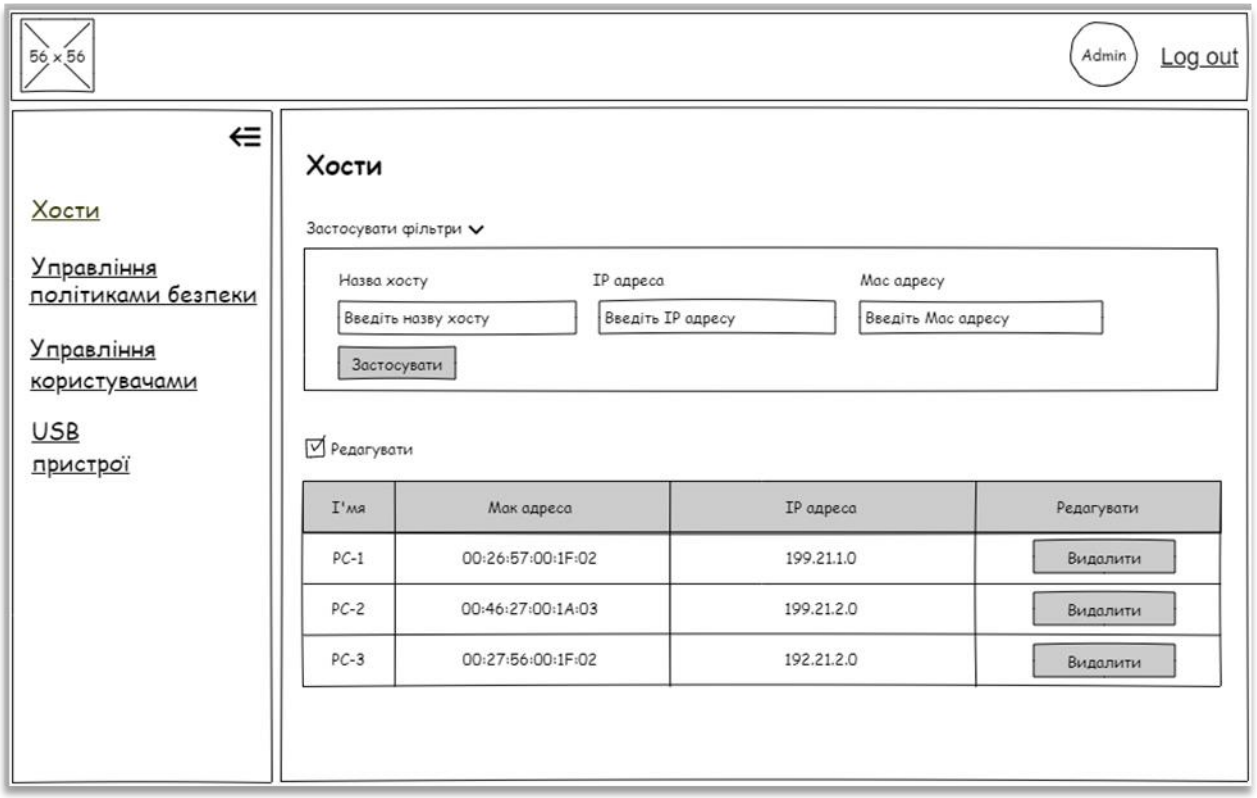

Рисунок 3.2 – Прототип головної сторінки.

Користувач буде мати змогу переглянути мак-адресу, IP-адресу та назву робочої станції. При значній кількості комп’ютерів в корпоративній мережі виникає необхідність здійснювати пошук. Пошук потрібної робочої станції буде відбуватися за допомогою прихованої панелі, яку потрібно розгорнути. Даний підхід до побудови програмного інтерфейсу дозволить не відволікати користувача від отримання основної інформації. У разі необхідності користувач матиме змогу видалити непотрібну робочу станцію. Даний функціонал буде прихований на початковому етапі та для його активації необхідно поставити галочку у полі редагувати, після чого в таблиці буде з’являтися додаткове поле з опціями для редагування. На цій сторінці користувачу буде доступною лише опція для видалення нерелевантного запису.

Натискаючи на кожний рядок в таблиці користувач матиме змогу перейти на сторінку перегляду подій для відповідної робочої станції прототип, якої можна побачити на рисунку 3.3.

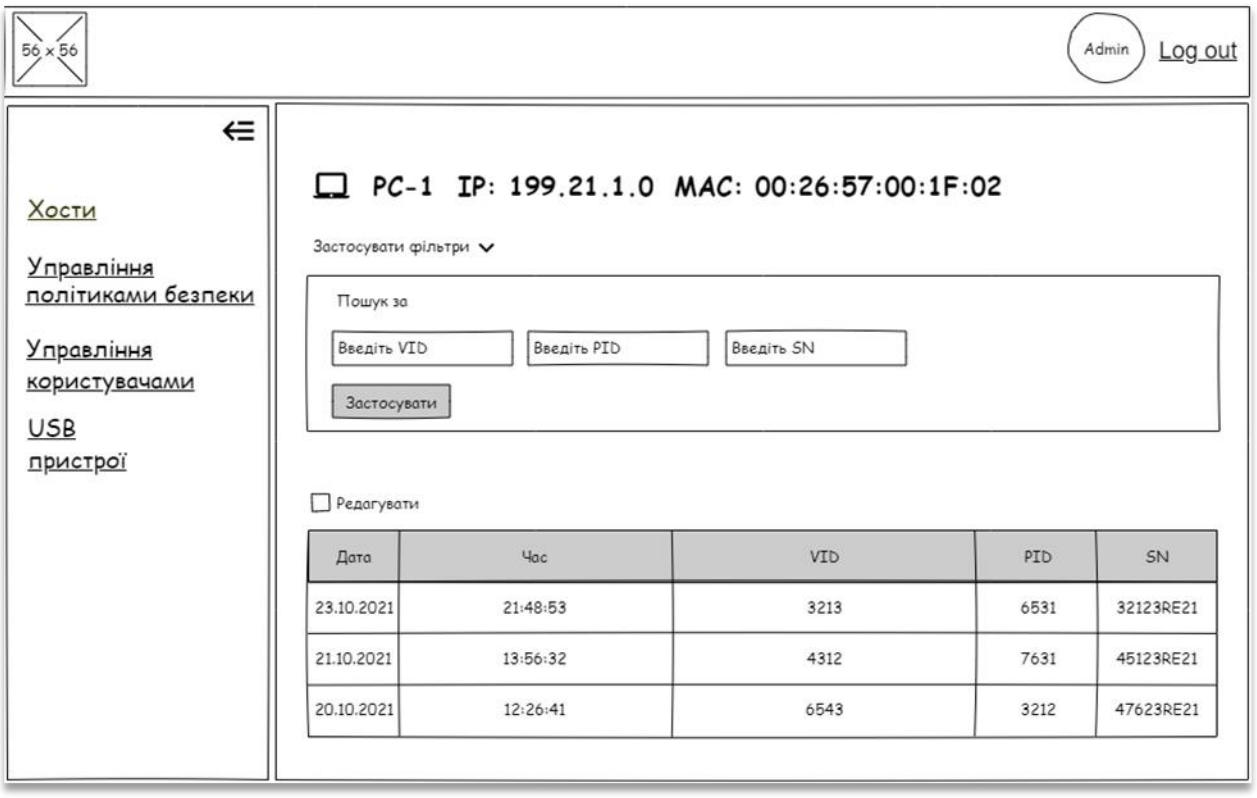

Рисунок 3.3 – Прототип сторінки перегляду подій.

В заголовку сторінки буде відображатися інформація про робочу станцію до якої належать події, а саме назва робочої станції, мак-адреса та IP-адреса. Основними компонентами цієї сторінки є таблиця перегляду подій, які відбулися на відповідній робочій станції. Користувачу буде мати можливість переглядати інформацію про USB-накопичувач, час та дату, коли він був під’єднаний або від’єднаний до робочої станції, та чи був даний накопичувач заблокований, чи ні. Користувач також зможе здійснювати подій за допомогою блоку, що розгортається. Також на сторінці буде можливість видаляти непотрібні події.

Для того, щоб користувач мав змогу управляти безпекою в корпоративній мережі, інтерфейс повинен надавати можливість визначати політики безпеки. На рисунку 3.4 зображений прототип сторінки управління політиками безпеки.

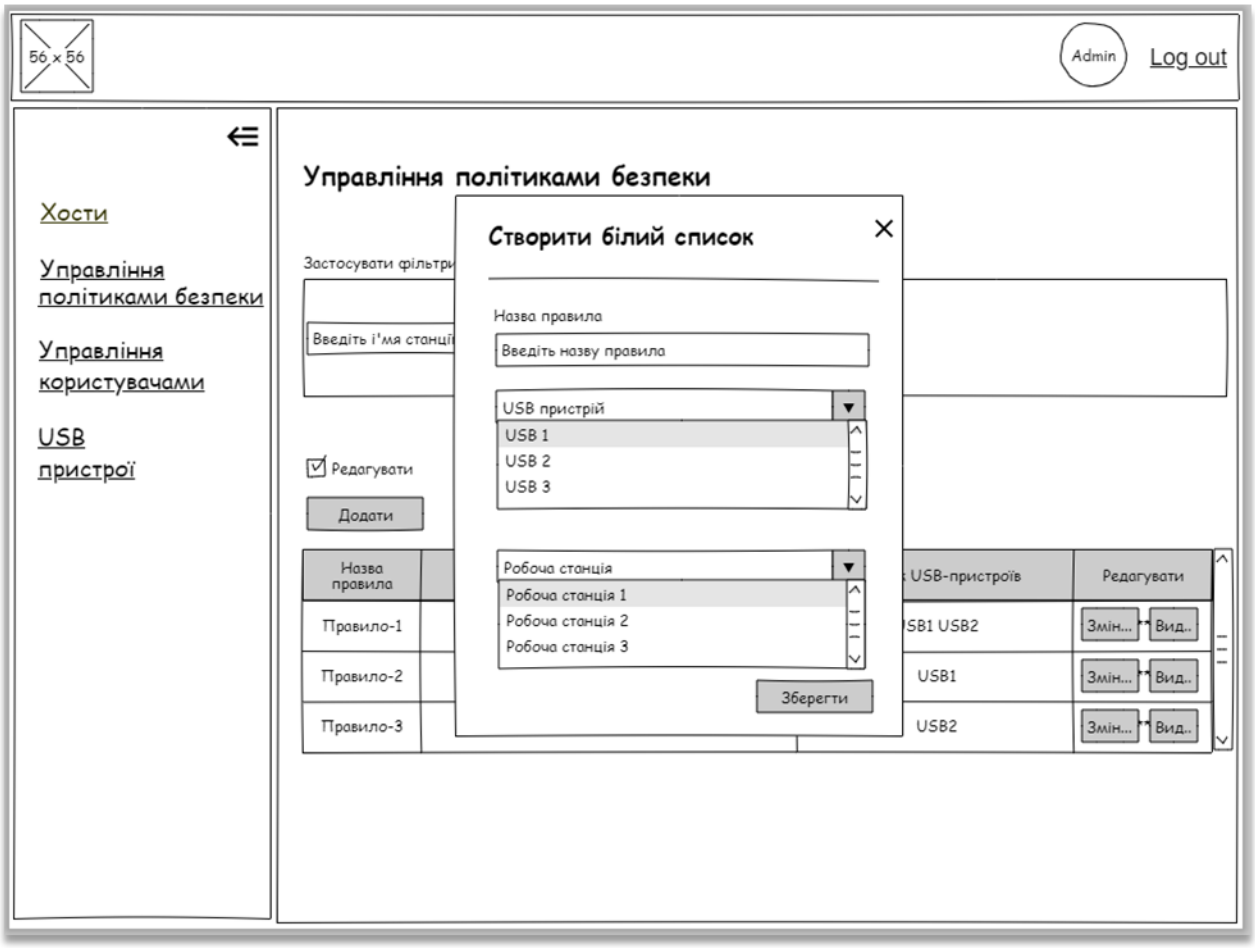

Рисунок 3.4 – Прототип сторінки управління політиками безпеки.

На цій сторінці користувач матиме можливість здійснювати пошук. Дана сторінка буде мати функціонал для створення, редагування та видалення правил. Натискаючи на кнопку додати або змінити, буде відкриватися модальне вікно, де користувач матиме змогу створити нове правило або редагувати вже існуюче. Правило буде задаватися шляхом вибору із вже існуючих даних в системі про USB-накопичувачі та робочі станції. Можливості редагування правил приховані на початковому стані. Для їх активації необхідно поставити галочку у полі редагувати.

На рисунку 3.5 зображений прототип сторінка управління користувачами. Функціонал даної сторінки збігається з функціоналом сторінки управління правилами безпеки. Користувач матиме можливість додавати, видаляти та змінювати дані про користувача. При значній кількості записів користувач має змогу їх відфільтрувати. Принцип роботи компонентів інтерфейсу зберігається.

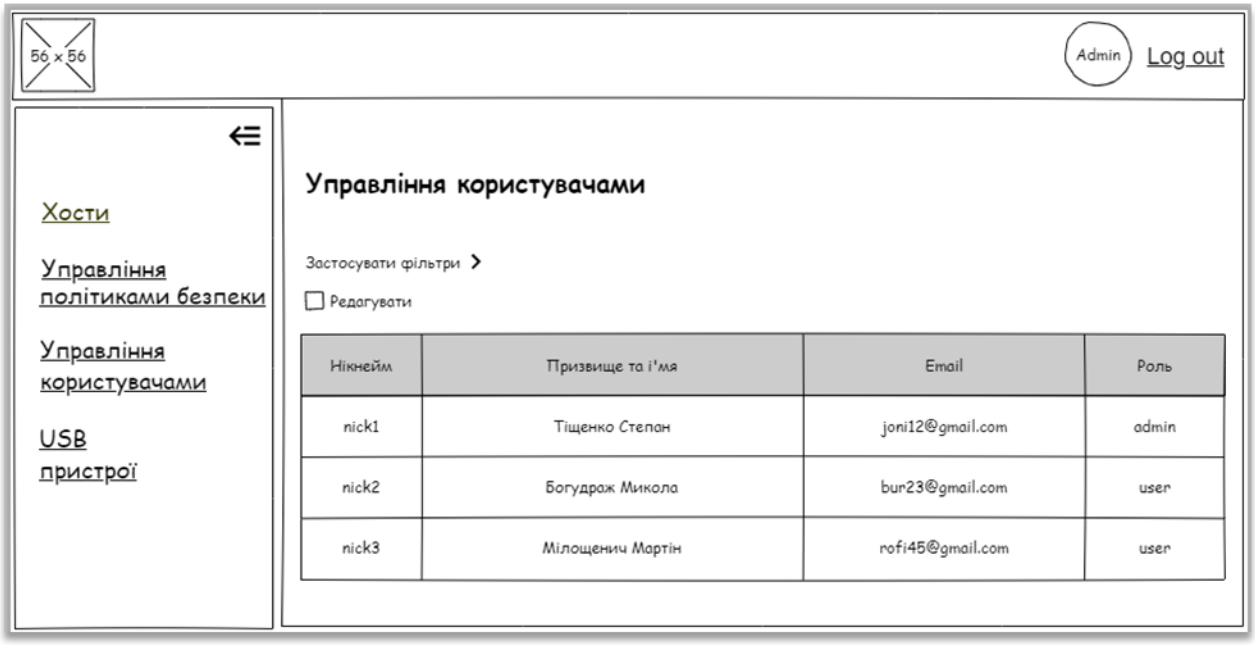

Рисунок 3.5 – Прототип сторінки управління користувачами.

Дані про пристрої, які використовуються в корпоративній мережі, повинні додаватися до системи при кожному під’єднанні USB-накопичувача до робочої станції та самим користувачем. На рисунку 3.6 відображається

прототип сторінки, що дозволяє здійснювати управління USB-пристроями. Принцип роботи та набір функціоналу залишається незмінним.

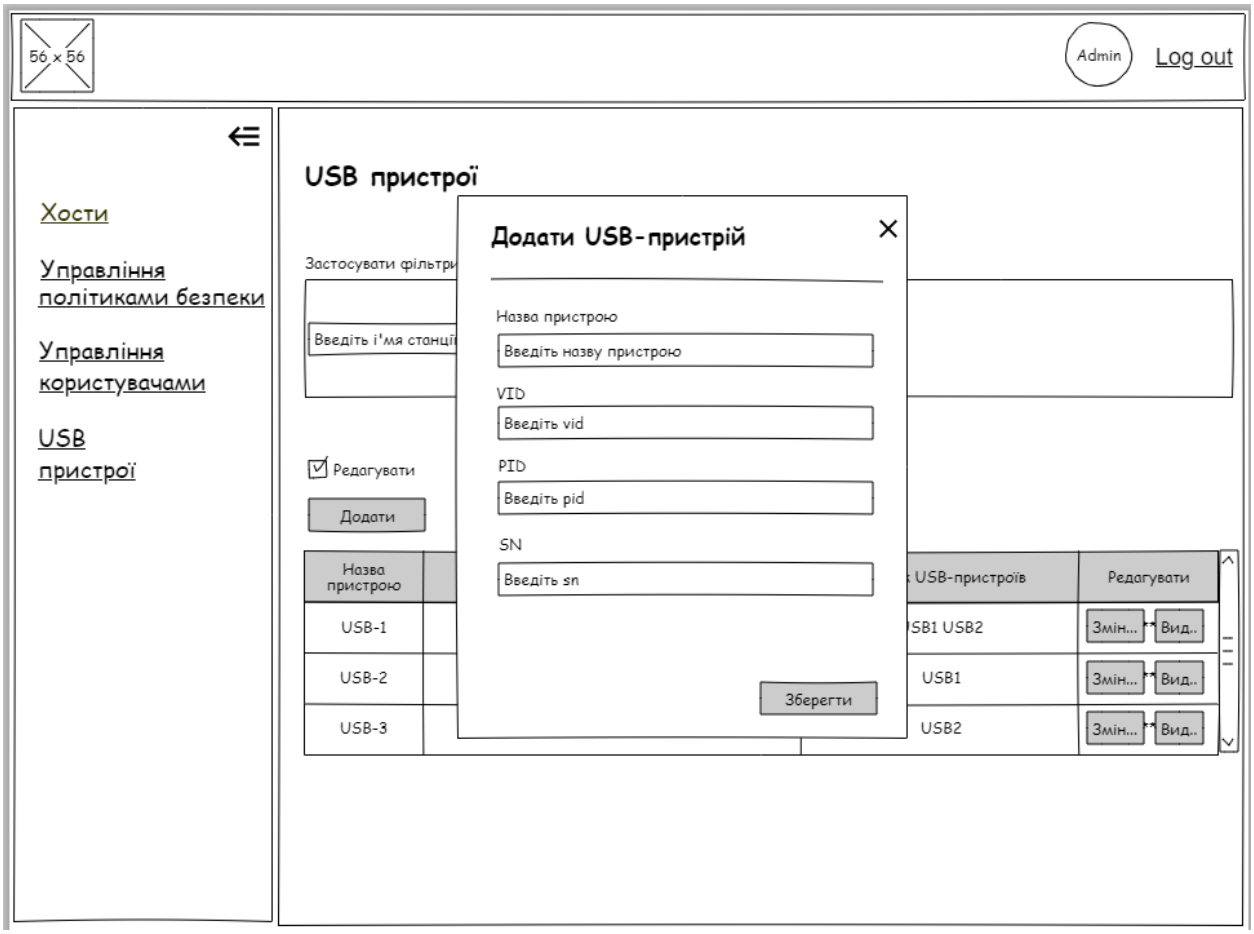

Рисунок 3.6 – Прототип сторінки, що дозволяє здійснювати управління USB-пристроями.

3.3. Розробка ER-діаграми

Для того, щоб спроектувати базу даних було використано ER-діаграму. Це дозволить визначити предметну область, виокремити головні сутності та встановити зв’язки між ними. На рисунку 3.6 побудована ER-діаграма, що відповідає потребам системи. Для уникнення потенційних суперечностей між даними модель було приведено до третьої нормальної форми.

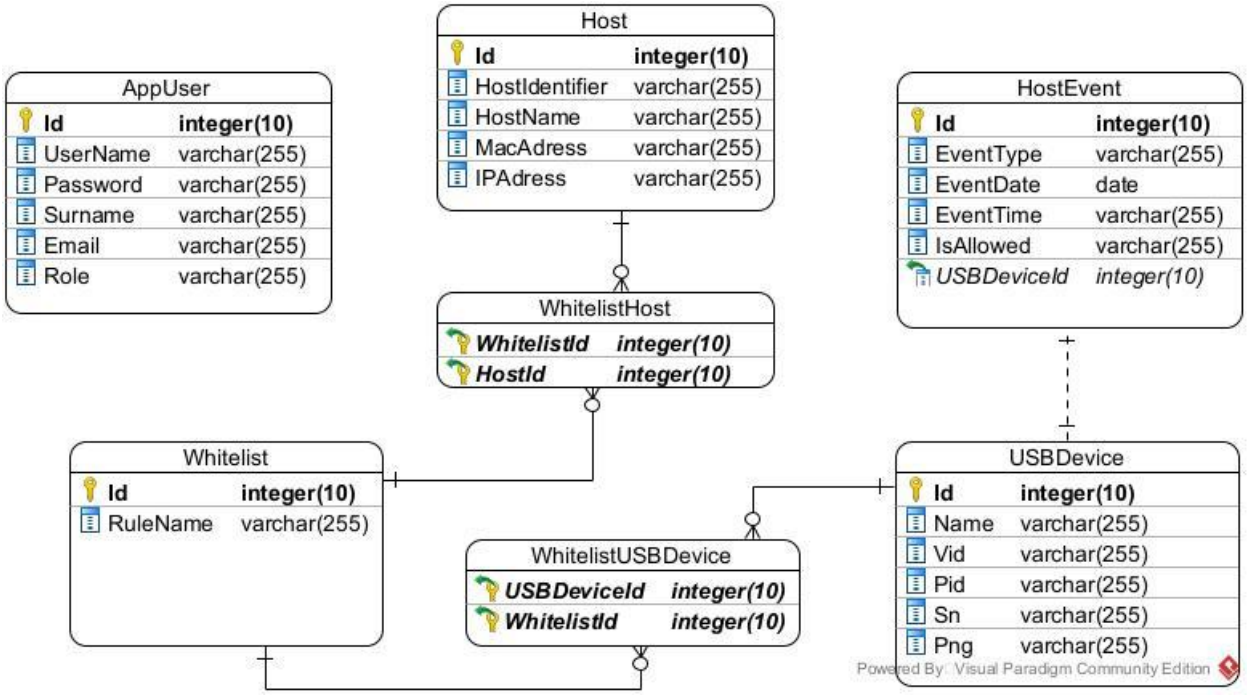

Рисунок 3.6 – ER-діаграма.

Таблиця 3.1. – Опис сутностей та їх атрибутів.

Таблиця	Поле	Ключі	Зміст
AppUser	Id	PK	Унікальний ідентифікатор сутності.
	UserName		Нікнейм користувача.
	Name		Ім’я користувача.

Таблиця	Поле	Ключі	Зміст
	Surname		Прізвище користувача.
	Email		Адреса почтової скриньки користувача.
	Password		Пароль користувача.
	Role		Права доступу користувача до системи.
Host	Id	PK	Унікальний ідентифікатор сутності.
	HostIdentifier		Ідентифікатор робочої станції.
Host	HostName		Ім’я робочої станції.
	MacAdress		Мак-адреса робочої станції.
	IPAdress		IP-адреса робочої станції.
HostEvent	Id	PK	Унікальний ідентифікатор сутності.
	EventDate		Дата коли сталася подія.
	EventTime		Час коли сталася подія.
	IsAllowed		Містить інформацію про те, чи був заблокований пристрій.
	USBDeviceId	FK	Унікальний ідентифікатор сутності «USBDevice».
USBDevice	Id	PK	Унікальний ідентифікатор сутності.
	Name		Назва пристрою.
	Pid		PID накопичувача.
	Vid		VID накопичувача.

Таблиця	Поле	Ключі	Зміст
	Sn		Серійний номер USB накопичувача.
	Png		Унікальний ідентифікатор для пошуку пристрою в системі.
WhitelistUSBDevice	USBDeviceId	FK	Унікальний ідентифікатор сутності «USBDevice».
	WhitelistId	FK	Унікальний ідентифікатор сутності «Whitelist».
Whitelist	Id	FK	Унікальний ідентифікатор сутності
	RuleName		Назва правила
WhitelistHost	WhitelistId		Унікальний ідентифікатор сутності «Whitelist».
	HostId		Унікальний ідентифікатор сутності «Host».

4. РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

4.1. Програмна реалізація клієнтської служби

Клієнтська служба виступає в якості агенту, що встановлюється на робочу станцію. Дана служба працює у фоновому режимі та забезпечує постійне прослуховування порту на підключення USB-пристроїв. Для її створення було використано платформу .NET 5, яка є найбільш сумісною з операційними системами Windows[22]. Для створення фонових служб дана платформа містить проектний шаблон Worker Service, що дає початкову точку для розробки довгострокових фонових сервісів.

Основною складовою служби є технологія Windows Management Instrumentation, що надає програмний інтерфейс для взаємодії з операційною системою Windows. WMI дає можливість відслідковувати зміни в системі за допомогою об’єктів, які представляють відповідні компоненти в системі. У нашому випадку, для прослуховування було обрано об’єкт Win32_USBHub. Події прослуховуються за допомогою класу ManagementEventWatcher.

Для відслідковування подій було виконано наступні дії:

1. Отримано об’єкт Win32_USBHub за допомогою WMI Query Language.
2. Створено об’єкт класу ManagementEventWatcher, який здійснює спостереження.
3. Зареєстровано функцію обробника подій.
4. Запущено процес спостереження.

Після того, як подія підключення або відключення трапилась, служба відправляє запит на сервер для перевірки USB-пристрою. Для досягнення цієї мети було реалізовано можливість мережевої взаємодії, за допомогою класу TcpClient. Цей клас базується на TCP протоколі, що гарантує доставку

повідомлень. Щоб встановити мережевий зв'язок між клієнтом та сервером, даний клас містить метод TcpClient.Connect, який приймає в якості параметрів порт до якого потрібно підключитися та IP-адреса або доменне ім’я віддаленого серверу. Після успішного підключення ми можемо отримати об’єкт класу NetworkStream, через який відбувається подальша взаємодія між сервером та клієнтом.

Після перевірки було реалізовано можливість блокування небажаних USB-пристроїв. Для цього було використано бібліотеку USBClocker, яка містить статичний метод SetDeviceEnabled. В якості параметрів вона приймає guid ідентифікатор пристрою, ідентифікатор, який базується на серійному номері пристрою, та булеве значення, що дозволяє заблокувати або дозволити USB-пристрій. Після того, як було прийнято рішення блокувати, чи дозволити USB-пристрій програма відправляє інформацію про подію на сервер. На сервер відправляються такі параметри, як vid, pid, серійний номер пристрою, тип події та ідентифікатор статусу події.

Для збереження мережевих налаштувань було використано файл конфігурації App.config.

4.2. Програмна реалізація веб-серверу та фонової служби

Після завершення розробки клієнтської служб наступним кроком було реалізація веб-серверу. Для його розробки було використано ASP.NET Core фреймворк, який входить в інфраструктуру платформи .NET 5. Даний фреймворк підтримує два шаблони – Web API, який використовує REST-архітектуру, та MVC[21]. Щоб реалізувати серверну частину системи було використано шаблон Web API. Це дає ряд переваг, такі як можливість реалізації Single Page Application та дозволить в подальшому розробити мобільний додаток без зміни серверної частини.

Для взаємодії між сервером та клієнтом було створено ряд контролерів, що дозволяють виконати певний перелік дій та відправити відповідь назад. REST- архітектура дозволяє використовувати контролери таких типів, як:

- GET
- POST
- PUT
- DELETE

В якості отримання відповідей та надсилання запитів було використано формат JSON.

В таблиці 4.1 нижче наведено опис REST-інтерфейс, який було реалізовано для задоволення потреб системи.

Таблиця 4.1 – Опис розробленого REST-інтерфейсу.

Тип	API інтерфейс	Призначення
GET	/api/Host	Дозволяє отримати список робочих

Тип	API інтерфейс	Призначення
		станцій.
POST	/api/Host/SearchCertainHosts	Дозволяє отримати певні робочі станції.
DELETE	/api/Host/DeleteHost/{id}	Дозволяє видалити непотрібну робочу станцію.
POST	/api/HostEvent/HostEventsSearch	Дозволяє отримати певні події на робочій станції.
GET	/api/HostEvent/{id}	Дозволяє отримати список подій на певній робочій станції.
DELETE	/api/HostEvent/DeleteUsbEvent/{id}	Дозволяє видалити непотрібну подію.
POST	/login	Дозволяє авторизувати користувача.
GET	/api/USBDevice	Дозволяє отримати список USB-пристроїв.
POST	/api/USBDevice/AddUSBDevice	Дозволяє додати USB-пристрій.
POST	/api/USBDevice/SearchCertainUSBDevice	Дозволяє знайти певні USB-пристрої.
PUT	/api/USBDevice/UpdateUSBDevice/{id}	Дозволяє оновити запис про USB-пристрій.

Тип	API інтерфейс	Призначення
DELETE	/api/USBDevice/DeleteUSBDevice/{id}	Дозволяє видалити непотрібний USB-пристрій.
GET	/api/User	Дозволяє отримати список USB-накопичувачів.
POST	/api/User/AddNewUser	Дозволяє отримати нового користувача.
PUT	/api/User/UpdateUser/{id}	Дозволяє оновити користувача.
DELETE	/api/User/DeleteUser/{id}	Дозволяє видалити користувача.
POST	/api/User/SearchCertainUsers	Дозволяє знайти певних користувачів.
GET	/api/Whitelist	Дозволяє отримати список з переліком білих списків.
GET	/api/Whitelist/{id}	Дозволяє отримати конкретний білий список.
POST	/api/Whitelist/AddNewWhitelist	Дозволяє додати новий білий список.
DELETE	/api/Whitelist/DeleteWhitelist/{id}	Дозволяє видалити непотрібний білий список.
PUT	/api/Whitelist/UpdateWhitelist/{id}	Дозволяє оновити білий список.

Після визначення переліку контролерів, наступним кроком була реалізація можливості зберігати інформацію в базі даних. Для взаємодії з базою даних було використано Entity Framework Core 5, представляє собою об’єктно орієнтовану, розширювану технологію для доступу к даним. Даний фрейм є ORM-інструментом, що дозволяє працювати з базами даних на більш високому рівні абстракції[24]. EF Core дає змогу не заглиблюватися в деталі бази даних та працювати з даними без строгої прив’язці до конкретної бази даних, тобто у разі необхідності ми можемо змінити СУБД, змінивши лише строку підключення.

Розробка схеми бази даних відбувалася за допомогою опису таблиць у вигляді класів. Для відображення таблиць безпосередньо в базі даних було використано бібліотеку Microsoft.EntityFrameworkCore.Tools, що дозволяє згенерувати таблиці відповідно заданих класів. Так, як в проекті використовується СУБД PostgreSQL, було використано провайдер Npgsql.EntityFrameworkCore.PostgreSQL. Строку підключення було винесено до файлу appsettings.json, що дозволить підвищити рівень безпеки додатку. Після того, як було описано модель для взаємодії з базами даних, вона була підключена до веб-додатку за допомогою механізму Dependency Injection. Оновлення, запис, видалення та отримання даних відбувається за допомогою мови запитів LINQ, що являє собою просту мову запитів до джерела даних. В даному випадку запити відбуваються до набору даних DataSet.

Щоб реалізувати можливість взаємодії між користувачем та робочими станціями було реалізовано фонову службу, яка працює синхронно з веб-сервером. Для розробки даної служби було використано абстрактний клас BackgroundService, який містить методи для запуску фонових процесів. Прив’язка веб-серверу та фонової служби відбувається за допомогою Dependency Injection, що дозволяє отримати всі сервіси asp.net додатку. Цей механізм дозволив використовувати вже вище описану модель для взаємодії з базами даних. Наступним кроком була реалізація можливості

встановлювати постійне TCP з’єднання з робочими станціями. Постійний TCP зв’язок дозволить користувачу мати постійний доступ до робочих станцій та дозволить відслідковувати момент припинення роботи клієнтської служби. Для цього було використано вбудований клас TcpListener, що дозволяє відкрити порт для прослуховування TCP з’єднань. Щоб почати прослуховування порту було виконано наступні дії:

1. Задано IP-адресу та порт для прослуховування;
2. Створено об’єкт класу TcpListener, за допомогою конструктора, який приймає вище задану IP-адресу та номер порту;
3. У вище створеного об’єкту визвано метод TcpListener.Start() у іншому потоці.
4. Далі з того ж самого об’єкту у циклі while будемо визивати метод TcpListener.AcceptTcpClientAsync(), що чекає на підключення робочої станції до сервера.
5. Після успішного встановлення з’єднання метод TcpListener. .AcceptTcpClientAsync() повертає об’єкт класу NetworkStream, що дозволить зчитувати або відправляти повідомлення.

4.3. Програмна реалізація інтерфейсу користувача

Останнім етапом розробки системи є реалізація веб-інтерфейсу за допомогою якого відбувається взаємодія користувача з робочими станціями. Інтерфейс було реалізовано за допомогою бібліотеки React, яка дозволяє створювати Single Page Application за допомогою декларативного підходу. Даний підхід робить код передбачуваним та простим для відладжування. Всі елементи інтерфейсу було розбито на компоненти, що дозволяє перевикористовувати їх у різних частинах інтерфейсу. Міжсторінкова маршрутизація була реалізована за допомогою окремої бібліотеки React Router, яка надає можливості для навігації за натиском, перенаправлення та доступ до історії переходів. Щоб спростити створення асинхронних запитів на Web API сервер було використано бібліотеку Axios. Також, для пришвидшення розробки веб-інтерфейсу було використано бібліотеку React Bootstrap, що додає адаптивність до сайту з коробки та має набір стилізованих компонентів.

Розробка веб-інтерфейсу почалася зі створення початкової сторінки входу, яку можна побачити на рисунку 4.1

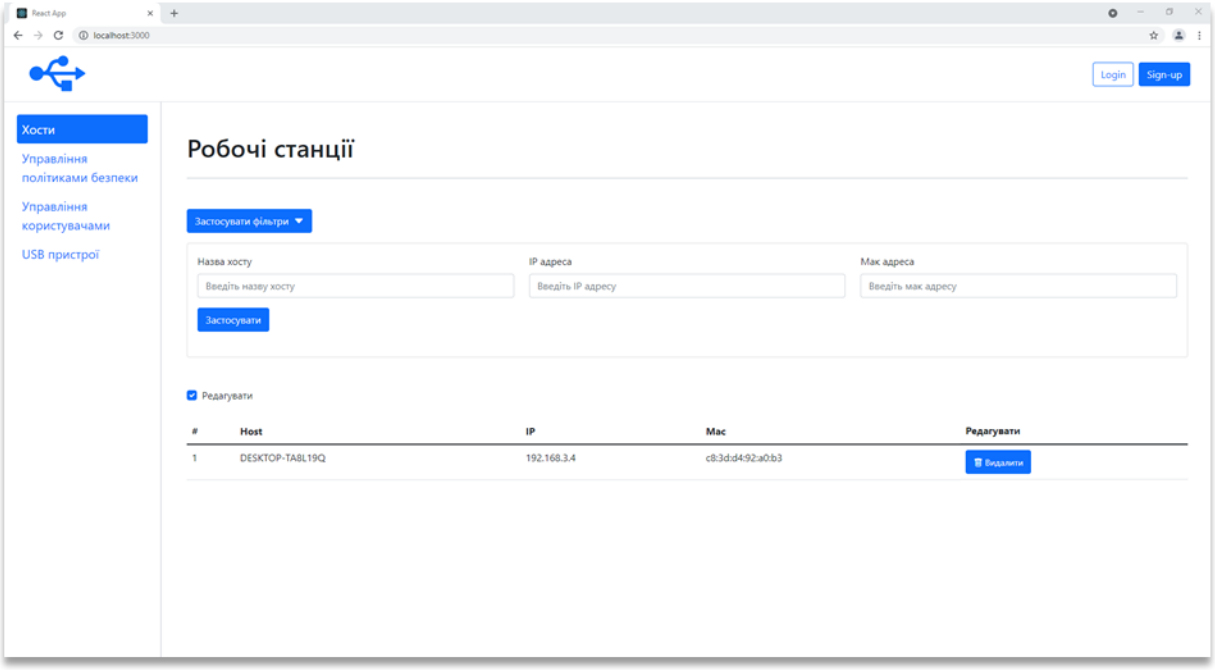

Рисунок 4.1 – Головна сторінка.

На сторінці, що зображена на рисунку вище, користувач може переглянути список робочих станцій, на яких встановленні клієнтські служби. На цій сторінці було реалізовано можливість фільтрувати дані за допомогою панелі для застосування фільтрів, також є можливість видаляти непотрібні станції. При натисканні на рядок в таблиці, користувач потрапить на сторінку, де відображатиметься список подій, які сталися на відповідній робочій станції. Дана сторінка зображена на рисунку 4.2.

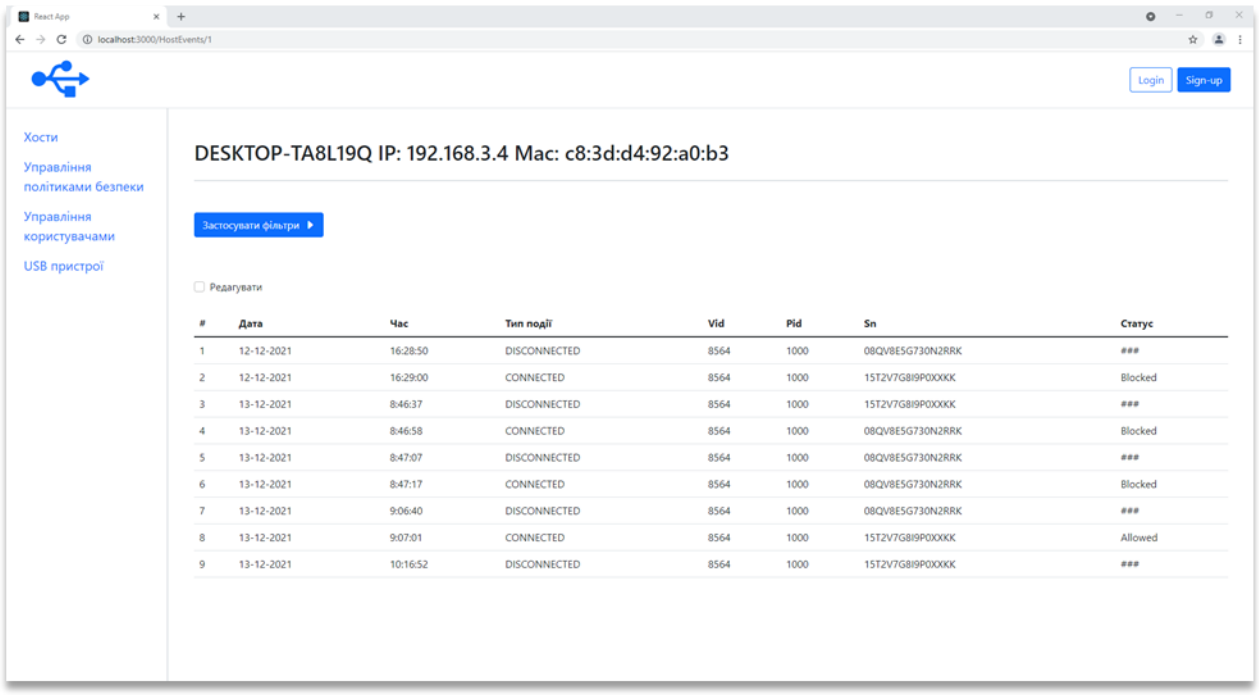

Рисунок 4.2 – Сторінка перегляду подій на робочій станції.

На цій сторінці користувач може отримати інформацію про дату та час, коли сталася подія, пристрій, який було під’єднано або від’єднано, та статус, що відображає інформацію про те, чи був даний пристрій заблокований або дозволений. Дана сторінка підтримує можливості фільтрування даних та видалення непотрібних записів, які зараз знаходяться в прихованому стані.

Далі було реалізовано сторінку, за допомогою якої користувач має змогу переглядати список USB-пристроїв, які коли небуть використовувалися в мережі або були додані до системи в ручну. Дану сторінку можна побачити на рисунку 4.3.

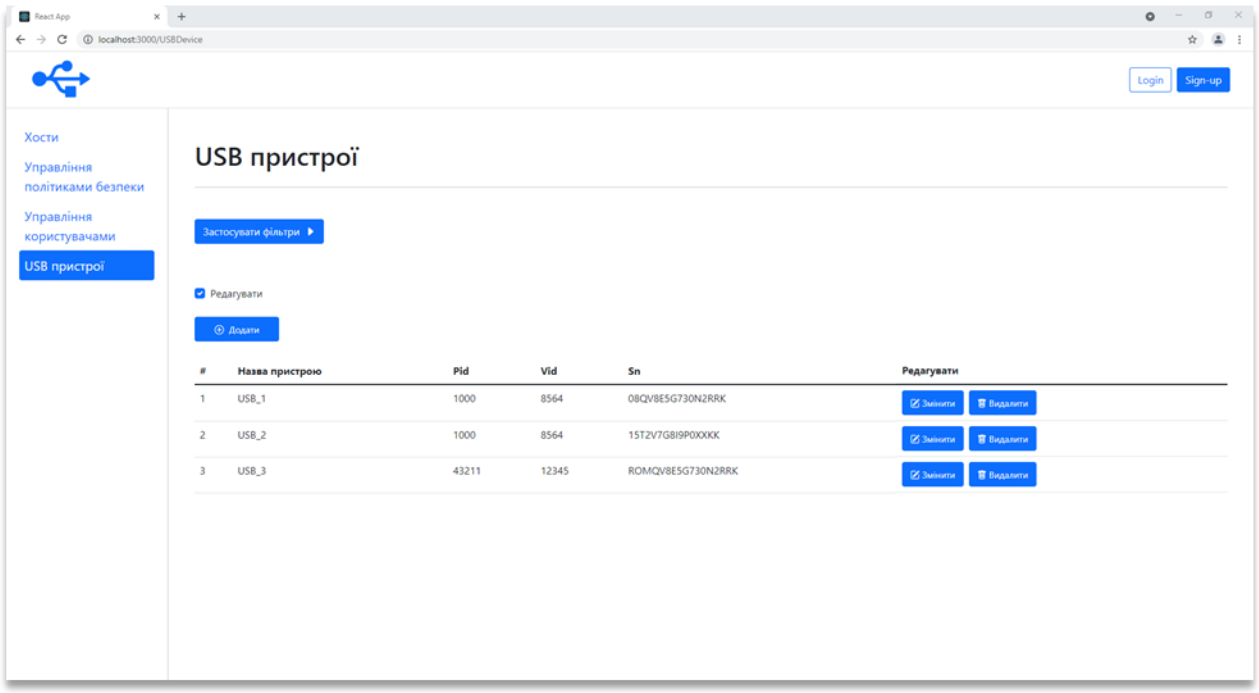

Рисунок 4.3 – Сторінка управління USB-пристроями.

На цій сторінці користувач може додавати, змінювати, видаляти та фільтрувати записи. Зміна або створення нового запису відбувається за допомогою модального вікна, яке можна побачити на рисунку 4.4.

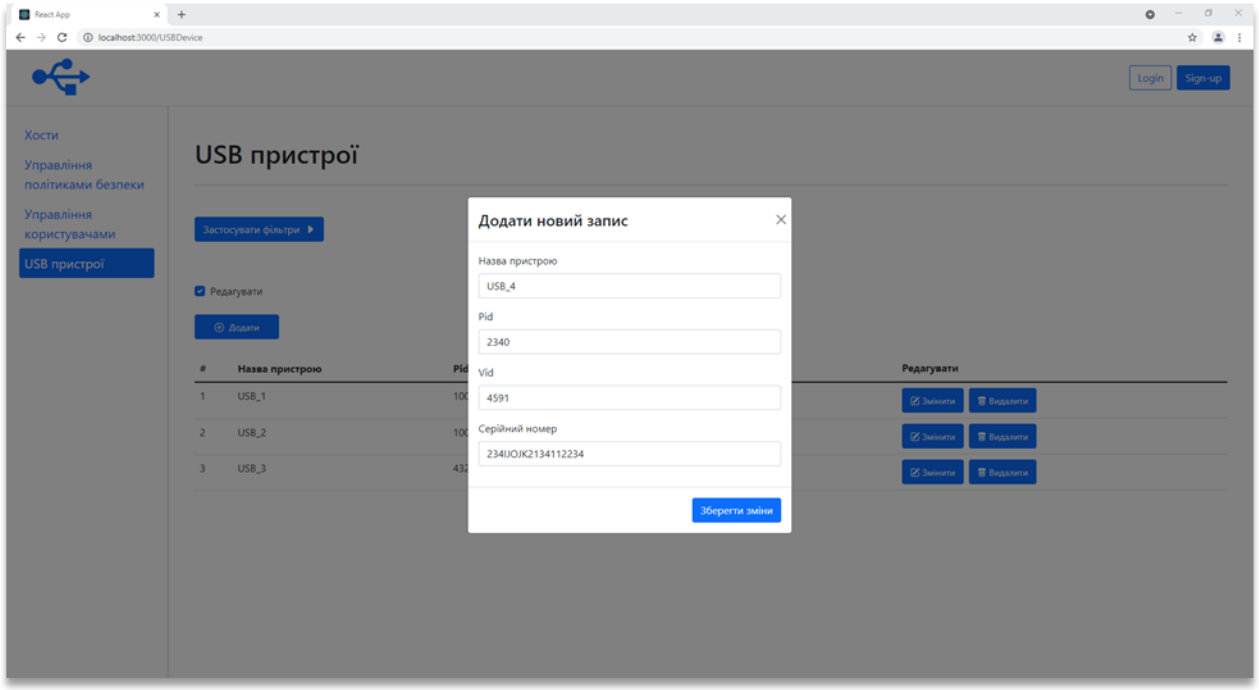

Рисунок 4.4 – Модальне вікно для редагування або добавлення USB-пристроїв.

Після того, як була додана можливість додавати USB-пристрої, було реалізовано сторінку для управління політиками безпеки, яку можна побачити на рисунку 4.5.

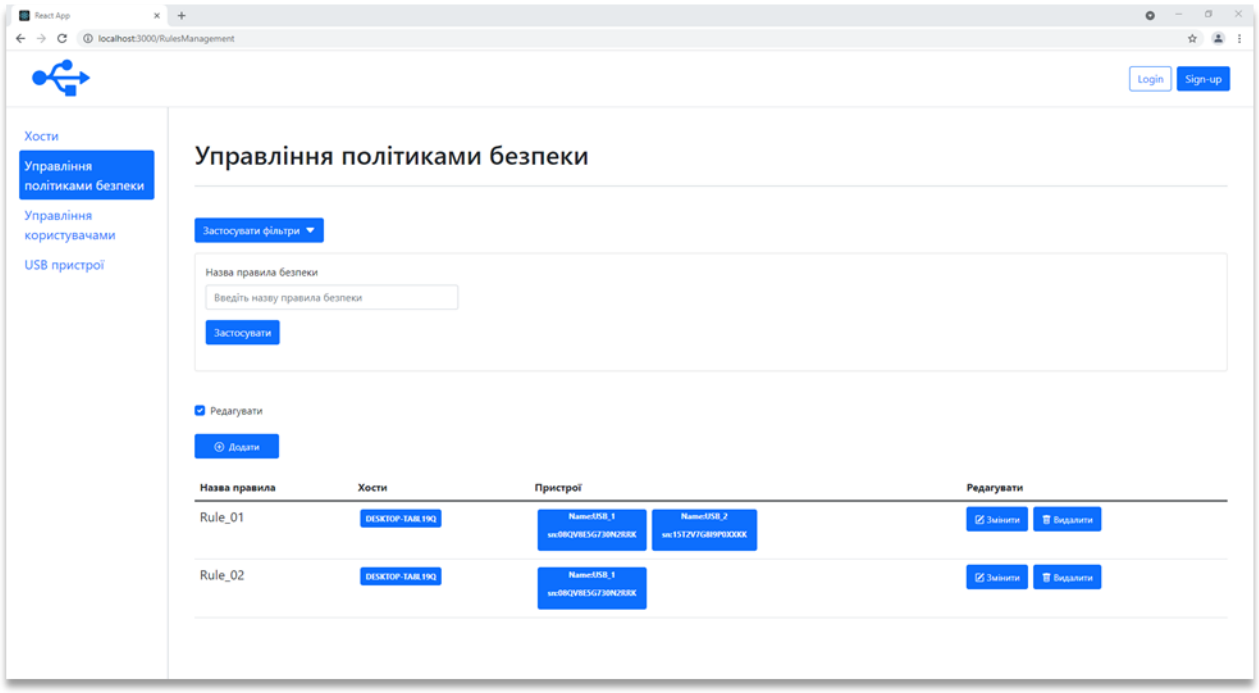

Рисунок 4.5 – Сторінка управління політиками безпеки.

На сторінці для управління політиками безпеки користувач може додавати, змінювати, видаляти та фільтрувати білі списки. Білий список складається з імені та набору USB-пристроїв та робочих станцій. На рисунку 4.6 можна побачити модальне вікно для створення білих списків.

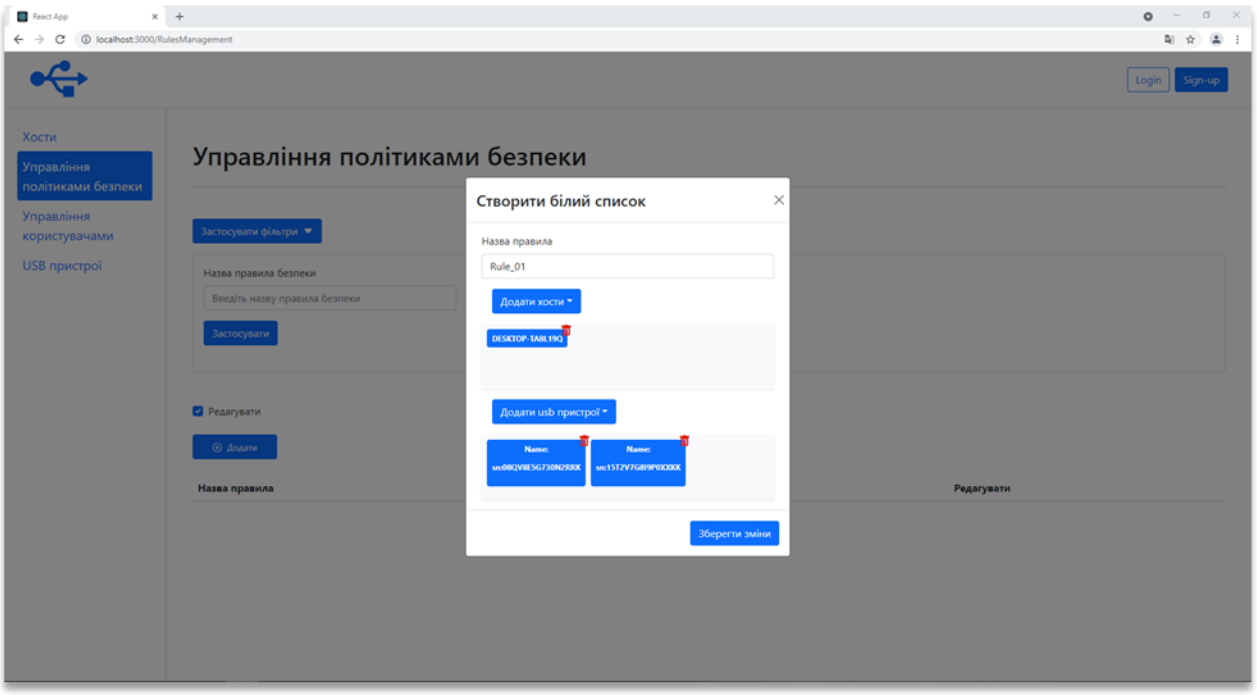

Рисунок 4.7 – Модальне вікно для редагування або створення білих списків.

Для редагування було використано такий самий компонент, звідси модальне вікно буде мати ідентичний вигляд.

Фінальним етапом є реалізація сторінки інтерфейсу користувача, яку можна побачити на рисунку 4.6. Де користувач має змогу додавати, редагувати інформацію про користувачів, видаляти непотрібні записи та здійснювати фільтрацію.

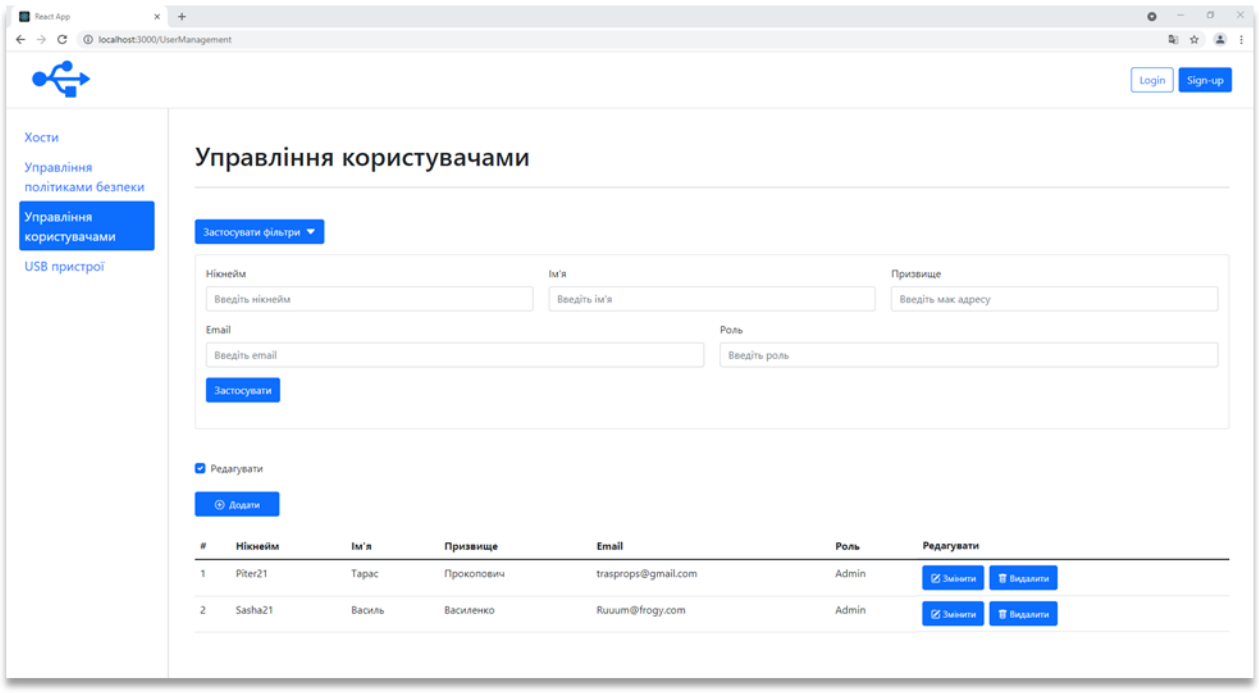

Рисунок 4.6 – Сторінка управління користувачами.

На рисунку 4.7 можна побачити модальне вікно, яке має ідентичний вигляд виглядає для редагування та створення нових користувачів.

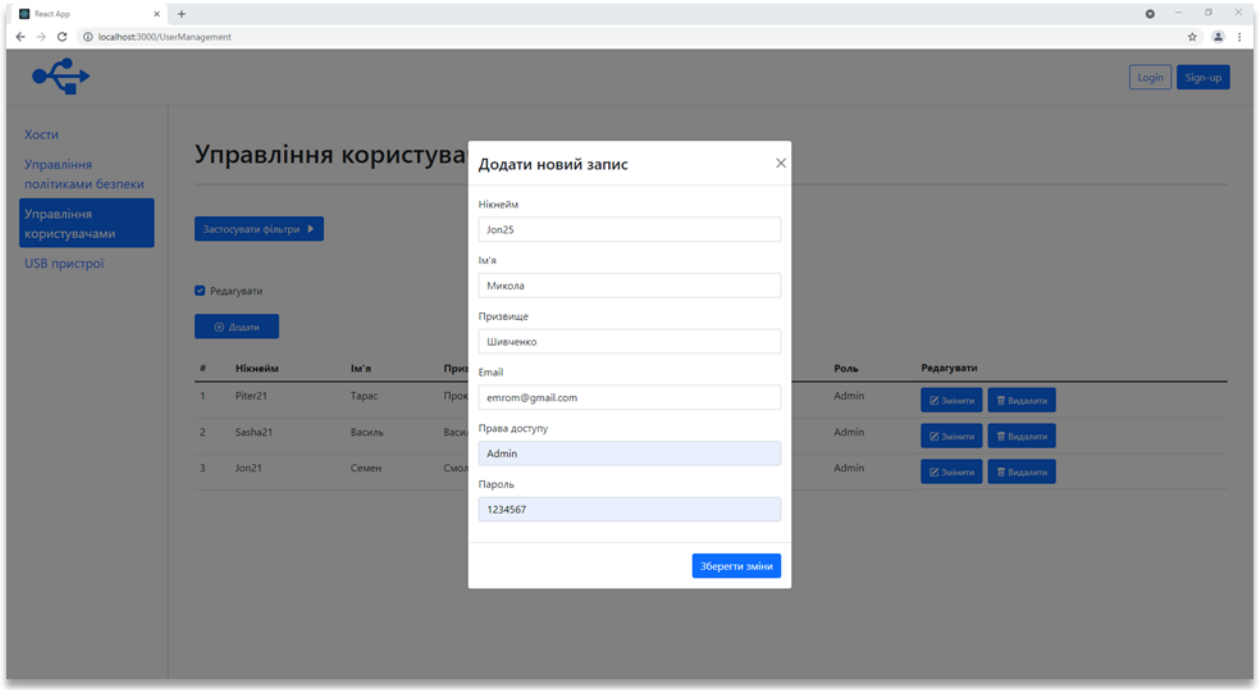

Рисунок 4.7 – Модальне вікно для редагування або створення записів про користувачів.

ВИСНОВОК

У цій роботі була розроблена інформаційна технологія забезпечення централізованого контролю за використанням USB-накопичувачів в корпоративній мережі. Створений продукт може використовуватися невеликими компаніями, які хочуть підвищити рівень безпеки в корпоративній мережі.

Під час роботи було проведено аналіз предметної області, який складався з огляду літератури та аналізу наявних рішень. Досліджено наявні способи забезпечення мережевої взаємодії, методи побудови веб-додатків та способи взаємодії з операційною системою Windows. Проведено аналіз засобів програмної реалізації, які дозволять розробити технологію централізованого контролю використання USB-накопичувачів в корпоративній мережі. Після аналізу предметної області та засобів реалізації було проведено проектування та моделювання технології під час, якої була розроблена діаграма варіантів використання, створений прототип веб-інтерфейсу та спроектована структура бази даних. На завершальному етапі було проведено програмну реалізацію. Результатом роботи є розроблена технологія, яка забезпечує можливість:

- задавати списки дозволених пристроїв на основі яких буде прийматися рішення;
- переглядати список підключених USB-накопичувачів, які використовуються в корпоративній мережі;
- блокувати USB-накопичувачі, що використовуються на комп’ютері;
- відслідковувати історію підключень носіїв інформації;
- централізованого управляти безпекою, за допомогою інтуїтивно зрозумілого веб-застосунку;
- розгортання на власних серверах;

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tatarinov V. et al. System for monitoring the connection of USB devices for cybersecurity auditing // CEUR Workshop Proceedings. CEUR-WS, 2020. Vol. 2654. P. 773–785.
2. Nissim N., Yahalom R., Elovici Y. USB-based attacks // Computers and Security. Elsevier Ltd, 2017. Vol. 70. P. 675–688.
3. Anderson B., Anderson B. Seven Deadliest USB Attacks // Seven Deadliest USB Attacks. Elsevier Inc., 2010.
4. Oliveira J., Pinto P., Santos H. Distributed Architecture to Enhance Systems Protection against Unauthorized Activity via USB Devices // Journal of Sensor and Actuator Networks 2021, Vol. 10, Page 19. Multidisciplinary Digital Publishing Institute, 2021. Vol. 10, № 1. P. 19.
5. Al-Dhief F.T. et al. Performance comparison between TCP and udp protocols in different simulation scenarios // International Journal of Engineering and Technology(UAE). Science Publishing Corporation Inc, 2018. Vol. 7, № 4.36 Special Issue 36. P. 172–176.
6. Zeng D. et al. Developing an Interactive Web-Based Programming Platform for Learning Computer Networking Protocols // Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST. Springer, Cham, 2020. Vol. 369. P. 611–625.
7. Gavrilă V., Băjenaru L., Dobre C. Modern single page application architecture: A case study // Studies in Informatics and Control. National Institute for R and D in Informatics, 2019. Vol. 28, № 2. P. 231–238.

8. Stępniak W., Nowak Z. Performance analysis of SPA web systems // Advances in Intelligent Systems and Computing. Springer Verlag, 2017. Vol. 521. P. 235–247.

9. Carter B. HTML Architecture, a Novel Development System (HANDS): An Approach for WEB Development // Proceedings - 2014 Annual Global Online Conference on Information and Computer Technology, GOCICT 2014. Institute of Electrical and Electronics Engineers Inc., 2014. P. 90–95.

10. Single-page App vs. Multi-page App: Pros, Cons, and Which is Better? [Electronic resource]. URL: https://lvivity.com/single-page-app-vs-multi-page-app.

11. Single-Page Application vs Multiple-Page Application: Which One To Choose For Your Project - Andrej Gajdos [Electronic resource]. URL: https://andrejgajdos.com/single-page-application-vs-multiple-page-application/.

12. Lissoir A. Leveraging WMI Scripting: Using Windows Management Instrumentation to Solve Windows Management Problems. 1st ed. Digital Press, 2003.

13. Windows Management Instrumentation (WMI) - Matthew M. Lavy, Ashley J. Meggitt - Google Books [Electronic resource]. URL: https://books.google.com.ua/books?hl=en&lr=&id=DD1jA3RgFEMC&oi=fnd&pg=PA1&dq=wmi&ots=FQyK6KPhJS&sig=YNbtFAZQaVu7xb46pW-ncnJvPqs&redir_esc=y#v=onepage&q=wmi&f=false.

14. What Is Windows Management Instrumentation (WMI)? Definition From SearchWindowsServer [Electronic resource]. URL: https://www.techtarget.com/searchwindowsserver/definition/Windows-Management-Instrumentation.

15. Best Device Control Software (Free and Paid) for 2021 [Electronic resource]. URL: https://www.softwaretestingmaterial.com/device-control-software/?utm_source=rss&utm_medium=rss&utm_campaign=device-control-software.

16. 25 Best Device Control Software in 2021 | Get Free Demo [Electronic resource]. URL: https://www.softwaresuggest.com/us/device-control-software.

17. 6 best USB blocker software for Windows 10 [Electronic resource]. URL: https://windowsreport.com/block-usb-ports-software/.

18. Edition F. C # 9 and . NET 5 – Modern Cross-Platform Development. 2020.

19. React – A JavaScript library for building user interfaces [Electronic resource]. URL: https://reactjs.org/.

20. Chęć D., Nowak Z. The performance analysis of web applications based on virtual DOM and reactive user interfaces // Advances in Intelligent Systems and Computing. Springer Verlag, 2019. Vol. 830. P. 119–134.

21. Introduction to ASP.NET Core | Microsoft Docs [Electronic resource]. URL: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0.

22. Japikse P., Grossnicklaus K., Dewey B. Build the RESTful Service with ASP.NET Core // Building Web Applications with .NET Core 2.1 and JavaScript. Apress, Berkeley, CA, 2020. P. 223–269.

23. ASP.NET Core in Action, Second Edition - Andrew Lock - Google Books [Electronic resource]. URL: https://books.google.com.ua/books?hl=en&lr=&id=FzgzEAAAQBAJ&oi=fnd&pg=PT16&dq=asp.net+core&ots=qJFEUWfYnN&sig=CqOYlBOKcOLYGieYpYc7bUmcJyc&redir_esc=y#v=onepage&q=asp.net%20core&f=false.

24. Schwichtenberg H. Modern Data Access with Entity Framework Core // Modern Data Access with Entity Framework Core. Apress, 2018.

Додаток А

Мережева бібліотека

```
namespace NetworkController
{
    public class ConnectionManager
    {
        private Dictionary<string, HostConnection> _hosts = new Dictionary<string,
HostConnection>();

        public void AddHostConnection(HostConnection hostConnection)
        {
            AddHostConnectionById(hostConnection.HostId, hostConnection);
        }

        public void AddHostConnectionById(string hostId, HostConnection
hostConnection)
        {
            if (!_hosts.ContainsKey(hostId))
            {
                _hosts.Add(hostId, hostConnection);
            }
        }

        public HostConnection GetHostConnection(string hostId)
        {
            return _hosts[hostId];
        }

        public void RemoveHostConnection(string hostId)
        {
            _hosts.Remove(hostId);
        }

    }
}


namespace NetworkController
{
    public class Dispat
    {
        private Dictionary<string, ICommandHandler> _controllers
          = new Dictionary<string, ICommandHandler>();

        public Dispat AddCommandHandler(string command, ICommandHandler controller)
        {
            if (!_controllers.ContainsKey(command))
            {
                _controllers.Add(command, controller);
            }
            else
            {
                _controllers[command] = controller;
            }

            return this;
        }
```

```
        public ICommandHandler GetCommandHandler(string key)
        {
            return _controllers[key];
        }
    }

}

namespace NetworkController
{
    public class FrontController
    {
        private readonly ILogger _loggeLocalExceptionsInService;
        private readonly Dispat _dispat;
        private readonly HostInteractor _hostInteractor;
        private readonly ChannelWriter<MessageNetPackage> _messageNetPackageWriter;

        public FrontController(Dispat dispat,
            HostInteractor hostInteractor,
            ChannelWriter<MessageNetPackage> messageNetPackageWriter)
        {
            _dispat = dispat;
            _hostInteractor = hostInteractor;
            _messageNetPackageWriter = messageNetPackageWriter;
        }

        public FrontController(Dispat dispat,
            HostInteractor hostInteractor,
            ChannelWriter<MessageNetPackage> messageNetPackageWriter,
            ILogger logger)
            : this(dispat, hostInteractor, messageNetPackageWriter)
        {
            _loggeLocalExceptionsInService = logger;
        }

        public void OnMessageNetPackageReceivedEvent(object sender,
MessageNetPackageRecivedEventArgs eventArgs)
        {
            Task.Run(async () =>
            {
                try
                {
                    var messageNetPackage =
JsonConvert.DeserializeObject<MessageNetPackage>(eventArgs.MessageNetPackage,
                        new JsonSerializerSettings() { Converters = { new
MessageNetPackageConverter() } });

                    var requestMessageNetPackage  = messageNetPackage as
RequestMessageNetPackage;
                    var responseMessageNetPackage = messageNetPackage as
ResponseMessageNetPackage;

                    if (requestMessageNetPackage != null)
                    {
                        ICommandHandler commandHandler = _dispat
                            .GetCommandHandler(requestMessageNetPackage.Command);

                        await commandHandler
                             .ExecuteCommand(requestMessageNetPackage,
_hostInteractor);
                    }

                    if(responseMessageNetPackage != null)
                    {
```

```
                        await
_messageNetPackageWriter.WriteAsync(responseMessageNetPackage);
                    }
                }
                catch (Exception err)
                {
                    _loggeLocalExceptionsInService?.LogError(err.MessageNetPackage);
                }
            });
        }
    }
}


namespace NetworkController
{
    public class HostConnection
    {
        public string HostId { get; set; }
        public string HostType { get; set; }
        public TcpClient TcpConnection { get => _hostConnection; }
        public NetworkStream NetStream { get => _networkMessageNetPackageSource; }

        private TcpClient _hostConnection;
        private NetworkStream _networkMessageNetPackageSource;
        private readonly ILogger _loggeLocalExceptionsInService;
        private ConnectionManager _connectionManager;
        private readonly FrontController _frontController;
        private readonly MessageNetPackageReceiver _messageNetPackageReceiver;
        public HostConnection(TcpClient hostConnection)
        {
            _hostConnection = hostConnection;
        }

        public HostConnection(TcpClient hostConnection, NetworkStream networkStream,
            ConnectionManager connectionManager, FrontController frontController)
        {
            _hostConnection    = hostConnection;
            _connectionManager = connectionManager;
            _frontController   = frontController;
            _networkMessageNetPackageSource     = networkStream;
            _messageNetPackageReceiver   = new
MessageNetPackageReceiver(networkStream, _loggeLocalExceptionsInService);
        }

        public HostConnection(TcpClient hostConnection, NetworkStream networkStream,
            ConnectionManager connectionManager, FrontController frontController,
ILogger logger)
            : this(hostConnection, networkStream, connectionManager,
frontController)
        {
            _loggeLocalExceptionsInService = logger;
        }

        public async Task<bool> InitHostConnectionAsync()
        {
            MessageNetPackageReceiver messageNetPackageReceiver = new
MessageNetPackageReceiver(_networkMessageNetPackageSource);
            RequestMessageNetPackage initMessageNetPackage = await
messageNetPackageReceiver.GetConnectionInitMessageNetPackageAsync();

            if(initMessageNetPackage.Command != INIT_CONNECTION_COMMAND)
            {
                _loggeLocalExceptionsInService?.LogInformation("Wrong
messageNetPackage type");
```

```
                return false;
            }

            HostId   = initMessageNetPackage.HostId;
            HostType = initMessageNetPackage.HostType;
            _connectionManager.AddHostConnection(this);

            return true;
        }

        public void Start(Stop_prosses_token_now stop_prosses_token_now)
        {
            Task.Run(async ()=> {
                _messageNetPackageReceiver.MessageNetPackageReceivedEvent +=
_frontController.OnMessageNetPackageReceivedEvent;
                await _messageNetPackageReceiver.StartAsync(stop_prosses_token_now);
            });
        }

        public void Stop()
        {
            _messageNetPackageReceiver?.Stop();
            CloseConnection();
        }

        public void CloseConnection()
        {
            _hostConnection?.Close();
            _networkMessageNetPackageSource?.Close();
        }
    }
}


namespace NetworkController
{
    public class HostInteractor
    {
        private readonly NetworkStream _networkMessageNetPackageSource;
        private readonly ConnectionManager _connectionManager;
        private readonly ChannelReader<MessageNetPackage> _messageNetPackageReader;

        public HostInteractor(NetworkStream networkStream,
ChannelReader<MessageNetPackage> messageNetPackageReader)
        {
            _networkMessageNetPackageSource = networkStream;
            _messageNetPackageReader = messageNetPackageReader;
        }

        public HostInteractor(ConnectionManager connectionManager,
ChannelReader<MessageNetPackage> messageNetPackageReader)
        {
            _connectionManager = connectionManager;
            _messageNetPackageReader     = messageNetPackageReader;
        }

        private MessageNetPackageSender GetMessageNetPackageSender(string hostId =
CLIENT_HOST_ID_MODE)
        {
            if (hostId == CLIENT_HOST_ID_MODE && _networkMessageNetPackageSource !=
null)
            {
                return new MessageNetPackageSender(_networkMessageNetPackageSource);
            }
            else if (_connectionManager != null)
```

```
            {
                NetworkStream networkStream = _connectionManager.
                                              GetHostConnection(hostId)?.
                                              NetStream;

                return (networkStream != null) ? new
MessageNetPackageSender(networkStream) : null;
            }

            return null;
        }

        public async Task SendAsync(MessageNetPackage messageNetPackage, string
hostId = CLIENT_HOST_ID_MODE)
        {
            var messageNetPackageSender = GetMessageNetPackageSender(hostId);
            await messageNetPackageSender.SendAsync(messageNetPackage);
        }

        public async Task<ResponseMessageNetPackage>
SendRequestAsync(RequestMessageNetPackage requestMessageNetPackage,
            string hostId = CLIENT_HOST_ID_MODE)
        {
            var messageNetPackageSender = GetMessageNetPackageSender(hostId);
            await messageNetPackageSender.SendAsync(requestMessageNetPackage);
            if(await _messageNetPackageReader.WaitToReadAsync())
                return await _messageNetPackageReader.ReadAsync() as
ResponseMessageNetPackage;
            return null;
        }

        public async Task SendResponseAsync(ResponseMessageNetPackage
responseMessageNetPackage,
            string hostId = CLIENT_HOST_ID_MODE)
        {
            var messageNetPackageSender = GetMessageNetPackageSender(hostId);
            await messageNetPackageSender.SendAsync(responseMessageNetPackage);
        }
    }
}


namespace NetworkController
{
    class MessageNetPackageReceiver
    {
        private NetworkStream _networkMessageNetPackageSource;
        private readonly ILogger _loggeLocalExceptionsInService;
        public event EventHandler ConnectionBrockenEvent;
        public event EventHandler<MessageNetPackageRecivedEventArgs>
MessageNetPackageReceivedEvent;
        private Stop_prosses_token_nowSource _stopTokenSource = new
Stop_prosses_token_nowSource();
        private Stop_prosses_token_now _stopToken;

        public MessageNetPackageReceiver(NetworkStream networkStream)
        {
            _networkMessageNetPackageSource = networkStream;
            _stopToken = _stopTokenSource.Token;
        }

        public MessageNetPackageReceiver(NetworkStream networkStream, ILogger
logger)
            : this(networkStream)
        {
```

```
            _loggeLocalExceptionsInService = logger;
        }

        public async Task StartAsync(Stop_prosses_token_now stop_prosses_token_now)
        {

            ThrowExceptionIfNull(_networkMessageNetPackageSource, new
NullStreamException("Stream is null!"));

            while (!stop_prosses_token_now.IsCancellationRequested &&
!_stopToken.IsCancellationRequested)
            {
                try
                {
                    string messageNetPackage = await
ReadMessageNetPackageFromStreamAsync(_networkMessageNetPackageSource);
                    MessageNetPackageReceivedEvent?.Invoke(this, new
MessageNetPackageRecivedEventArgs(messageNetPackage));
                }
                catch (Exception e)
                {
                    if ((e is IOException)
                      || e is ConnectionBrokenException)
                    {
                        ConnectionBrockenEvent?.Invoke(this, new EventArgs());
                    }
                    else
                    {
                        _loggeLocalExceptionsInService.LogInformation($"Error:
{e.MessageNetPackage}");
                    }
                }
            }
        }

        public async Task<RequestMessageNetPackage>
GetConnectionInitMessageNetPackageAsync()
        {
            ThrowExceptionIfNull(_networkMessageNetPackageSource, new
NullStreamException("Stream is null!"));
            string messageNetPackage = await
ReadMessageNetPackageFromStreamAsync(_networkMessageNetPackageSource);
            return
JsonConvert.DeserializeObject<RequestMessageNetPackage>(messageNetPackage);
        }

        private async Task<string>
ReadMessageNetPackageFromStreamAsync(NetworkStream stream)
        {
            byte[] data = new byte[256];
            StringBuilder builder = new StringBuilder();
            int bytes = 0;
            do
            {
                bytes = await stream.ReadAsync(data, 0, data.Length);
                if (bytes == 0) throw new ConnectionBrokenException("Connection is
broken!");
                builder.Append(Encoding.Unicode.GetString(data, 0, bytes));
            }
            while (stream.DataAvailable);

            return builder.ToString();
        }

        public void Stop()
```

```
        {
            _stopTokenSource.Cancel();
        }
    }
}


namespace NetworkController
{
    public class MessageNetPackageSender
    {
        private readonly ILogger _loggeLocalExceptionsInService;
        private NetworkStream _networkMessageNetPackageSource;

        public MessageNetPackageSender(NetworkStream networkStream)
        {
            _networkMessageNetPackageSource = networkStream;
        }

        public MessageNetPackageSender(NetworkStream networkStream, ILogger logger)
            : this(networkStream)
        {
            _loggeLocalExceptionsInService = logger;
        }

        public async Task SendAsync(MessageNetPackage messageNetPackage) {

            ThrowExceptionIfNull(_networkMessageNetPackageSource, new
NullStreamException("Stream is null!"));

            try
            {
                string messageNetPackageJson =
JsonConvert.SerializeObject(messageNetPackage);
                byte[] data = Encoding.Unicode.GetBytes(messageNetPackageJson);
                await _networkMessageNetPackageSource.WriteAsync(data, 0,
data.Length);
            }
            catch (Exception e)
            {
                _loggeLocalExceptionsInService.LogError(e.MessageNetPackage);
            }
        }
    }
}


namespace NetworkController.CustomEventArgs
{
    public class MessageNetPackageRecivedEventArgs : EventArgs
    {
        public MessageNetPackageRecivedEventArgs(string messageNetPackage)
        {
            MessageNetPackage = messageNetPackage;
        }

        public string MessageNetPackage { get; set; }
    }
}


namespace NetworkController.CustomExceptions
{
    class ConnectionBrokenException : Exception
    {
```

```
        public ConnectionBrokenException(){}
        public ConnectionBrokenException(string messageNetPackage) :
base(messageNetPackage){}
    }
}


namespace NetworkController
{
    static class ErrorHandlingUtility
    {
        public static void ThrowExceptionIfNull<T> (object obj, T exception) where T
: Exception
        {
            if (obj == null) throw exception;
        }
    }
}


namespace NetworkController.CustomExceptions
{
    class NullStreamException : Exception
    {
        public NullStreamException() { }
        public NullStreamException(string messageNetPackage) :
base(messageNetPackage) { }
    }
}


namespace NetworkController.CustomExceptions
{
    class NullTcpClientException : Exception
    {
        public NullTcpClientException() { }
        public NullTcpClientException(string messageNetPackage) :
base(messageNetPackage) { }
    }
}



namespace NetworkController
{
    public interface ICommandHandler
    {
        public Task ExecuteCommand(RequestMessageNetPackage messageNetPackage,
HostInteractor interactor);
    }
}


namespace NetworkController
{
    public abstract class MessageNetPackage
    {
        public string HostId { get; set; }
        public string HostType { get; set; }
        public Dictionary<string, object> Parameters { get; set; } = new
Dictionary<string, object>();

        public object this[string key]
        {
            get
```

```
            {
                return Parameters[key];
            }
            set {
                Parameters[key] = value;
            }
        }

    }
}


namespace NetworkController
{
    public class RequestMessageNetPackage : MessageNetPackage
    {
        public string Command { get; set; }
        public virtual RequestMessageNetPackage addParameter(string key, object
value)
        {
            Parameters.Add(key, value);
            return this;
        }

        public virtual RequestMessageNetPackage setMessageNetPackageHeader(string
hostId, string hostType, string command)
        {
            HostId = hostId;
            HostType = hostType;
            Command = command;
            return this;
        }
    }
}

namespace NetworkController
{
    public class ResponseMessageNetPackage : MessageNetPackage
    {
        public int StatusCode { get; set; }
        public ResponseMessageNetPackage addParameter(string key, object value)
        {
            Parameters.Add(key, value);
            return this;
        }

        public ResponseMessageNetPackage setStatusCode(int code)
        {
            this.StatusCode = code;
            return this;
        }

        public ResponseMessageNetPackage setMessageNetPackageHeader(string hostId,
string hostType)
        {
            HostId = hostId;
            HostType = hostType;
            return this;
        }
    }
}


namespace NetworkController
{
```

```
    public class NetConnector
    {
        private readonly int _port;
        private readonly ILogger _loggeLocalExceptionsInService;
        private TcpClient _serverConnector;
        private NetworkStream _networkMessageNetPackageSource;
        private HostInteractor _hostInteractor;
        private readonly int _attemptTime;
        private readonly string _ip;
        private MessageNetPackageReceiver _messageNetPackageReceiver;
        private FrontController _frontController;
        private Dispat _dispat;
        private readonly Channel<MessageNetPackage> _messageNetPackageChannel;
        public delegate MessageNetPackage ConnectionToServerEventHandler();
        public event ConnectionToServerEventHandler ConnectionToServerEvent;

        public NetConnector(string ip, int port, int attemptTime, Dispat dispat)
        {
            _ip = ip;
            _port = port;
            _attemptTime = attemptTime;
            _dispat  = dispat;
            _messageNetPackageChannel =
Channel.CreateUnbounded<MessageNetPackage>();
        }

        public NetConnector(string ip, int port, int attemptTime, Dispat dispat,
ILogger logger)
            : this(ip, port, attemptTime, dispat)
        {
            _loggeLocalExceptionsInService = logger;
        }

        public void Start(Stop_prosses_token_now stop_prosses_token_now)
        {
           Task.Run(async () => {
               await ConnectAsync(stop_prosses_token_now);
               _messageNetPackageReceiver = new
MessageNetPackageReceiver(_networkMessageNetPackageSource,
_loggeLocalExceptionsInService);
               _messageNetPackageReceiver.MessageNetPackageReceivedEvent   +=
_frontController.OnMessageNetPackageReceivedEvent;
               _messageNetPackageReceiver.ConnectionBrockenEvent += (object sender,
EventArgs e)
                                                                    =>
Reconnect(stop_prosses_token_now);
               await _messageNetPackageReceiver.StartAsync(stop_prosses_token_now);
           });
        }

        private void Reconnect(Stop_prosses_token_now stop_prosses_token_now)
        {
            var Connect = ConnectAsync(stop_prosses_token_now);
            Task.WaitAll(Connect);
        }

        private async Task ConnectAsync(Stop_prosses_token_now
stop_prosses_token_now)
        {
           await Task.Run(async () =>
           {
               while (!stop_prosses_token_now.IsCancellationRequested)
               {
                   try
                   {
```

```
                        _serverConnector = new TcpClient();
                        _serverConnector.Connect(_ip, _port);
                        _networkMessageNetPackageSource   =
_serverConnector.GetStream();

                        _hostInteractor  = new
HostInteractor(_networkMessageNetPackageSource,
                            _messageNetPackageChannel.Reader);

                        _frontController = new FrontController(_dispat,
                            _hostInteractor,
                            _messageNetPackageChannel.Writer,
                            _loggeLocalExceptionsInService);

                         var messageNetPackage = ConnectionToServerEvent?.Invoke();
                         await _hostInteractor.SendAsync(messageNetPackage);
                         _loggeLocalExceptionsInService?.LogInformation("The
connection is established");
                         break;
                    }
                    catch(Exception)
                    {
                        _loggeLocalExceptionsInService?.LogInformation("The
connection is not established");
                        Thread.Sleep(_attemptTime);
                    }
                }
            });
        }

        public HostInteractor GetHostInteractor()
        {
            return _hostInteractor;
        }

        public void Stop()
        {
            _serverConnector?.Close();
        }
    }
}


namespace NetworkController
{
    public class NetListener
    {
        private TcpListener _serverListener;
        private readonly int _port;
        private readonly ILogger _loggeLocalExceptionsInService;
        private readonly ConnectionManager _connectionManager;
        private readonly FrontController   _frontController;
        private readonly Channel<MessageNetPackage>  _messageNetPackageChannel;
        private HostInteractor _hostInteractor;

        public NetListener(int port, Dispat dispat)
        {
            _port = port;
            _messageNetPackageChannel    =
Channel.CreateUnbounded<MessageNetPackage>();
            _connectionManager = new ConnectionManager();
            _hostInteractor    = new HostInteractor(_connectionManager,
_messageNetPackageChannel.Reader);
            _frontController   = new FrontController(dispat, _hostInteractor,
_messageNetPackageChannel.Writer);
```

```
        }

        public NetListener(int port, Dispat dispat, ILogger logger)
            : this(port, dispat)
        {
            _loggeLocalExceptionsInService = logger;
        }

        public void Start(Stop_prosses_token_now stop_prosses_token_now)
        {
            Task.Run( async () => {

                try
                {
                    _serverListener = TcpListener.Create(_port);
                    _serverListener.Start();
                    await StartListeningAsync(stop_prosses_token_now);
                }
                catch (SocketException e)
                {
                    _loggeLocalExceptionsInService?.LogError($"SocketException:
{e.MessageNetPackage}");
                }
                catch (Exception e)
                {
                    _loggeLocalExceptionsInService?.LogError($"Exception:
{e.MessageNetPackage}");
                }
                finally
                {
                    _serverListener?.Stop();
                }

            });
        }

        private async Task StartListeningAsync(Stop_prosses_token_now
stop_prosses_token_now)
        {
            while (!stop_prosses_token_now.IsCancellationRequested)
            {
                try
                {
                    TcpClient connection = await
_serverListener.AcceptTcpClientAsync();
                    NetworkStream networkStream = connection.GetStream();

                    HostConnection hostConnection = new HostConnection(connection,
networkStream,
                        _connectionManager, _frontController,
_loggeLocalExceptionsInService);

                    if(await hostConnection.InitHostConnectionAsync())
                    {
                        hostConnection.Start(stop_prosses_token_now);
                    }

                }
                catch (Exception e)
                {
                    _loggeLocalExceptionsInService?.LogError($"Exception:
{e.MessageNetPackage}");
                }
            }
```

```
            Stop();
        }

        public HostInteractor GetHostInteractor()
        {
            return _hostInteractor;
        }

        public void Stop()
        {
            _serverListener?.Stop();
        }
    }
}
```

Клієнтська служба

```
namespace ClientService
{
    public class Worker : BackgroundService
    {
        private readonly ILogger<Worker> _loggeLocalExceptionsInService;
        private readonly Dispat _dispat;
        private readonly NetConnector _netConnector;
        private readonly USBEventWatcher _usbEventWatcher;
        private readonly USBEventConsumer _usbEventConsumer;
        private readonly string _hostIdentifier;
        private readonly int _port;
        private readonly string _ip;
        private readonly int _attemptTime;

        public Worker(ILogger<Worker> logger)
        {
            var appSettings = ConfigurationManager.AppSettings;

            _port = Int32.Parse(appSettings["port"]);
            _ip   = appSettings["ip"];
            _attemptTime = Int32.Parse(appSettings["attemptTime"]);

            _hostIdentifier = GetUniqueIdentifier();
            _loggeLocalExceptionsInService = logger;
            _dispat = new Dispat();
            _netConnector = new NetConnector(_ip, _port, _attemptTime, _dispat,
_loggeLocalExceptionsInService);
            _netConnector.ConnectionToServerEvent += OnConnectionToServerEvent;

            _usbEventConsumer = new USBEventConsumer(_netConnector, _hostIdentifier,
_loggeLocalExceptionsInService);
            _usbEventWatcher  = new USBEventWatcher(_usbEventConsumer);
        }

        protected override Task ExecuteAsync(Stop_prosses_token_now
stop_prosses_token_now)
        {
            _netConnector?.Start(stop_prosses_token_now);
            _usbEventWatcher?.Start();
            return Task.CompletedTask;
        }

        public override async Task StopAsync(Stop_prosses_token_now
stop_prosses_token_now)
        {
            _netConnector?.Stop();
            await base.StopAsync(stop_prosses_token_now);
        }

        public static RequestMessageNetPackage OnConnectionToServerEvent()
        {
            var initConnectionMessageNetPackage = new RequestMessageNetPackage();
            initConnectionMessageNetPackage
                .setMessageNetPackageHeader(
                    GetUniqueIdentifier(),
                    HostType.AGENT,
                    INIT_CONNECTION_COMMAND);

            return initConnectionMessageNetPackage;
        }
```

```
    }
}

namespace ClientService
{
#pragma warning disable CA1416
    class USBEventWatcher
    {
        private readonly ManagementEventWatcher _eventWatcher;

        public USBEventWatcher(USBEventConsumer usbEventConsumer)
        {
            var GetUSBObjQuery = new WqlEventQuery(GET_USB_OBJ_QUERY);
            _eventWatcher = new ManagementEventWatcher(GetUSBObjQuery);
            _eventWatcher.EventArrived += usbEventConsumer.OnUSBConnectionEvent;
        }

        public void Start()
        {
            _eventWatcher?.Start();
        }

        public void Stop()
        {
            _eventWatcher?.Stop();
        }
    }
#pragma warning restore CA1416
}

namespace ClientService
{
#pragma warning disable CA1416
    class USBEventConsumer
    {
        private readonly ILogger<Worker> _loggeLocalExceptionsInService;
        private readonly NetConnector _netConnector;
        private readonly USBConnectionLogger _usbConnectionLogger;
        private readonly USBDeviceChecker _usbDeviceChecker;
        private readonly string _hostIdentifier;

        public USBEventConsumer(NetConnector netConnector, string hostIdentifier,
ILogger<Worker> logger)
        {
            _netConnector = netConnector;
            _loggeLocalExceptionsInService = logger;
            _hostIdentifier = hostIdentifier;
            _usbConnectionLogger = new USBConnectionLogger(_netConnector,
_hostIdentifier);
            _usbDeviceChecker = new USBDeviceChecker(_netConnector,
_hostIdentifier);
        }

        public async void OnUSBConnectionEvent(object sender, EventArrivedEventArgs
eventArgs)
        {
            try
            {
                var usbEventData = GetUsbConnectionEventData(eventArgs);

                if (usbEventData != null)
                {
                    bool isAllowed = false;
                    if (usbEventData.ActionType == USB_DEVICE_CONNECTED)
                    {
```

```
                        isAllowed = await
_usbDeviceChecker.CheckUSBDevice(usbEventData);
                        if (isAllowed)
                        {
                            EnableUSBDevice(usbEventData.USBDeviceId);
                        }
                        else
                        {
                            BlockUSBDevice(usbEventData.USBDeviceId);
                        }
                    }
                    usbEventData.IsAllowed = isAllowed;
                    await
_usbConnectionLogger.SaveUSBConnectionEventData(usbEventData);
                }
            }
            catch (Exception ex)
            {
                _loggeLocalExceptionsInService.LogError(ex.MessageNetPackage);
            }
        }

        private USBEventData GetUsbConnectionEventData(EventArrivedEventArgs
eventArgs)
        {
            var USBTargetDevice = (ManagementBaseObject)eventArgs.NewEvent
                                    .Properties["TargetInstance"].Value;
            var usbAEventType =
eventArgs.NewEvent.SystemProperties["__Class"].Value.ToString();

            if (usbAEventType != INSTANCE_DELETION_EVENT && usbAEventType !=
INSTANCE_CREATION_EVENT)
                return null;

            return GetUSBEventData(USBTargetDevice, usbAEventType);
        }
    }
#pragma warning restore CA1416
}

namespace ClientService
{
#pragma warning disable CA1416
    static class SystemUtility
    {
        public static string GetUniqueIdentifier()
        {
            return new DeviceIdBuilder()
                .AddMachineName()
                .AddOSVersion()
                .AddProcessorId()
                .AddSystemUUID()
                .AddUserName()
                .ToString();
        }

        public static IPAddress GetIPAddress()
        {
            using(Socket socket = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, 0))
            {
                socket.Connect("8.8.8.8", 65530);
                IPEndPoint IpendPoint = socket.LocalEndPoint as IPEndPoint;
                return IpendPoint.Address;
            }
```

```
        }

        [DllImport("iphlpapi.dll", ExactSpelling = true)]
        public static extern int SendARP(int DeIP, int SoIP, [Out] byte[] pcMACAddr,
ref int pPhyAddrLen);
        public static string GetMacAddress()
        {
            var ip = GetIPAddress();
            byte[] mac = new byte[6];
            int macLen = mac.Length;

            if (SendARP(BitConverter.ToInt32(ip.GetAddressBytes(), 0), 0, mac, ref
macLen) != 0)
                throw new InvalidOperationException("SendARP failed.");

            return GetMacString(mac);
        }

        private static string GetMacString(byte[] mac)
        {
            List<string> macStr = new List<string>();

            foreach (byte macByte in mac)
                macStr.Add(macByte.ToString("x2"));

            return string.Join(':', macStr);
        }

        public static string GetNameOfHost()
        {
            var globalPCProp = IPGlobalProperties.GetIPGlobalProperties();
            return globalPCProp.HostName;
        }

        public static USBEventData GetUSBEventData(ManagementBaseObject target,
string usbAEventType)
        {
            string usbDeviceID = (string)target["DeviceID"];
            Regex USBInfoRegex = new Regex(USB_INFO_PATTERN);
            MatchCollection USBDevice =
USBInfoRegex.Matches(Convert.ToString(target["PNPDeviceID"]));
            string prefix = (usbAEventType == INSTANCE_DELETION_EVENT) ? "DIS" : "";
            string usbAction = prefix + "CONNECTED";

            return new USBEventData(usbDeviceID, USBDevice[0].Value,
USBDevice[1].Value, USBDevice[2].Value, usbAction);
        }

        static List<ManagementBaseObject> GetDevicesList()
        {
            var devices = new List<ManagementBaseObject>();

            using (var usbDeviceSearcherOnLocalPC = new
ManagementObjectUsbDeviceSearcherOnLocalPC(CIMV2, SELECT_DEVICES))
            {
                foreach (var device in usbDeviceSearcherOnLocalPC.Get())
                {
                    devices.Add(device);
                }
            }
            return devices;
        }

        public static void BlockUSBDevice(string targetDeviceID)
        {
```

```
            EnableOrBlock(targetDeviceID, false);
        }

        public static void EnableUSBDevice(string targetDeviceID)
        {
            EnableOrBlock(targetDeviceID, true);
        }

        private static void EnableOrBlock(string targetDeviceID, bool flage)
        {
            var devices = GetDevicesList();
            foreach (var device in devices)
            {
                var currDeviceID = (string)device["DeviceID"];
                var currDeviceGuid = (string)device["ClassGuid"];
                if (targetDeviceID == currDeviceID)
                {
                    USBBlocler.SetDeviceEnabled(new Guid(currDeviceGuid),
currDeviceID, flage);
                }
            }
        }
    }
#pragma warning restore CA1416
}



namespace ClientService.Schemas
{
    class USBEventData
    {
        public string VID { get; set; }
        public string PID { get; set; }
        public string SN { get; set; }
        public string USBDeviceId { get; set; }
        public string ActionType { get; set; }
        public bool? IsAllowed { get; set; } = null;

        public USBEventData(string usbDeviceId, string vid, string pid, string sn,
string actionType)
        {
            USBDeviceId = usbDeviceId;
            VID = vid;
            PID = pid;
            SN = sn;
            ActionType = actionType;
        }
    }
}

namespace ClientService.OnEvent
{
    class USBDeviceChecker
    {
        private readonly NetConnector _netConnector;
        private readonly string _hostIdentifier;

        public USBDeviceChecker(NetConnector netConnector, string hostInteractor)
        {
            _hostIdentifier = hostInteractor;
            _netConnector = netConnector;
        }

        public async Task<bool> CheckUSBDevice(USBEventData usbEventData)
```

```
        {
            if (usbEventData == null)
                throw new ArgumentNullException(nameof(usbEventData));

            var serverInteractor = _netConnector.GetHostInteractor();

            var USBEventMessageNetPackage = new RequestMessageNetPackage()
                .setMessageNetPackageHeader(
                    _hostIdentifier,
                    HostType.AGENT,
                    CHECK_USB_DEVICE)
                .addParameter("HostName", GetNameOfHost())
                .addParameter("Vid", usbEventData.VID)
                .addParameter("Pid", usbEventData.PID)
                .addParameter("Sn", usbEventData.SN);

            var responseMessageNetPackage = await serverInteractor
                .SendRequestAsync(USBEventMessageNetPackage);

            return (bool)responseMessageNetPackage["isAllowed"];
        }
    }
}

namespace ClientService.OnEvent
{
#pragma warning disable CA1416
    class USBConnectionLogger
    {
        private readonly NetConnector _netConnector;
        private readonly string _hostIdentifier;

        public USBConnectionLogger(NetConnector netConnector,
            string hostIdentifier)
        {
            _hostIdentifier = hostIdentifier;
            _netConnector   = netConnector;
        }

        public async Task SaveUSBConnectionEventData(USBEventData usbEventData)
        {
            if(usbEventData == null)
                throw new ArgumentNullException(nameof(usbEventData));

            var serverInteractor = _netConnector.GetHostInteractor();

            string isAllowed = "###";
            if(usbEventData.ActionType == USB_DEVICE_CONNECTED)
            {
                isAllowed = (bool) usbEventData.IsAllowed ? "Allowed" : "Blocked";
            }

            var USBEventMessageNetPackage = new RequestMessageNetPackage()
                .setMessageNetPackageHeader(
                    _hostIdentifier,
                    HostType.AGENT,
                    SAVE_HOST_EVENT_COMMAND)
                .addParameter("HostName", GetNameOfHost())
                .addParameter("IPAdress", GetIPAddress().ToString())
                .addParameter("MacAdress", GetMacAddress())
                .addParameter("Vid", usbEventData.VID)
                .addParameter("Pid", usbEventData.PID)
                .addParameter("Sn", usbEventData.SN)
                .addParameter("IsAllowed", isAllowed)
                .addParameter("Png", usbEventData.USBDeviceId)
```

```
                .addParameter("EventType", usbEventData.ActionType);

                var responseMessageNetPackage
                        = await
serverInteractor.SendRequestAsync(USBEventMessageNetPackage);

        }
    }
#pragma warning restore CA1416
}
```

Модель бази даних

```
namespace DB.Models
{
    public class Host
    {
        public int Id { get; set; }
        [Required]
        public string HostIdentifier { get; set; }
        [Required]
        public string HostName { get; set; }
        [Required]
        public string MacAdress { get; set; }
        [Required]
        public string IPAdress { get; set; }
        public List<HostEvent> HostEvents { get; set; } = new List<HostEvent>();
        public List<Whitelist> Whitelists { get; set; } = new List<Whitelist>();
    }
}
```

```
namespace DB.Models
{
    public class HostEvent
    {
        public int Id { get; set; }
        [Required]
        [Column(TypeName = "Date")]
        public DateTime EventDate { get; set; }
        [Required]
        public TimeSpan EventTime { get; set; }
        public string EventType { get; set; }
        public int USBDeviceId { get; set; }
        public string IsAllowed { get; set; }
        public USBDevice USBDevice { get; set; }
        public int HostId { get; set; }
        public Host Host { get; set; }
    }
}
```

```
namespace DB.Models
{
    public class USBDevice
    {
        public int Id { get; set; }
        public string Name { get; set; }
        [Required]
        public string Vid { get; set; }
        [Required]
        public string Pid { get; set; }
        [Required]
        public string Sn { get; set; }
        public string Png { get; set; }
        public List<HostEvent> HostEvents { get; set; } = new List<HostEvent>();
        public List<Whitelist> Whitelists { get; set; }  = new List<Whitelist>();
    }
}
```

```
namespace DB.Models
{
    public class Whitelist
    {
        public int Id { get; set; }
        [Required]
        public string RuleName { get; set; }
        public List<USBDevice> USBDevices { get; set; } = new List<USBDevice>();
        public List<Host> Hosts { get; set; } = new List<Host>();
    }
}


namespace DB
{
    public class AppDBContext : DbContext
    {

        private readonly IConfiguration _config;

        public AppDBContext(DbContextOptions<AppDBContext> options, IConfiguration
config)
            : base(options)
        {
            _config = config;
        }

        public DbSet<AppUser> AppUsers { get; set; }
        public DbSet<Host> Hosts { get; set; }
        public DbSet<HostEvent> HostEvents { get; set; }
        public DbSet<Whitelist> Whitelists { get; set; }
        public DbSet<USBDevice> USBDevices { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {

            modelBuilder.Entity<AppUser>()
                .HasIndex(user => new { user.Email, user.UserName })
                .IsUnique();

            modelBuilder.Entity<Host>()
                .HasIndex(host => host.HostIdentifier)
                .IsUnique();

            base.OnModelCreating(modelBuilder);
        }
    }
}
```

WEB API сервер

```
namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class HostController : ControllerBase
    {

        private readonly AppDBContext _DBContext;

        public HostController(AppDBContext DBContext)
        {
            _DBContext = DBContext;
        }

        [HttpGet]
        public async Task<JsonResult> GetHosts()
        {
            var hosts = await _DBContext.Hosts
                .Select(host => new
                {
                    Id = host.Id,
                    HostIdentifier = host.HostIdentifier,
                    HostName = host.HostName,
                    MacAdress = host.MacAdress,
                    IPAdress = host.IPAdress
                })
                .ToListAsync();

            return new JsonResult(hosts);
        }

        [HttpPost("SearchCertainHosts")]
        public async Task<JsonResult> SearchCertainHosts([FromBody] HostData
hostSearchData)
        {
            var (_, HostName, MacAdress, IPAdress) = hostSearchData;

            var hosts = await _DBContext.Hosts
                .Where(host =>
                       (host.HostName  == HostName  ||
String.IsNullOrEmpty(HostName))
                    && (host.MacAdress == MacAdress ||
String.IsNullOrEmpty(MacAdress))
                    && (host.IPAdress  == IPAdress  ||
String.IsNullOrEmpty(IPAdress))
                    && (!String.IsNullOrEmpty(HostName)
                    || !String.IsNullOrEmpty(MacAdress)
                    || !String.IsNullOrEmpty(IPAdress)))
                .Select(host => new
                {
                    Id    = host.Id,
                    HostIdentifier = host.HostIdentifier,
                    HostName  = host.HostName,
                    MacAdress = host.MacAdress,
                    IPAdress  = host.IPAdress
                })
                .ToListAsync();

            return new JsonResult(hosts);
        }
```

```
        [HttpDelete("DeleteHost/{id}")]
        public async Task<IActionResult> DeleteHost(int id)
        {
            var host = await _DBContext.Hosts
                            .Where(host   => host.Id == id)
                            .Include(host => host.HostEvents)
                            .FirstOrDefaultAsync();

            if (host == null)
            {
                return NotFound("Host not found");
            }

            _DBContext.Hosts.Remove(host);
            await _DBContext.SaveChangesAsync();

            return Ok();
        }
    }
}



namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class HostEventController : ControllerBase
    {

        private readonly AppDBContext _DBContext;

        public HostEventController(AppDBContext DBContext)
        {
            _DBContext = DBContext;
        }

        [HttpPost("HostEventsSearch")]
        public async Task<JsonResult> SearchCertainHostEvents([FromBody]
HostEventData hostEventSearchData)
        {
            var (HostId, Vid, Pid, Sn, EventDate, EventTime) = hostEventSearchData;

            var HostEvents = await _DBContext.Hosts
                .Where(host => host.Id == HostId)
                .Include(host => host.HostEvents)
                .ThenInclude(hostEvents => hostEvents.USBDevice)
                .Select(host => new
                {
                    HostName = host.HostName,
                    MacAdress = host.MacAdress,
                    IPAdress = host.IPAdress,
                    HostEvents = host.HostEvents
                        .Where(he =>
                            (he.USBDevice.Vid == Vid || String.IsNullOrEmpty(Vid))
                            && (he.USBDevice.Pid == Pid ||
String.IsNullOrEmpty(Pid))
                            && (he.USBDevice.Sn  == Sn  || String.IsNullOrEmpty(Sn))
                            && (he.EventDate == EventDate || EventDate == null)
                            && (he.EventTime == EventTime || EventTime == null)
                            && (!String.IsNullOrEmpty(Vid)
                                || !String.IsNullOrEmpty(Pid)
                                || !String.IsNullOrEmpty(Sn)
```

```
                                || EventDate != null
                                || EventTime != null))
                        .Select(he => new
                        {
                            Id  = he.Id,
                            Vid = he.USBDevice.Vid,
                            Pid = he.USBDevice.Pid,
                            Sn = he.USBDevice.Sn,
                            IsAllowed = he.IsAllowed,
                            EventType = he.EventType,
                            EventTime = he.EventTime,
                            EventData = he.EventDate
                        })
                })
                .ToListAsync();

            return new JsonResult(HostEvents);
        }

        [HttpGet("{id}")]
        public async Task<JsonResult> GetAllHostEvents(int id)
        {
            var hostEvents = await _DBContext.Hosts
                .Where(host => host.Id == id)
                .Include(host => host.HostEvents)
                .ThenInclude(hostEvent => hostEvent.USBDevice)
                .Select(host => new {
                    HostName   = host.HostName,
                    MacAdress  = host.MacAdress,
                    IPAdress   = host.IPAdress,
                    HostEvents = host.HostEvents
                    .Select(he => new
                    {
                        Id = he.Id,
                        Vid = he.USBDevice.Vid,
                        Pid = he.USBDevice.Pid,
                        Sn  = he.USBDevice.Sn,
                        IsAllowed = he.IsAllowed,
                        EventType = he.EventType,
                        EventTime = he.EventTime,
                        EventData = he.EventDate
                    })
                })
                .FirstOrDefaultAsync();

            return new JsonResult(hostEvents);
        }

        [HttpDelete("DeleteUsbEvent/{id}")]
        public async Task<IActionResult> DeleteUsbEvent(int id)
        {
            var hostEvent = await _DBContext.HostEvents
                  .FirstOrDefaultAsync(hostEvent => hostEvent.Id == id);

            if (hostEvent == null)
            {
                return NotFound("Usb event not found");
            }

            _DBContext.HostEvents.Remove(hostEvent);
            await _DBContext.SaveChangesAsync();
            return Ok();
        }
    }
}
```

```
namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class LoginController : ControllerBase
    {
        private readonly IConfiguration _config;
        private readonly AppDBContext _DBContext;
        private readonly string _hashSalt;

        public LoginController(IConfiguration config, AppDBContext DBContext) {

            _config = config;
            _DBContext = DBContext;
            _hashSalt  = _config["HashSalt"];
        }

        [HttpPost("/login")]
        public async Task<IActionResult> Login(LoginData loginData)
        {

            var user = await GetUserAsync(loginData);

            if (user == null)
            {
                return NotFound("User not found");
            }

            var jwtToken = getJwtToken(user);

            return Ok(jwtToken);
        }

        private async Task<AppUser> GetUserAsync(LoginData loginData)
        {
            string currEmail = loginData.Email;
            string currUserName = loginData.UserName;
            string currHashedPassword = BCryptoLib.HashPassword(loginData.Password,
_hashSalt);

            return await _DBContext.AppUsers
                                   .Where(user => (user.Email == currEmail ||
user.UserName == currUserName)
                                          && user.Password == currHashedPassword)
                                   .FirstOrDefaultAsync();
        }

        private string getJwtToken(AppUser user)
        {
            var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
            var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);

            var claims = new[]
            {
                new Claim(ClaimTypes.NameIdentifier, user.UserName),
                new Claim(ClaimTypes.GivenName, user.Name),
                new Claim(ClaimTypes.Surname, user.Surname),
                new Claim(ClaimTypes.Email, user.Email),
                new Claim(ClaimTypes.Role, user.Role),
            };
```

```
            var jwtToken = new JwtSecurityToken(
                _config["Jwt:Issuer"],
                _config["Jwt:Audience"],
                claims,
                expires: DateTime.Now.AddDays(1),
                signingCredentials: credentials);

            return new JwtSecurityTokenHandler().WriteToken(jwtToken);
        }
    }
}


namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class USBDeviceController : ControllerBase
    {
        private readonly AppDBContext _DBContext;
        public USBDeviceController(AppDBContext DBContext)
        {
            _DBContext = DBContext;
        }

        [HttpGet]
        public async Task<JsonResult> GetAllUSBDevices()
        {
            var USBDevices = await _DBContext.USBDevices
                .Select( usbDevice => new
                {
                  Id = usbDevice.Id,
                  name = usbDevice.Name,
                  vid = usbDevice.Vid,
                  pid = usbDevice.Pid,
                  sn  = usbDevice.Sn
                })
                .ToListAsync();

            return new JsonResult(USBDevices);
        }

        [HttpPost("AddUSBDevice")]
        public async Task<ActionResult> AddUSBDevices([FromBody] USBDeviceData
usbDeviceData)
        {
            var (_, name, vid, pid, sn) = usbDeviceData;

            var usbDevice = new USBDevice(){
                Name = name,
                Pid = pid,
                Vid = vid,
                Sn = sn };

            await _DBContext.USBDevices.AddAsync(usbDevice);
            await _DBContext.SaveChangesAsync();

            return Ok();
        }

        [HttpPost("SearchCertainUSBDevice")]
        public async Task<JsonResult> SearchCertainUSBDevice(
            [FromBody] USBDeviceData usbDeviceSearchData)
        {
            var (usbDeviceId, name, vid, pid, sn) = usbDeviceSearchData;
```

```
    var USBDevices = await _DBContext.USBDevices
        .Where(usbDevice =>
            (usbDevice.Id == usbDeviceId || usbDeviceId == null)
            && (usbDevice.Name == name || String.IsNullOrEmpty(name))
            && (usbDevice.Vid == vid || String.IsNullOrEmpty(vid))
            && (usbDevice.Pid == pid || String.IsNullOrEmpty(pid))
            && (usbDevice.Sn == sn || String.IsNullOrEmpty(sn))
            && (usbDeviceId != null
                || !String.IsNullOrEmpty(name)
                || !String.IsNullOrEmpty(vid)
                || !String.IsNullOrEmpty(pid)
                || !String.IsNullOrEmpty(sn)))
        .Select(usbDevice => new
        {
            Id = usbDevice.Id,
            Name = usbDevice.Name,
            Vid = usbDevice.Vid,
            Pid = usbDevice.Pid,
            Sn = usbDevice.Sn
        })
        .ToListAsync();

    return new JsonResult(USBDevices);
}

[HttpPut("UpdateUSBDevice/{id}")]
public async Task<ActionResult> UpdateUSBDevice(
    int id,
    [FromBody] USBDeviceData usbDeviceUpdateData)
{
    var (_, name, vid, pid, sn) = usbDeviceUpdateData;

    var USBDevice = await _DBContext.USBDevices
        .FirstOrDefaultAsync(usbDevice => usbDevice.Id == id);

    if (USBDevice == null)
    {
        return NotFound("USB Device not found");
    }

    if(!String.IsNullOrEmpty(name))
        USBDevice.Name = name;

    if(!String.IsNullOrEmpty(vid))
        USBDevice.Vid = vid;

    if(!String.IsNullOrEmpty(pid))
        USBDevice.Pid = pid;

    if (!String.IsNullOrEmpty(sn))
        USBDevice.Sn = sn;

    await _DBContext.SaveChangesAsync();

    return Ok();
}

[HttpDelete("DeleteUSBDevice/{id}")]
public async Task<ActionResult> DeleteUSBDevice(int id)
{
    var USBDevice = await _DBContext.USBDevices
        .FirstOrDefaultAsync(usbDevice => usbDevice.Id == id);

    if (USBDevice == null)
```

```
            {
                return NotFound("USB Device not found");
            }

            _DBContext.USBDevices.Remove(USBDevice);
            await _DBContext.SaveChangesAsync();

            return Ok();
        }
    }
}


namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UserController : ControllerBase
    {

        private readonly IConfiguration _config;
        private readonly AppDBContext _DBContext;
        private readonly string _hashSalt;

        public UserController(IConfiguration config, AppDBContext DBContext)
        {
            _config = config;
            _DBContext = DBContext;
            _hashSalt = _config["HashSalt"];
        }

        [HttpGet]
        public async Task<JsonResult> GetAllUsers()
        {
            var allUsers = await _DBContext.AppUsers
                .Select(user => new
                {
                    Id = user.Id,
                    UserName = user.UserName,
                    Name = user.Name,
                    Surname = user.Surname,
                    Email = user.Email,
                    Role = user.Role
                }

                ).ToListAsync();

            return new JsonResult(allUsers);
        }

        [HttpPost("AddNewUser")]
        public async Task<ActionResult> AddNewUser([FromBody] UserData newUser)
        {
            var (_,
               userName,
               name,
               surname,
               email,
               password,
               role) = newUser;

            password = BCryptoLib.HashPassword(password, _hashSalt);
            var user = new AppUser() {

                    UserName = userName,
```

```
                    Name = name,
                    Surname = surname,
                    Email = email,
                    Password = password,
                    Role = role,
            };
            await _DBContext.AppUsers.AddAsync(user);
            await _DBContext.SaveChangesAsync();
            return Ok();
        }

        [HttpPut("UpdateUser/{id}")]
        public async Task<ActionResult> UpdateUser(int id, [FromBody] UserData
updatedUserData)
        {
            var user = await GetUserEntityById(id);

            var (_,
                updatedUserName,
                updatedName,
                updatedSurname,
                updatedEmail,
                updatedPassword,
                updatedRole) = updatedUserData;

            if (user == null)
            {
                return NotFound("User not found");
            }

            if(!String.IsNullOrEmpty(updatedUserName))
                user.UserName = updatedUserName;

            if(!String.IsNullOrEmpty(updatedName))
                user.Name = updatedName;

            if (!String.IsNullOrEmpty(updatedSurname))
                user.Surname = updatedSurname;

            if (!String.IsNullOrEmpty(updatedEmail))
                user.Email = updatedEmail;

            if (!String.IsNullOrEmpty(updatedRole))
                user.Role = updatedUserData.Role;

            if(!String.IsNullOrEmpty(updatedPassword))
                user.Password = BCryptoLib.HashPassword(updatedUserData.Password,
_hashSalt);

            await _DBContext.SaveChangesAsync();

            return Ok();

        }

        [HttpDelete("DeleteUser/{id}")]
        public async Task<ActionResult> DeleteUser(int id)
        {
            var user = await GetUserEntityById(id);

            if(user == null)
            {
                return NotFound("User not found");
            }
```

```
            _DBContext.AppUsers.Remove(user);
            await _DBContext.SaveChangesAsync();
            return Ok();
        }

        [HttpPost("SearchCertainUsers")]
        public async Task<JsonResult> SearchCertainUsers([FromBody] UserData
userSearchData)
        {

            var (_, UserName, Name, Surname, Email, _, Role) = userSearchData;

            var users = await _DBContext.AppUsers
                .Where(user =>
                    (user.UserName == UserName || String.IsNullOrEmpty(UserName))
                    && (user.Name == Name || String.IsNullOrEmpty(Name))
                    && (user.Surname == Surname || String.IsNullOrEmpty(Surname))
                    && (user.Email == Email || String.IsNullOrEmpty(Email))
                    && (user.Role == Role || String.IsNullOrEmpty(Role))
                    && (!String.IsNullOrEmpty(UserName)
                        || !String.IsNullOrEmpty(Name)
                        || !String.IsNullOrEmpty(Surname)
                        || !String.IsNullOrEmpty(Email)
                        || !String.IsNullOrEmpty(Role)))
                .Select(user => new
                {
                    Id = user.Id,
                    UserName = user.UserName,
                    Name = user.Name,
                    Surname = user.Surname,
                    Email = user.Email,
                    Role = user.Role
                })
                .ToListAsync();

            return new JsonResult(users);
        }

        private async Task<AppUser> GetUserEntityById(int id)
        {
            return await _DBContext.AppUsers
                        .Where(user => user.Id == id)
                        .FirstOrDefaultAsync();
        }
    }
}


namespace WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class WhitelistController : ControllerBase
    {
        private readonly AppDBContext _DBContext;

        public WhitelistController(AppDBContext DBContext)
        {
            _DBContext = DBContext;
        }

        [HttpGet]
        public async Task<JsonResult> GetAllWhitelists()
        {
            var whitelists = await _DBContext.Whitelists
```

```
                .Include(whitelist => whitelist.USBDevices)
                .Include(whitelist => whitelist.Hosts)
                .Select(whitelist => new
                {
                    Id = whitelist.Id,
                    RuleName = whitelist.RuleName,
                    hostsId = whitelist.Hosts.Select(host => host.Id),
                    usbDevicesId = whitelist.USBDevices.Select(usbDevice =>
usbDevice.Id)
                })
                .ToListAsync();

            return new JsonResult(whitelists);
        }

        [HttpGet("{id}")]
        public async Task<JsonResult> GetWhitelistById(int id)
        {
            var whitelists = await _DBContext.Whitelists
                .Where(whitelist => whitelist.Id == id)
                .Include(whitelist => whitelist.USBDevices)
                .Include(whitelist => whitelist.Hosts)
                .Select(whitelist => new
                {
                    Id = whitelist.Id,
                    RuleName = whitelist.RuleName,
                    hostsId = whitelist.Hosts.Select(host => host.Id),
                    usbDevicesId = whitelist.USBDevices.Select(usbDevice =>
usbDevice.Id)
                })
                .ToListAsync();

            return new JsonResult(whitelists);
        }

        [HttpPost("AddNewWhitelist")]
        public async Task<ActionResult> AddNewWhitelist([FromBody] WhitelistData
whitelistData)
        {
            var (ruleName, hostsId, usbDevicesId) = whitelistData;

            var whitelist = await _DBContext.Whitelists
                .FirstOrDefaultAsync(whitelist => whitelist.RuleName == ruleName);

            if(whitelist != null)
            {
                return BadRequest("Whitelist already exists!");
            }

            var hosts = await _DBContext.Hosts
                .Where(host => hostsId.Contains(host.Id)).ToListAsync();

            var usbDevices = await _DBContext.USBDevices
                .Where(usbDevice =>
usbDevicesId.Contains(usbDevice.Id)).ToListAsync();

            whitelist = new Whitelist();
            whitelist.RuleName = ruleName;
            whitelist.USBDevices.AddRange(usbDevices);
            whitelist.Hosts.AddRange(hosts);

            await _DBContext.AddAsync(whitelist);
            await _DBContext.SaveChangesAsync();

            return Ok();
```

```
        }

        [HttpDelete("DeleteWhitelist/{id}")]
        public async Task<ActionResult> DeleteWhitelist(int id)
        {
            var whitelist = await _DBContext.Whitelists
                .Include(whitelist => whitelist.USBDevices)
                .Include(whitelist => whitelist.Hosts)
                .FirstOrDefaultAsync(whitelist => whitelist.Id == id);


            if (whitelist == null)
            {
                return NotFound("Whitelist not found");
            }

            var hostsIdentifier = whitelist.Hosts
                .Select(host => host.HostIdentifier).ToList();

            var usbDevicesId = whitelist.USBDevices
                .Select(usbDevice => usbDevice.Png).ToList();

            _DBContext.Whitelists.Remove(whitelist);
            await _DBContext.SaveChangesAsync();

            return Ok();
        }

        [HttpPut("UpdateWhitelist/{id}")]
        public async Task<ActionResult> UpdateWhitelist(int id, [FromBody]
WhitelistData whitelistData)
        {
            var currWhitelist = await _DBContext.Whitelists
                .Include(whitelist => whitelist.USBDevices)
                .Include(whitelist => whitelist.Hosts)
                .FirstOrDefaultAsync(whitelist => whitelist.Id == id);

            var whitelist = await _DBContext.Whitelists
                .Where(whitelist => whitelist.Id == id)
                .Include(whitelist => whitelist.USBDevices)
                .Include(whitelist => whitelist.Hosts)
                .Select(whitelist => new
                {
                    Id = whitelist.Id,
                    RuleName = whitelist.RuleName,
                    hostsId = whitelist.Hosts.Select(host => host.Id),
                    usbDevicesId = whitelist.USBDevices.Select(usbDevice =>
usbDevice.Id)
                }).FirstOrDefaultAsync();

            if (whitelist == null)
            {
                return NotFound("Whitelist not found");
            }

            var hostsIdToDelete = whitelist.hostsId
                    .Except(whitelistData.HostsId).ToList();

            var usbDevicesIdToDelete = whitelist.usbDevicesId
                .Except(whitelistData.USBDevicesId).ToList();

            var hostsIdToAdd = whitelistData.HostsId
                .Except(whitelist.hostsId).ToList();

            var usbDevicesIdToAdd = whitelistData.USBDevicesId
```

```
                .Except(whitelist.usbDevicesId).ToList();


            var hostsToDelete = await _DBContext.Hosts
                .Where(host => hostsIdToDelete.Contains(host.Id)).ToListAsync();

            var usbDevicesToDelete = await _DBContext.USBDevices
                .Where(usbDevice =>
usbDevicesIdToDelete.Contains(usbDevice.Id)).ToListAsync();

            var hostsToAdd = await _DBContext.Hosts
                .Where(host => hostsIdToAdd.Contains(host.Id)).ToListAsync();

            var usbDevicesToAdd = await _DBContext.USBDevices
                .Where(usbDevice =>
usbDevicesIdToAdd.Contains(usbDevice.Id)).ToListAsync();

            if (currWhitelist.RuleName != whitelistData.RuleName)
                currWhitelist.RuleName = whitelistData.RuleName;
            await _DBContext.SaveChangesAsync();

            foreach (var host in hostsToDelete)
            {
                currWhitelist.Hosts.Remove(host);
                await _DBContext.SaveChangesAsync();
            }

            foreach(var usbDevice in usbDevicesToDelete)
            {
                currWhitelist.USBDevices.Remove(usbDevice);
                await _DBContext.SaveChangesAsync();
            }

            foreach(var usbDevice in usbDevicesToAdd)
            {
                currWhitelist.USBDevices.Add(usbDevice);
                await _DBContext.SaveChangesAsync();
            }

            foreach (var host in hostsToAdd)
            {
                currWhitelist.Hosts.Add(host);
                await _DBContext.SaveChangesAsync();
            }

            return Ok();
        }

    }
}

namespace WebAPI.Schemas
{
    public class AllowedUSBDevice
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Vid { get; set; }
        public string Pid { get; set; }
        public string Sn { get; set; }
        public bool isBlocked { get; set; }
    }
}

namespace WebAPI.Schemas
```

```
{
    public class HostData
    {
        public int Id { get; set; }
        public string HostName { get; set; }
        public string MacAdress { get; set; }
        public string IPAdress { get; set; }

        public void Deconstruct(out int id,
            out string hostName,
            out string macAdress,
            out string ipAdress)
        {
            id = Id;
            hostName = HostName;
            macAdress = MacAdress;
            ipAdress = IPAdress;
        }
    }
}

namespace WebAPI.Schemas
{
    public class LoginData
    {
        public string UserName { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }

        public void Deconstruct(out string userName,
            out string email,
            out string password)
        {
            userName = UserName;
            email = Email;
            password = Password;
        }
    }
}

namespace WebAPI.Schemas
{
    public class USBDeviceData
    {
        public int? Id { get; set; }
        public string Name { get; set; }
        public string Pid { get; set; }
        public string Vid { get; set; }
        public string Sn { get; set; }

        public void Deconstruct(
            out int? id,
            out string name,
            out string pid,
            out string vid,
            out string sn)
        {
            id   = Id;
            name = Name;
            pid = Pid;
            vid = Vid;
            sn = Sn;
        }
    }
}
```

```
namespace WebAPI.Schemas
{
    public class USBEventsData
    {
        public string Vid { get; set; }
        public string Pid { get; set; }
        public string Sn { get; set; }
        public DateTime EventDate { get; set; }
        public TimeSpan EventTime { get; set; }
    }
}

namespace WebAPI.Schemas
{
    public class UserData
    {
        public int Id { get; set;}
        public string UserName { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
        public string Role { get; set; }

        public void Deconstruct(
            out int id,
            out string userName,
            out string name,
            out string surname,
            out string email,
            out string password,
            out string role)
        {
            id = Id;
            userName = UserName;
            name = Name;
            surname = Surname;
            email = Email;
            password = Password;
            role = Role;
        }
    }
}

namespace WebAPI.Schemas
{
    public class WhitelistData
    {
        public string RuleName { get; set; }
        public List<int> HostsId { get; set; } = new List<int> ();
        public List<int> USBDevicesId { get; set; } = new List<int>();

        public void Deconstruct(out string ruleName,
            out List<int> hostsId,
            out List<int> usbDevicesId)
        {
            ruleName = RuleName;
            hostsId = new List<int>(HostsId);
            usbDevicesId = new List<int>(USBDevicesId);
        }
    }
}
```

```
namespace WebAPI.BackgroundServices
{
    public class CommunicatorBackgroundService : BackgroundService
    {
        private readonly ILogger<CommunicatorBackgroundService>
_loggeLocalExceptionsInService;
        private readonly IConfiguration _config;
        private NetListener _netListener;
        private Dispat  _dispat;
        private readonly int _port;
        private readonly RequestService  _request;
        private readonly ResponseService _response;
        private readonly IServiceProvider _serviceProvider;

        public CommunicatorBackgroundService(
           IConfiguration  config,
           RequestService  request,
           ResponseService response,
           ILogger<CommunicatorBackgroundService> logger,
           IServiceProvider serviceProvider
           )
        {
            _config = config;
            _request  = request;
            _response = response;
            _serviceProvider = serviceProvider;
            _loggeLocalExceptionsInService = logger;

            _port = Int32.Parse(_config["Port"]);

            _dispat = new Dispat()
                .AddCommandHandler(SAVE_HOST_EVENT_COMMAND,
                    new SaveHostEvent(_response, _loggeLocalExceptionsInService,
_serviceProvider))
                .AddCommandHandler(CHECK_USB_DEVICE,
                    new CheckUSBDevice(_response, _loggeLocalExceptionsInService,
_serviceProvider));

            _netListener = new NetListener(_port, _dispat, logger);
        }

        protected override async Task ExecuteAsync(Stop_prosses_token_now
stop_prosses_token_now)
        {
            _netListener.Start(stop_prosses_token_now);
            await ListenServerRequest(stop_prosses_token_now);
        }

        private async Task ListenServerRequest(Stop_prosses_token_now
stop_prosses_token_now)
        {
            while (await
_request.WaitMessageNetPackageAsync(stop_prosses_token_now))
            {
                var requestMessageNetPackage = (RequestMessageNetPackage) await
_request.ReadMessageNetPackageAsync();

                var hostInteractor = _netListener.GetHostInteractor();
                await hostInteractor.SendRequestAsync(requestMessageNetPackage,
                    (string)requestMessageNetPackage["HostIdentifier"]);
            }
        }

    }
}
```

```
namespace WebAPI.Commands
{
    public class CheckUSBDevice : ICommandHandler
    {
        private readonly ResponseService _response;
        private readonly IServiceProvider _serviceProvider;
        private ILogger _loggeLocalExceptionsInService;
        public CheckUSBDevice(ResponseService response,
            ILogger logger,
            IServiceProvider serviceProvider)
        {
            _serviceProvider = serviceProvider;
            _response = response;
            _loggeLocalExceptionsInService = logger;
        }

        public async Task ExecuteCommand(RequestMessageNetPackage messageNetPackage,
HostInteractor interactor)
        {
            ResponseMessageNetPackage responseMessageNetPackage = new
ResponseMessageNetPackage()
                .setMessageNetPackageHeader("webserver_id", HostType.WEB_SERVER);

            try
            {
                var HostIdentifier = messageNetPackage.HostId;
                var USBDeviceSn = (string) messageNetPackage["Sn"];

                using (var scope = _serviceProvider.CreateScope())
                {
                    var _dbContext =
scope.ServiceProvider.GetRequiredService<AppDBContext>();

                    var rule = await _dbContext.Whitelists
                        .Include(whitelist => whitelist.USBDevices)
                        .Include(whitelist => whitelist.Hosts)
                        .Where(whitelist =>
                            whitelist.Hosts.Select(host => host.HostIdentifier)
                                .Contains(HostIdentifier)
                            && whitelist.USBDevices.Select(usbDevice =>
usbDevice.Sn)
                                .Contains(USBDeviceSn))
                        .FirstOrDefaultAsync();

                    responseMessageNetPackage
                   .setStatusCode(DATA_SAVED_SUCCESS);

                    if (rule == null)
                    {
                        responseMessageNetPackage.addParameter("isAllowed", false);
                    }
                    else
                    {
                        responseMessageNetPackage.addParameter("isAllowed", true);
                    }

                    responseMessageNetPackage.addParameter("Text: ",
                        "The USB Device has been successfully checked.");
                }
            }
            catch(Exception ex)
            {
                responseMessageNetPackage
```

```
                    .setStatusCode(DATA_SAVED_ERROR)
                    .addParameter("Text: ", ex.MessageNetPackage);
                _loggeLocalExceptionsInService.LogError(ex.MessageNetPackage);
            }

            await interactor.SendResponseAsync(responseMessageNetPackage,
messageNetPackage.HostId);
        }
    }
}


namespace WebAPI.NetworkControllers
{
    public class ReceiveResponse : ICommandHandler
    {
        private readonly ResponseService _response;
        private ILogger _loggeLocalExceptionsInService;
        public ReceiveResponse(ResponseService response, ILogger logger)
        {
            _response = response;
            _loggeLocalExceptionsInService   = logger;
        }

        public async Task ExecuteCommand(RequestMessageNetPackage messageNetPackage,
HostInteractor interactor)
        {
            try
            {
                await _response.SendMessageNetPackageAsync(messageNetPackage);
            }
            catch(Exception ex)
            {
                _loggeLocalExceptionsInService.LogError(ex.MessageNetPackage);
            }
        }
    }
}


namespace WebAPI.CommandHandlers
{
    public class SaveHostEvent : ICommandHandler
    {
        private readonly ResponseService _response;
        private readonly ILogger _loggeLocalExceptionsInService;
        private readonly IServiceProvider _serviceProvider;

        public SaveHostEvent(ResponseService receiveResponse,
            ILogger logger,
            IServiceProvider serviceProvider)
        {
            _loggeLocalExceptionsInService = logger;
            _response = receiveResponse;
            _serviceProvider = serviceProvider;
        }

        public async Task ExecuteCommand(RequestMessageNetPackage messageNetPackage,
HostInteractor interactor)
        {
            ResponseMessageNetPackage responseMessageNetPackage = new
ResponseMessageNetPackage()
                .setMessageNetPackageHeader("webserver_id", HostType.WEB_SERVER);

            try
```

```
            {
                using (var scope = _serviceProvider.CreateScope())
                {
                    var _dbContext =
scope.ServiceProvider.GetRequiredService<AppDBContext>();

                    var host = _dbContext.Hosts.FirstOrDefault(host =>
                        host.HostIdentifier == messageNetPackage.HostId);

                    if (host == null)
                    {
                        host = new Host
                        {
                            HostIdentifier = messageNetPackage.HostId,
                            HostName  = (string)messageNetPackage["HostName"],
                            IPAdress  = (string)messageNetPackage["IPAdress"],
                            MacAdress = (string)messageNetPackage["MacAdress"]
                        };

                        await _dbContext.Hosts.AddAsync(host);
                    }
                    else
                    {
                        if (host.IPAdress != (string)messageNetPackage["IPAdress"])
                            host.IPAdress = (string)messageNetPackage["IPAdress"];

                        if (host.MacAdress !=
(string)messageNetPackage["MacAdress"])
                            host.MacAdress = (string)messageNetPackage["MacAdress"];
                    }

                    await _dbContext.SaveChangesAsync();

                    var usbDevice = _dbContext.USBDevices
                        .FirstOrDefault(device =>
                            device.Sn == (string)messageNetPackage["Sn"]);

                    if(usbDevice == null)
                    {
                        usbDevice = new USBDevice
                        {
                            Vid = (string)messageNetPackage["Vid"],
                            Pid = (string)messageNetPackage["Pid"],
                            Sn = (string)messageNetPackage["Sn"],
                            Png = (string)messageNetPackage["Png"]
                        };
                    }

                    var hostEvent = new HostEvent
                    {
                        EventType= (string)messageNetPackage["EventType"],
                        IsAllowed= (string)messageNetPackage["IsAllowed"],
                        USBDevice = usbDevice,
                        EventDate = DateTime.Now.Date,
                        EventTime = DateTime.Now.TimeOfDay,
                        Host = host
                    };

                    await _dbContext.HostEvents.AddAsync(hostEvent);

                    await _dbContext.SaveChangesAsync();
                }

                responseMessageNetPackage
                    .setStatusCode(DATA_SAVED_SUCCESS)
```

```
                    .addParameter("Text:",
                        "The USB connection event has been successfully saved.");

            }
            catch(Exception ex)
            {
                responseMessageNetPackage
                    .setStatusCode(DATA_SAVED_ERROR)
                    .addParameter("Text: ", ex.MessageNetPackage);
                _loggeLocalExceptionsInService.LogError(ex.MessageNetPackage);
            }

            await interactor.SendResponseAsync(responseMessageNetPackage,
messageNetPackage.HostId);
        }
    }
}
```

Веб-інтерфейс

```
const FilterPanale = (props) => {

  const [isFilterActive, setIsFilterActive] = useState(false);
  const [iconDirection, setIconDirection]   = useState("right");
  const [displayFilterPanale, setDisplayFilterPanale] = useState("d-none");

  const toggleFilterPanale = () => {
    if(!isFilterActive){

      setIconDirection("down");
      setDisplayFilterPanale("d-block");
    }
    else{
      setIconDirection("right")
      setDisplayFilterPanale("d-none");
    }
    setIsFilterActive(!isFilterActive)
  };

  return (
    <div className="">
      <Button variant="primary" className="mb-3" onClick={toggleFilterPanale}>
        <span className="me-2">Застосувати фільтри</span>
        <span><i className={`bi bi-caret-${iconDirection}-fill`}></i></span>
      </Button>

      <Card className={`${displayFilterPanale} pb-4`} body>{props.children}</Card>
  </div>
  );
}

export default FilterPanale;

const Header = (props) => {
  let classStyles = props.className ?? "";

  return (
    <header className={`${classStyles} py-4 border-bottom`}>
      <div className="container-fluid position-relative">
        <div className="row">
          <div className="col">
            <i className="bi bi-usb-symbol logo"></i>
          </div>
          <div className="col-3">
            <div className="float-end">
              <button data-bs-content="And here's some amazing content. It's very
engaging. Right?" className="btn btn-outline-primary me-2">Login</button>
              <button className="btn btn-primary">Sign-up</button>
            </div>
          </div>
        </div>
      </div>
    </header>
  );
}

export default Header;

const HostEventsPage = (props) => {
```

```
const [hostEvents, setHostEvents] = useState([]);
const { id } = useParams();

useEffect(() => {
  (async () => {
    await loadTableData();
  })();
}, []);

const loadTableData = async () => {
  try {
    let data = await getHostEventsAsync(id);
    setHostEvents(data.hostEvents);
  } catch (error) {
    console.log(error);
  }
}

return(
  <div className="">
    <h1 className="border-bottom border-2 pb-4 mb-5">Управління користувачами</h1>
    <FilterPanale>
      <Row>
        <Col>
          <Form.Label htmlFor="fp-usb-name">Назва пристрою</Form.Label>
          <InputGroup className="mb-3">
            <FormControl
              id="fp-usb-name"
              placeholder="Введіть назву пристрою"
            />
          </InputGroup>
        </Col>
        <Col>
          <Form.Label htmlFor="fp-usb-sn">Серійний номер</Form.Label>
          <InputGroup className="mb-3">
            <FormControl
              id="fp-usb-sn"
              placeholder="Серійний номер"
            />
          </InputGroup>
        </Col>
      </Row>
      <Row>
        <Col>
          <Form.Label htmlFor="fp-usb-vid">Vid</Form.Label>
          <InputGroup className="mb-3">
            <FormControl
              id="fp-usb-vid"
              placeholder="Введіть vid"
            />
          </InputGroup>
        </Col>
        <Col>
          <Form.Label htmlFor="fp-usb-pid">Pid</Form.Label>
          <InputGroup className="mb-3">
            <FormControl
              id="fp-usb-pid"
              placeholder="Введіть pid"
            />
          </InputGroup>
        </Col>
      </Row>
      <Button variant="primary">
        Застосувати
      </Button>
```

```
      </FilterPanale>

      <div className="mt-5">
        <ViewTable
          headTitle={["#","Дата", "Час", "Тип події", "Vid", "Pid", "Sn", "Статус"]}
          colParam={["id", "eventDate", "eventTime", "eventType", "vid", "pid"
,"sn", "isAllowed"]}
          dataSource={hostEvents}
          deleteRowAsync={deleteHostEventsAsync}
          reloadTableData={loadTableData}
        />
      </div>
    </div>
  );
}

export default HostEventsPage;


const HostsPage = () => {

  const navigate = useNavigate();
  const [hostsData, setHostsData] = useState([]);

  useEffect(() => {
    (async () => {
      try
      {
        setHostsData(await getHostsAsync());
      } catch(error){
        console.log(error);
      }
    })();

  }, []);

  const reloadHostsData = async () => {
    try{
      setHostsData(await getHostsAsync());
    } catch(error){
      console.log(error)
    }
  }

  const contextMenu = [
    {
      title: "Переглянути події",
      onClick: (event, id) => navigate(`/HostEvents/${id}`)
    },
  ]

  return (
    <div className="HostsPage">
    <h1 className="border-bottom border-2 pb-4 mb-5">Робочі станції</h1>

      <FilterPanale>
        <Row>
          <Col>
            <Form.Label htmlFor="fp-host-name">Назва хосту</Form.Label>
            <InputGroup className="mb-3">
              <FormControl
                id="fp-host-name"
                placeholder="Введіть назву хосту"
              />
            </InputGroup>
```

```
          </Col>
          <Col>
            <Form.Label htmlFor="fp-ip-address">IP адреса</Form.Label>
            <InputGroup className="mb-3">
              <FormControl
                id="fp-ip-address"
                placeholder="Введіть IP адресу"
              />
            </InputGroup>
          </Col>
          <Col>
            <Form.Label htmlFor="fp-mac-address">Мак адреса</Form.Label>
            <InputGroup className="mb-3">
              <FormControl
                id="fp-mac-address"
                placeholder="Введіть мак адресу"
              />
            </InputGroup>
          </Col>
        </Row>
        <Button variant="primary">
          Застосувати
        </Button>
      </FilterPanale>

      <div className="mt-5">
        <ViewTable
          headTitle={["#", "Host", "IP", "Mac"]}
          colParam={["id", "hostName", "ipAdress", "macAdress"]}
          modalForms={[
            {
              name:"hostName",
              placeholder:"Введіть назву хосту",
              label:"Назва хосту",
            },
            {
              name:"ip",
              placeholder:"Введіть ip адресу",
              label:"IP адреса"
            },
            {
              name:"mac",
              placeholder:"Введіть mac адресу",
              label:"Mac адреса"
            }
          ]}
          dataSource={hostsData}
          contextMenu={contextMenu}
          deleteRowAsync={deleteHostAsync}
          />
      </div>
    </div>
  );
}

export default HostsPage;


const Main = (props) => {
  let classStyles = props.className ?? "";
  return(
    <div className={`${classStyles} py-5`}>
       <Routes>
          <Route exact path='/' element={<HostsPage/>}/>
          <Route exact path="/UserManagement" element={<UserManagementPage/>}/>
```

```
          <Route exact path="/RulesManagement" element={<RulesManagementPage/>}/>
          <Route exact path="/HostEvents/:id" element={<HostEventsPage/>}/>
          <Route exact path="/USBDevice" element={<USBDevicePage/>}/>
      </Routes>
    </div>
  );
}

export default Main;

const RulesManagementPage = (props) => {

  const [editMode, setEditMode] = useState("d-none");
  const [hosts, setHosts] = useState([]);
  const [usbDevices, setUsbDevices] = useState([]);
  const [whitelists, setWhitelists] = useState([]);
  const [modalRowId, setModalRowId] = useState();
  const [displayUpdatingModal, setDisplayUpdatingModal] = useState(false);
  const [displayAddingModal, setDisplayAddingModal] = useState(false);

  useEffect(() => {
    (async () => {
      try {
        setWhitelists(await getWhitelistsAsync())
        setHosts(await getHostsAsync());
        setUsbDevices(await getUsbDevicesAsync());
      } catch (error) {
        console.log(error);
      }
    })();
  }, []);

  const handleEditCheckBox = (event) => {
    if (event.target.checked) {
      setEditMode("table-row");
    } else {
      setEditMode("d-none");
    }
  }

  const handleShowUpdatingModal = (event, rowId) => {
    setDisplayUpdatingModal(true);
    setModalRowId(rowId);
  }

  const handleShowAddingModal = (event) => {
    setDisplayAddingModal(true);
  }

  const handleHideUpdatingModal = (event) => {
    setDisplayUpdatingModal(false);
  }

  const handleHideAddingModal = (event) => {
    setDisplayAddingModal(false);
  }

  const handleDeleteWhitelist = async (event, rowId) => {
    await deleteWhitelistsAsync(rowId);
    await reloadTableAsync();
  }

  async function reloadTableAsync() {
    try {
      setWhitelists(await getWhitelistsAsync())
```

```
      setHosts(await getHostsAsync());
      setUsbDevices(await getUsbDevicesAsync());
    } catch (error) {
      console.log(error);
    }
  }

  return (
    <div>
      <h1 className="border-bottom border-2 pb-4 mb-5">Управління політиками
безпеки</h1>
      <FilterPanale>
        <Row>
          <Col>
            <Form.Label htmlFor="fp-host-name">Назва правила безпеки</Form.Label>
            <InputGroup className="mb-3 w-25">
              <FormControl
                id="fp-host-name"
                placeholder="Введіть назву правила безпеки"
              />
            </InputGroup>
          </Col>
        </Row>
        <Button variant="primary">
          Застосувати
        </Button>
      </FilterPanale>

      <div className="mt-5">
        <Form.Check onChange={handleEditCheckBox} className="mb-4" type="checkbox"
label="Редагувати" />

        <Button className={`${editMode} btn-width`} onClick={handleShowAddingModal}
variant="primary mb-4">
          <i className="bi bi bi-plus-circle text-size-small me-2"></i>
          <span className="text-size-small">Додати</span>
        </Button>

        <Table hover>
          <thead>
            <tr>
              <th>Назва правила</th>
              <th>Хости</th>
              <th>Пристрої</th>
              <th className={editMode}>Редагувати</th>
            </tr>
          </thead>
          <tbody>
            {whitelists?.map(rule => {
              return (
                <tr>
                  <td className="fs-5">{rule.ruleName}</td>
                  <td>
                    <div className="rule-list">
                      {rule?.hostsId.map(hostId => {
                        let host = hosts.find(host => host.id === hostId);
                        if (!host) return null;
                        return (
                          <Badge className="p-2 m-1" bg="primary">
                            {host?.hostName}
                          </Badge>
                        );
                      })}
                    </div>
                  </td>
```

```
                  <td>
                    <div className="rule-list">
                      {rule?.usbDevicesId.map(usbDeviceId => {
                        let usb = usbDevices.find(usbDevice => usbDevice.id ===
usbDeviceId);
                        if (!usb) return null;
                        return (
                          <Badge bg="primary pe-3 ps-3 m-1">
                            <span>
                              <p>Name:{usb?.name}</p>
                              <p>sn:{usb?.sn}</p>
                            </span>
                          </Badge>
                        );
                      })}
                    </div>
                  </td>
                  <td>
                    <div className={editMode}>
                      <Button onClick={e => handleShowUpdatingModal(e, rule.id)}
variant="primary me-2">
                        <i className="bi bi-pencil-square text-size-small me-1"></i>
                        <span className="text-size-small">Змінити</span>
                      </Button>
                      <Button onClick={e => handleDeleteWhitelist(e, rule.id)}
variant="primary">
                        <i className="bi bi-trash text-size-small me-1"></i>
                        <span className="text-size-small">Видалити</span>
                      </Button>
                    </div>
                  </td>
                </tr>
              );
            })}
          </tbody>
        </Table>

      </div>
      <SecRuleModal
        modalTitle="Створити білий список"
        isDisplayModal={displayAddingModal}
        onHideModal={handleHideAddingModal}
        reloadTableAsync={reloadTableAsync}
        onSaveData={addWhitelistAsync}
        hostsData={hosts}
        usbDevicesData={usbDevices}
      />

      <SecRuleModal
        modalTitle="Редагувати білий список"
        isDisplayModal={displayUpdatingModal}
        onHideModal={handleHideUpdatingModal}
        onSaveData={updateWhitelistAsync}
        reloadTableAsync={reloadTableAsync}
        rowId={modalRowId}
        whitelists={whitelists}
        hostsData={hosts}
        usbDevicesData={usbDevices}
      />
    </div>
  );
}

export default RulesManagementPage;
```

```
const SecRuleModal = (props) => {

  const {
    isDisplayModal,
    onHideModal,
    modalTitle,
    hostsData,
    rowId,
    usbDevicesData,
    onSaveData,
    reloadTableAsync } = props;

  const [hostsInWhitelist, setHostsInWhitelist] = useState([]);
  const [usbDevicesInWhitelist, setUsbDevicesInWhitelist] = useState([]);
  const [secRuleName, setSetRuleName] = useState("");

  useEffect(() => {
    (async () => {
      try {
        if (rowId && isDisplayModal) {
          let whitelist = await getWhitelistsByIdAsync(rowId);
          setSetRuleName(whitelist.ruleName);
          setHostsInWhitelist([...whitelist.hostsId]);
          setUsbDevicesInWhitelist([...whitelist.usbDevicesId]);
        }
      } catch (error) {
        console.log(error);
      }
    })();
  }, [isDisplayModal, rowId]);

  const handleAddHostToRule = (hostId) => {
    setHostsInWhitelist([
      ...hostsInWhitelist,
      +hostId
    ]);
  }

  const handleAddUsbDeviceToRule = (usbDeviceId) => {
    setUsbDevicesInWhitelist([
      ...usbDevicesInWhitelist,
      +usbDeviceId
    ]);
  }

  const handleChangeSecRuleName = (event) => {
    setSetRuleName(event.target.value);
  }

  const handleRemoveHostFromWhitelistOnClientSide = (event, id) => {
    let hosts = hostsInWhitelist
      .filter(hostId => hostId !== id);
    setHostsInWhitelist(hosts);
  }

  const handleRemoveUsbDeviceFromWhitelistOnClientSide = (event, id) => {
    let usbDevices = usbDevicesInWhitelist
      .filter(usbDeviceId => usbDeviceId !== id);
    setUsbDevicesInWhitelist(usbDevices);
  }

  const handleSaveData = async (event) => {
    let whitelist = {
      "ruleName": secRuleName,
      "hostsId": hostsInWhitelist,
```

```
      "usbDevicesId": usbDevicesInWhitelist
    }
    await onSaveData(whitelist, rowId);
    await reloadTableAsync();
    onHideModal();
  }

  return (
    <div>
      <Modal aria-labelledby="contained-modal-title-vcenter"
        centered show={isDisplayModal} onHide={onHideModal}>
        <Modal.Header closeButton>
          <Modal.Title>{modalTitle}</Modal.Title>
        </Modal.Header>
        <Modal.Body className="ms-2 me-2">

          <Form.Label htmlFor="sec_rule_id">Назва правила</Form.Label>
          <InputGroup>
            <FormControl
              id="sec_rule_id"
              value={secRuleName}
              placeholder="Введіть назву правила"
              onChange={handleChangeSecRuleName}
            />
          </InputGroup>

          <InputGroup className="m-3">
            <DropdownButton onSelect={handleAddHostToRule} title="Додати хости">
              {hostsData?.map(item => {
                if (hostsInWhitelist.includes(item.id))
                  return null;
                let dropdownText = `${item["hostName"]}`
                return (<Dropdown.Item eventKey={item.id} href="#">
                  <Badge className="p-2" bg="primary">{dropdownText}</Badge>
                </Dropdown.Item>);
              })}
            </DropdownButton>
          </InputGroup>

          <div className="w-100 bg-light shadow-sm p-1 badge-box-h overflow-auto">
            {hostsInWhitelist.map(id => {
              console.log(id);
              return (
                <Badge className="p-2 m-1 position-relative" bg="primary">
                  <i onClick={e => handleRemoveHostFromWhitelistOnClientSide(e, id)}
                    className="bi bi-trash-fill delete-icon-position delete-
icon"></i>
                  {hostsData?.find(host => host.id === id)?.hostName}
                </Badge>
              );
            })}
          </div>

          <InputGroup className="m-3">
            <DropdownButton onSelect={handleAddUsbDeviceToRule} title="Додати usb
пристрої">
              {usbDevicesData?.map(usb => {
                if (usbDevicesInWhitelist.includes(usb.id))
                  return null;
                return (
                <Dropdown.Item eventKey={usb.id} href="#">
                  <div className="mb-1">
                    <Badge className="p-2 w-100" bg="primary">
                      <span>
                        <p>Name:{usb["name"]}</p>
```

```
                      <p>sn:{usb["sn"]}</p>
                    </span>
                  </Badge>
                </div>
              </Dropdown.Item>);
            })}
          </DropdownButton>
        </InputGroup>

        <div className="w-100 bg-light shadow-sm p-1 badge-box-h overflow-auto">
          {usbDevicesInWhitelist?.map(id => {
            let usb = usbDevicesData?.find(usbDevice => usbDevice.id === id);
            return (
              <Badge className="p-2 m-1 position-relative" bg="primary">
                <i onClick={e => handleRemoveUsbDeviceFromWhitelistOnClientSide(e,
id)}
                  className="bi bi-trash-fill delete-icon-position delete-
icon"></i>
                <span>
                  <p>Name:{usb["name"]}</p>
                  <p>sn:{usb["sn"]}</p>
                </span>
              </Badge>
            );
          })}
        </div>

      </Modal.Body>
      <Modal.Footer>
        <Button variant="primary" onClick={handleSaveData}>
          Зберегти зміни
        </Button>
      </Modal.Footer>
    </Modal>
  </div>
 );
};

export default SecRuleModal;

const Sidebar = (props) => {
  let MenuBarStyles = props.className ?? "";

  return (
    <div className={`${MenuBarStyles} border-end border-2`}>
      <nav className="nav nav-pills flex-column mb-auto fs-5">
        <NavLink className="p-2 dbl nav-link w-100" to="/">
          Хости
        </NavLink>

        <NavLink className="p-2 dbl nav-link w-100" to="/RulesManagement">
          Управління політиками безпеки
        </NavLink>

        <NavLink className="p-2 dbl nav-link w-100" to="/UserManagement">
          Управління користувачами
        </NavLink>

        <NavLink className="p-2 dbl nav-link w-100" to="/USBDevice">
          USB пристрої
        </NavLink>
      </nav>
    </div>
  );
}
```

```
export default Sidebar;


const TableModal = (props) => {

  const {
    modalTitle,
    isDisplayModal,
    modalRowId,
    modalForms,
    onHideModal,
    onModalFormChange,
    onSaveData} = props;

  return (
    <div>
      <Modal aria-labelledby="contained-modal-title-vcenter"
      centered show={isDisplayModal} onHide={onHideModal}>
        <Modal.Header closeButton>
          <Modal.Title>{modalTitle}</Modal.Title>
        </Modal.Header>
        <Modal.Body>
          {modalForms?.map((form, index) => {
            return(
              <Row key={form.name}>
              <Form.Label htmlFor={`mf-${index + 1}`}>{form.label}</Form.Label>
              <InputGroup className="mb-3">
                <FormControl
                  id={`mf-${index + 1}`}
                  name={form.name}
                  placeholder={form.placeholder}
                  onChange={onModalFormChange}
                />
              </InputGroup>
            </Row>);
          })}
        </Modal.Body>
        <Modal.Footer>
          <Button variant="primary" onClick={(e) => onSaveData(e, modalRowId)}>
            Зберегти зміни
          </Button>
        </Modal.Footer>
      </Modal>
    </div>
  );
}


const USBDevicePage = () => {

  const [usbDevices, setUSBDevices] = useState();
  const [searchUsbDevicesData, setSearchUsbDevicesData] =useState({
    name:"", pid:"", vid:"", sn:""});

  useEffect(() => {
    (async () => {
      try {
        setUSBDevices(await getUsbDevicesAsync());
      } catch (error) {
        console.log(error);
      }
    })();
  }, []);
```

```
const reloadUsbDevices = async () => {
  try {
    setUSBDevices(await getUsbDevicesAsync());
  } catch (error) {
    console.log(error)
  }
}

const handleFormChange = (event) => {
  const {name, value} = event.target;
  console.log(searchUsbDevicesData);
  setSearchUsbDevicesData({...searchUsbDevicesData, [name]:value})
}

const handleSearchCertainUSBDevices = async (event) => {
  let USBDevices = await searchCertainUSBDevicesAsync(searchUsbDevicesData);
  console.log(USBDevices);
  setUSBDevices(USBDevices);
}

const updatingModalForms = [
  {
    name: "name",
    placeholder: "Введіть назву пристрою",
    label: "Назва пристрою"
  },
  {
    name: "pid",
    placeholder: "Введіть pid",
    label: "Pid"
  },
  {
    name: "vid",
    placeholder: "Введіть vid",
    label: "Vid"
  },
  {
    name: "sn",
    placeholder: "Введіть серійний номер",
    label: "Серійний номер"
  }
];

const addingModalForms = updatingModalForms;

return (
  <div className="">
    <h1 className="border-bottom border-2 pb-4 mb-5">USB пристрої</h1>
    <FilterPanale>
      <Row>
        <Col>
          <Form.Label htmlFor="fp-usb-name">Назва пристрою</Form.Label>
          <InputGroup className="mb-3">
            <FormControl
              id="fp-usb-name"
              name="name"
              placeholder="Введіть назву пристрою"
              onChange={handleFormChange}
            />
          </InputGroup>
        </Col>
        <Col>
          <Form.Label htmlFor="fp-usb-sn">Серійний номер</Form.Label>
          <InputGroup className="mb-3">
            <FormControl
```

```
                id="fp-usb-sn"
                name="sn"
                placeholder="Серійний номер"
                onChange={handleFormChange}
              />
            </InputGroup>
          </Col>
        </Row>
        <Row>
          <Col>
            <Form.Label htmlFor="fp-usb-vid">Vid</Form.Label>
            <InputGroup className="mb-3">
              <FormControl
                id="fp-usb-vid"
                name="vid"
                placeholder="Введіть vid"
                onChange={handleFormChange}
              />
            </InputGroup>
          </Col>
          <Col>
            <Form.Label htmlFor="fp-usb-pid">Pid</Form.Label>
            <InputGroup className="mb-3">
              <FormControl
                id="fp-usb-pid"
                name="pid"
                placeholder="Введіть pid"
                onChange={handleFormChange}
              />
            </InputGroup>
          </Col>
        </Row>
        <Button onClick={handleSearchCertainUSBDevices} variant="primary">
          Застосувати
        </Button>
      </FilterPanale>
      <div className="mt-5">
        <ViewTable
          headTitle={["#", "Назва пристрою", "Pid", "Vid", "Sn"]}
          colParam={["id", "name", "pid", "vid", "sn"]}
          updatingModalForms={updatingModalForms}
          addingModalForms={addingModalForms}
          dataSource={usbDevices}
          addRowAsync={addUsbDevicesAsync}
          updateRowAsync={updateUsbDevicesAsync}
          deleteRowAsync={deleteUsbDevicesAsync}
          reloadTableData={reloadUsbDevices}
        />
      </div>
    </div>
  );
}

export default USBDevicePage;


const UserManagementPage = (props) => {

  const [usersData, setUsersData] = useState([]);

  useEffect(() => {
    (async () => {
      try
      {
        setUsersData(await getAllUsersAsync());
```

```
    } catch(error){
      console.log(error);
    }
  })();
}, []);

const reloadUsersData = async () => {
  try{
    setUsersData(await getAllUsersAsync());
  } catch(error){
    console.log(error)
  }
}

const updatingModalForms = [
  {
    name:"userName",
    placeholder:"Введіть нікнейм",
    label:"Нікнейм",
  },
  {
    name:"name",
    placeholder:"Введіть ім'я",
    label:"Ім'я"
  },
  {
    name:"surname",
    placeholder:"Введіть призвище",
    label:"Призвище"
  },
  {
    name:"email",
    placeholder:"Введіть email",
    label:"Email"
  },
  {
    name:"role",
    placeholder:"Введіть права доступу",
    label:"Права доступу"
  }
];

const addingModalForms = [
  ...updatingModalForms,
  {
    name:"password",
    placeholder:"Введіть пароль",
    label:"Пароль"
  }
];

return (
  <>
    <h1 className="border-bottom border-2 pb-4 mb-5">Управління користувачами</h1>
    <FilterPanale>
      <Row>
        <Col>
          <Form.Label htmlFor="fp-nick-name">Нікнейм</Form.Label>
          <InputGroup className="mb-3">
            <FormControl
              id="fp-nick-name"
              placeholder="Введіть нікнейм"
            />
          </InputGroup>
        </Col>
```

```
          <Col>
            <Form.Label htmlFor="fp-name">Ім'я</Form.Label>
            <InputGroup className="mb-3">
              <FormControl
                id="fp-name"
                placeholder="Введіть ім'я"
              />
            </InputGroup>
          </Col>
          <Col>
            <Form.Label htmlFor="fp-surname">Призвище</Form.Label>
            <InputGroup className="mb-3">
              <FormControl
                id="fp-surname"
                placeholder="Введіть мак адресу"
              />
            </InputGroup>
          </Col>
        </Row>
        <Row>
          <Col>
            <Form.Label htmlFor="fp-email">Email</Form.Label>
              <InputGroup className="mb-3">
                <FormControl
                  id="fp-email"
                  placeholder="Введіть email"
                />
            </InputGroup>
          </Col>
          <Col>
            <Form.Label htmlFor="fp-role">Роль</Form.Label>
              <InputGroup className="mb-3">
                <FormControl
                  id="fp-role"
                  placeholder="Введіть роль"
                />
            </InputGroup>
          </Col>
        </Row>
        <Button variant="primary">
          Застосувати
        </Button>
      </FilterPanale>
      <div className="mt-5">
        <ViewTable
          headTitle={["#", "Нікнейм", "Ім'я", "Призвище", "Email", "Роль"]}
          colParam={["id", "userName", "name" ,"surname", "email", "role"]}
          updatingModalForms={updatingModalForms}
          addingModalForms={addingModalForms}
          dataSource={usersData}
          addRowAsync={addUserAsync}
          updateRowAsync={updateUserAsync}
          deleteRowAsync={deleteUserAsync}
          reloadTableData={reloadUsersData}
        />
      </div>
    </>
  );
}

const ViewTable = (props) => {
  const {
    headTitle,
    colParam,
    addingModalForms,
```

```
    updatingModalForms,
    dataSource,
    reloadTableData,
    addRowAsync,
    updateRowAsync,
    deleteRowAsync,
    contextMenu,
  } = props;

  const [editMode, setEditMode] = useState("d-none");
  const [modalRowId, setModalRowId] = useState();
  const [displayUpdatingModal, setDisplayUpdatingModal] = useState(false);
  const [displayAddingModal, setDisplayAddingModal] = useState(false);
  const [dataFromAddingForm, setDataFromAddingForm] =
useState(initFormValues(addingModalForms));
  const [dataFromUpdatingForm, setDataFromUpdatingForm] =
useState(initFormValues(updatingModalForms));
  const [hostId, setHostId] = useState();
  const [xMenuPos, setXMenuPos] = useState("0px");
  const [yMenuPos, setYMenuPos] = useState("0px");
  const [menu, showMenu] = useState(false);

  const handleEditCheckBox = (event) => {
    if(event.target.checked){
      setEditMode("d-block");
    }else{
      setEditMode("d-none");
    }
  }

  const handleShowUpdatingModal = (event, rowId) => {
    setModalRowId(rowId);
    setDisplayUpdatingModal(true);
  }

  const handleHideUpdatingModal = (event) => {
    setDisplayUpdatingModal(false);
  }

  const handleShowAddingModal = (event) => {
    setDisplayAddingModal(true);
  }

  const handleHideAddingModal = (event) => {
    setDisplayAddingModal(false);
  }

  const handleModalFormChange = (event, newData, setNewData) => {
    const {name, value} = event.target;
    console.log(newData);
    setNewData({...newData, [name]:value})
  }

  const handleAddRowAsync = async (event, rowId) => {
    try{
      await addRowAsync(dataFromAddingForm);
      await reloadTableData();

    }catch(error){
      console.log(error);
    }finally{
      setDisplayAddingModal(false);
    }
  }
```

```
  const handleUpdateRowAsync = async (event, rowId) => {
    try{
      await updateRowAsync(rowId, dataFromUpdatingForm);
      await reloadTableData();
    } catch(error){
      console.log(error);
    }finally{
      setDisplayUpdatingModal(false);
    }
  }
  const handleDeleteRowAsync = async (event, rowId) => {
    try{
      await deleteRowAsync(rowId);
      await reloadTableData();
    }catch(error){
      console.log(error);
    }
  }

  const handleOpenContextMenu = (event, index) => {
    console.log(event);
    setXMenuPos(`${event.pageX}px`);
    setYMenuPos(`${event.pageY}px`);
    setHostId(index);
    showMenu(true);
  }

  const handleOpenMenuAction = (event, menu) => {
    menu.onClick(event, hostId);
    showMenu(false);
  }

  function initFormValues(modalForms) {
      return modalForms?.reduce((values, form) => {
        values[form.name] = "";
        return values;
    }, {});
  }

  let addButton = null;
  let updateButton = null;
  let deleteButton = null;
  let editCheckBox = null;
  let showContextMenu = null;

  if(addRowAsync){
    addButton =
      <Button className={`${editMode} btn-width`} onClick={handleShowAddingModal}
variant="primary mb-4">
        <i className="bi bi bi-plus-circle text-size-small me-2"></i>
        <span className="text-size-small">Додати</span>
      </Button>
  }

  if(updateRowAsync){
    updateButton = {}
    updateButton["show"] = function(data) {
      return(
        <Button onClick={e => handleShowUpdatingModal(e, data.id)} variant="primary
me-2">
          <i className="bi bi-pencil-square text-size-small me-1"></i>
          <span className="text-size-small">Змінити</span>
        </Button>
      );
    }
```

```
  }

  if(deleteRowAsync){
    deleteButton = {}
    deleteButton["show"] = function(data) {
      return(
        <Button onClick={e => handleDeleteRowAsync(e, data.id)} variant="primary">
          <i className="bi bi-trash text-size-small me-1"></i>
          <span className="text-size-small">Видалити</span>
        </Button>
      );
    }
  }

  if(addRowAsync || updateRowAsync || deleteRowAsync){
    editCheckBox =
      <Form.Check onChange={handleEditCheckBox} className="mb-4" type="checkbox"
label="Редагувати"/>
  }

  if (menu && contextMenu) {

    showContextMenu = (
      <div className="context-menu rounded shadow bg-light" style={{ top: yMenuPos,
left: xMenuPos }}>
        <CloseButton onClick={e => showMenu(false)}/>
        {contextMenu.map(menu => {
          return(
            <p className="border-bottom border-top" onClick={e =>
handleOpenMenuAction(e, menu)}>
              {menu.title}
            </p>
          );
        })}
      </div>
    );
  }

  return (
    <>
      {editCheckBox}
      {addButton}

      <Table hover>
        <thead>
          <tr>
            {headTitle?.map(title => <th key={title}>{title}</th>)}
            <th className={editMode}>Редагувати</th>
          </tr>
        </thead>
        <tbody>
          {dataSource?.map((data, i) => {
            let row = colParam.map(param => <td key={param}>{data[param]}</td>);
            return (
                <tr onClick={e => handleOpenContextMenu(e, data.id)} key={data.id}>
                  {row}
                  <td className={editMode}>
                    {(updateButton) ? updateButton.show(data) : null}
                    {(deleteButton) ? deleteButton.show(data) : null}
                  </td>
                </tr>
              );
          })}
        </tbody>
      </Table>
```

```
      {showContextMenu}
      <TableModal
        modalTitle="Редагувати рядок"
        isDisplayModal={displayUpdatingModal}
        onHideModal={handleHideUpdatingModal}
        onShowModal={handleShowUpdatingModal}
        onModalFormChange={e =>
          handleModalFormChange(e,
          dataFromUpdatingForm,
          setDataFromUpdatingForm)}
        onSaveData={handleUpdateRowAsync}
        modalRowId={modalRowId}
        modalForms={updatingModalForms}
      />

      <TableModal
        modalTitle="Додати новий запис"
        isDisplayModal={displayAddingModal}
        onHideModal={handleHideAddingModal}
        onShowModal={handleShowAddingModal}
        onModalFormChange={e =>
          handleModalFormChange(e,
          dataFromAddingForm,
          setDataFromAddingForm)}
        onSaveData={handleAddRowAsync}
        modalForms={addingModalForms}
      />
    </>
  );
}

export default ViewTable;

import axios from "axios";
import apiURL from "../apiURL";
import moment from 'moment';

export const getHostsAsync = async () => {
    const res = await axios.get(apiURL.getHostsURL());
    return res.data;
}

export const getAllUsersAsync = async () => {
    const res = await axios.get(apiURL.getUsersURL());
    return res.data;
}

export const deleteHostAsync = async (id) => {
    await axios.delete(apiURL.deleteHostURL(id));
}

export const addUserAsync = async(userData) => {
    await axios.post(apiURL.addUserURL(), userData);
}

export const deleteUserAsync = async (id) => {
    await axios.delete(apiURL.deleteUserURL(id));
}

export const updateUserAsync = async(id, updatedUserData) => {
    await axios.put(apiURL.updateUserURL(id), updatedUserData);
}

export const getUsbSecRulesAsync = async() => {
    const res = await axios.get(apiURL.getUsbSecRulesURL());
```

```
    return res.data;
}

export const getUsbDevicesAsync = async() => {
    const res = await axios.get(apiURL.getUsbDevicesURL());
    return res.data;
}

export const deleteUsbDevicesAsync = async (id) => {
    await axios.delete(apiURL.deleteUsbDevicesURL(id));
}

export const updateUsbDevicesAsync = async(id, updatedUserData) => {
    await axios.put(apiURL.updateUsbDevicesURL(id), updatedUserData);
}

export const addUsbDevicesAsync = async(UsbDevices) => {
    await axios.post(apiURL.addUsbDevicesURL(), UsbDevices);
}

export const getWhitelistsAsync = async() => {
    const res = await axios.get(apiURL.getWhitelistsURL());
    return res.data;
}

export const getWhitelistsByIdAsync = async(id) => {
    const res = await axios.get(apiURL.getWhitelistsByIdURL(id));
    return res.data.pop();
}

export const deleteWhitelistsAsync = async(id) => {
    await axios.delete(apiURL.deleteWhitelistsURL(id));
}

export const addWhitelistAsync = async (whitelist) => {
    await axios.post(apiURL.addWhitelistURL(), whitelist);
}

export const updateWhitelistAsync = async(whitelist, id) => {
    await axios.put(apiURL.updateWhitelistsURL(id), whitelist)
}

export const deleteHostEventsAsync = async(id) => {
    await axios.delete(apiURL.deleteHostEventURL(id));
}

export const getHostEventsAsync = async(id) => {
    const res = await axios.get(apiURL.getHostEventsURL(id));
    const data = res.data;
    let hostEvents = data.hostEvents.map(hostEvent => {
        let time =  hostEvent.eventTime.split(':');
        hostEvent.eventTime = moment({hour: time[0], minute: time[1], seconds:
time[2]}).format("H:mm:ss");
        hostEvent.eventDate = moment(hostEvent.eventData.split('T')[0]).format("DD-
MM-YYYY");
        return hostEvent;
    });
    data.hostEvents = hostEvents;
    return data;
}

export const searchCertainUSBDevicesAsync = async (data) => {
    const res = await axios.post(apiURL.searchCertainUSBDevicesURL(), data);
    return res.data;
}
```

```
export const getAllowedHostDeviceAsync = async (id) => {
    const res = await axios.get(apiURL.getAllowedHostDeviceURL(id));
    return res.data;
}

const apiURL = {
    getHostsURL: () => `${process.env.REACT_APP_APIDOMAIN}/api/Host`,
    deleteHostURL: (id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/Host/DeleteHost/${id}`,
    searchCertainHostsURL: () =>
`${process.env.REACT_APP_APIDOMAIN}/api/Host/SearchCertainHosts`,
    getUsersURL: () => `${process.env.REACT_APP_APIDOMAIN}/api/User`,
    addUserURL: () => `${process.env.REACT_APP_APIDOMAIN}/api/User/AddNewUser`,
    updateUserURL: (id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/User/UpdateUser/${id}`,
    deleteUserURL: (id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/User/DeleteUser/${id}`,
    getUsbSecRulesURL: () => `${process.env.REACT_APP_APIDOMAIN}/api/Whitelist`,
    getUsbDevicesURL: () => `${process.env.REACT_APP_APIDOMAIN}/api/USBDevice`,
    getAddUsbDevicesURL: () => `${process.env.REACT_APP_APIDOMAIN}/api/UsbDevices`,
    updateUsbDevicesURL: (id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/USBDevice/UpdateUSBDevice/${id}`,
    deleteUsbDevicesURL: (id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/USBDevice/DeleteUSBDevice/${id}`,
    addUsbDevicesURL: () =>
`${process.env.REACT_APP_APIDOMAIN}/api/USBDevice/AddUSBDevice`,
    getWhitelistsURL: () => `${process.env.REACT_APP_APIDOMAIN}/api/Whitelist`,
    getWhitelistsByIdURL: (id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/Whitelist/${id}`,
    addWhitelistURL:() =>
`${process.env.REACT_APP_APIDOMAIN}/api/Whitelist/AddNewWhitelist`,
    deleteWhitelistsURL:(id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/Whitelist/DeleteWhitelist/${id}`,
    updateWhitelistsURL:(id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/Whitelist/UpdateWhitelist/${id}`,
    getHostEventsURL:(id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/HostEvent/${id}`,
    deleteHostEventURL:(id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/HostEvent/DeleteUsbEvent/${id}`,
    searchCertainUSBDevicesURL:() =>
`${process.env.REACT_APP_APIDOMAIN}/api/USBDevice/SearchCertainUSBDevice`,
    getAllowedHostDeviceURL:(id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/HostAllowedUSBDevices/${id}`,
    blockUSBDeviceURL:(id) =>
`${process.env.REACT_APP_APIDOMAIN}/api/HostAllowedUSBDevices/BlockUSBDevice/${id}`,
    enableUSBDeviceURL:(id) =>
`${process.env.REACT_APP_APIDOMAIN}//api/HostAllowedUSBDevices/EnableUSBDevice/${id}
`
}
export default apiURL;

function App() {
  return (
    <div className="container-fluid">
      <div className="row">
        <Header className="app-header"/>
      </div>
      <div className="row">
        <Sidebar className="app-sidebar"/>
        <Main className="app-main"/>
      </div>
    </div>
  );
}export default App;
```