Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: "Розроблення мобільного додатка SafeBarrier для блокування доступу до Internet з використанням технологій локального VPN, Kotlin та Android SDK"

Виконав: студент 4 курсу групи
ІСТ-41
спеціальності
126 "Інформаційні системи та технології"
(шифр і назва напряму підготовки, спеціальності)

Джугалик Д. І.
(прізвище та ініціали)

Керівник Бекас Б. О.
(прізвище та ініціали)

Сторожук О. Л.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 «Інформаційні системи та технології»
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІСКМ
______________ Сторожук О.Л.
«15» 11 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Джугалику Денису Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення мобільного додатка SafeBarrier для блокування доступу до Internet з використанням технологій локального VPN, Kotlin та Android SDK

Керівники роботи *Сторожук Олександр Леонідович, канд. техн. наук, доцент; Бекас Богдан Олексійович, старший викладач*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "15" листопада 2024 року №.С-884
2. Термін подання студентом роботи 12 червня 2025 року
3. Вихідні дані до роботи Розроблено мобільний застосунок, що забезпечує обмеження доступу до мережі Інтернет шляхом використання локального VPN.

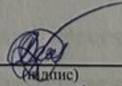
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1. Стан проблемної області
2. Інформаційне та математичне забезпечення
3. Програмне та технічне забепечення

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалів до доповіді

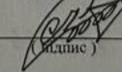
6. Дата видачі завдання "18" листопада 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	19.11.24 – 15.01.25	Виконано
2	Постановка задачі та її формалізації	15.01.25 – 30.01.25	Виконано
3	Реалізація мобільного застосунку	30.01.25 – 30.04.25	Виконано
4	Тестування мобільного застосунку	30.04.25 – 15.05.25	Виконано
5	Виправлення дефектів	15.05.25 – 30.05.25	Виконано
6	Оформлення пояснювальної записки	30.05.25 – 09.06.25	Виконано

Студент  Джугалик Д. І.
(підпис) (прізвище та ініціали)

Керівники роботи Бекас Б. О.
(підпис) (прізвище та ініціали)

 Сторожук О. Л.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 54 сторінок пояснювальної записки, 18 рисунки, 4 додатка, 15 джерел.

Результатом виконання дипломної роботи є розробка мобільного додатка SafeBarrier для блокування доступу до Інтернету з використанням технологій локального VPN, Kotlin та Android SDK. Основні функціональні можливості SafeBarrier включають реалізацію локального VPN-сервісу за допомогою розширення класу VpnService, що дозволяє перехоплювати та обробляти мережевий трафік без зовнішніх серверів. Користувачі можуть блокувати доступ до певних веб-ресурсів або сервісів, забезпечуючи додатковий рівень контролю.

Ключові слова: SafeBarrier, мобільний додаток, блокування Інтернету, локальний VPN, Kotlin, Android SDK, контроль трафіку, фільтрація контенту.

ABSTRACT

The diploma thesis contains 54 pages of explanatory notes, 18 figures, 4 appendices, 15 sources.

The result of the thesis is the development of the SafeBarrier mobile application for blocking access to the Internet using local VPN, Kotlin and Android SDK technologies. The main functionalities of SafeBarrier include the implementation of a local VPN service using the VpnService class extension, which allows intercepting and processing network traffic without external servers. Users can block access to certain web resources or services, providing an additional level of control.

Keywords: SafeBarrier, mobile application, Internet blocking, local VPN, Kotlin, Android SDK, traffic control, content filtering.

ТЕХНІЧНЕ ЗАВДАННЯ

Для розроблення мобільного додатка **SafeBarrier** для блокування доступу до Internet з використанням технологій локального VPN, Kotlin та Android SDK передбачено створення кількох ключових функціональних модулів. Необхідно реалізувати систему локального VPN, яка дозволить перехоплювати та контролювати мережевий трафік на пристрої користувача без передачі даних на зовнішні сервери.

Додаток повинен забезпечувати можливість вибору, які саме додатки або типи трафіку блокувати, а також надавати користувачу інтерфейс для керування цими налаштуваннями. Інтерфейс має бути зрозумілим, інтуїтивно доступним та відповідати сучасним вимогам UI/UX-дизайну Android-додатків.

Також необхідно реалізувати систему журналювання, яка фіксує спроби виходу в Інтернет заблокованими додатками або службами, з можливістю перегляду статистики користувачем.

Усі компоненти мають бути реалізовані мовою Kotlin із використанням Android SDK, дотримуючись рекомендацій Google щодо безпеки, продуктивності та ефективності мобільних додатків.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 7
ВСТУП 8
РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ 10
1.1 Існуючі підходи до обмеження доступу до Інтернету на Android 10
1.2 Технологія локального VPN для моніторингу та фільтрації трафіку 11
1.3 Огляд інструментів, які реалізують подібну функціональність 13
1.3.1 Blokada - універсальний блокувальник реклами 13
1.3.2 Intra - DNS-захист від Google Jigsaw 14
1.4 Порівняльний аналіз 15
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ 17
2.1 Формування дерева проблем та дерева цілей 17
2.2 Об'єкти дослідження та предметна область 21
2.3 Інформаційна модель системи 22
2.4 Обмеження на вхідні та вихідні дані 24
2.5 Концептуальне проектування системи 25
2.6 Реалізація математичних і алгоритмічних методів 28
2.7 Використання нативного коду для фільтрації мережевого трафіку 29
РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ 34
3.1 Структура проекту та середовище розробки 34
3.2 Функціонал розробленого додатку 41
3.3 Апаратне і програмне забезпечення 46
ВИСНОВОК 47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 48
ДОДАТКИ 50
ДОДАТОК А 50
ДОДАТОК Б 64
ДОДАТОК В 67
ДОДАТОК Г 70

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface;

VPN - Virtual Private Network;

DNS - Domain Name System;

IP - Internet Protocol;

TCP - Transmission Control Protocol;

UDP - User Datagram Protocol;

ICMP - Internet Control Message Protocol;

HTTP/HTTPS - HyperText Transfer Protocol (Secure);

TUN - Tunnel Interface;

MTU - Maximum Transmission Unit;

OSI - Open Systems Interconnection;

JNI – Java Native Interface;

UID – User Identifier;

SQLite – вбудована реляційна база даних;

MVVM – Model-View-ViewModel – архітектурний патерн;

LiveData – компонент Android Architecture для спостереження за даними;

SELinux – Security-Enhanced Linux;

VpnService – базовий клас Android для створення VPN-додатків;

TUN Interface – віртуальний мережевий інтерфейс третього рівня OSI;

ConnectivityManager – сервіс Android для управління мережевими з’єднаннями;

PackageManager – сервіс Android для роботи з встановленими пакетами;

SharedPreferences – механізм зберігання налаштувань Android;

ВСТУП

Сучасний світ характеризується стрімкою цифровізацією всіх сфер життєдіяльності, що супроводжується експоненціальним зростанням використання мобільних пристроїв та їх постійним підключенням до глобальної мережі Інтернет. За статистичними даними, понад 6,8 мільярда людей у світі активно користуються мобільними пристроями, проводячи в середньому 7-8 годин на добу у мережі. Це значне поширення цифрових технологій стимулює розвиток інформаційних систем, але одночасно призводить до збільшення кількості загроз інформаційній безпеці.

Зростання цифрової активності супроводжується значними ризиками, серед яких особливо варто виділити несанкціонований доступ до персональних даних, витік конфіденційної інформації, кібератаки різного ступеня складності, небажаний мережевий трафік та надмірне споживання мобільного Інтернету. Особливо гостро стоїть питання контролю доступу до мережевих ресурсів у контексті сучасних викликів цифрової безпеки. Це питання є актуальним для широкого кола користувачів: батьків, які прагнуть убезпечити дітей від шкідливого контенту; освітніх закладів, що потребують підтримки продуктивності навчального процесу; корпоративних структур, які захищають свої інформаційні ресурси; а також звичайних користувачів, що цінують особисту конфіденційність і хочуть контролювати витрати мобільного трафіку.

Актуальність дипломної роботи - полягає у необхідності створення інструментів для гнучкого та ефективного контролю доступу до Інтернету на мобільному пристрої з забезпеченням конфіденційності за допомогою локального VPN, який обробляє трафік без передачі на зовнішні сервери.

Об’єкт дослідження - засоби створення мобільних додатків на платформі Android із застосуванням сучасних технологій та мови програмування Kotlin.

Предмет дослідження - методи та алгоритми реалізації фільтрації і блокування мережевого трафіку з використанням локального VPN у середовищі Android.

Мета роботи - розробити мобільний додаток, що забезпечить ефективний і безпечний контроль доступу до Інтернету на Android-пристроях із використанням технології локального VPN.

Завдання дипломної роботи включають - дослідження існуючих методів контролю мережевого трафіку на мобільних платформах з аналізом їх переваг, недоліків та обмежень; вивчення можливостей та архітектури Android VpnService API, включаючи особливості роботи з TUN-інтерфейсами та системами дозволів. Розробку ефективних алгоритмів фільтрації мережевих пакетів на основі користувацьких налаштувань та правил доступу з оптимізацією для мінімального впливу на продуктивність. Створення сучасного та інтуїтивного користувацького інтерфейсу на мові Kotlin використовуючи технологію Material Design. Впровадження комплексної системи ведення детального журналу мережевої активності з можливостями фільтрації. Забезпечення стабільної роботи додатка в різних версіях Android з дотриманням вимог щодо енергоефективності та оптимізації батареї. Проведення комплексного тестування функціональності на різних пристроях та в різних мережевих умовах.

Практична значимість роботи полягає у створенні зручного і безпечного мобільного додатка SafeBarrier для індивідуального контролю Інтернет-доступу, що забезпечить підвищення цифрової безпеки користувачів без передачі даних на зовнішні сервери. Розроблений додаток матиме широке практичне застосування для різних категорій користувачів: батьки зможуть ефективно контролювати онлайн-активність дітей, обмежуючи доступ до неприйнятного контенту та регулюючи час використання мережі. Студенти та працівники отримають інструмент для підвищення продуктивності шляхом блокування відволікаючих додатків та веб-сайтів у робочий час.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Існуючі підходи до обмеження доступу до Інтернету на Android

Сучасні Android-пристрої пропонують кілька методів контролю мережевого трафіку, кожен з яких має свої технічні особливості, переваги та обмеження. Розвиток технологій контролю мережевого доступу на мобільних платформах відбувався поступово, від простих системних обмежень до складних рішень на базі віртуальних приватних мереж.

Класичним рішенням для Android є використання фаєрволів на рівні операційної системи з правами root. Цей підхід базується на безпосередньому управлінні вбудованими механізмами Linux ядра, зокрема системою netfilter та утилітою iptables. Програми на кшталт AFWall+ надають графічний інтерфейс до iptables і дозволяють точно налаштувати, які додатки можуть виходити в Інтернет через мобільні або Wi-Fi мережі.

AFWall+ як представник цієї категорії рішень демонструє високу ефективність та гранульований контроль. Додаток дозволяє створювати правила фільтрації на основі UID (User ID) додатків, IP-адрес, портів та типів мережевих з'єднань. Технічно AFWall+ генерує та виконує команди iptables, створюючи правила в таблицях filter, nat та mangle ядра Linux. Це забезпечує контроль трафіку на найнижчому рівні операційної системи, що гарантує максимальну ефективність та неможливість обходу обмежень додатками.

Основними перевагами root-базованих фаєрволів є:

- абсолютний контроль над мережевим трафіком на рівні ядра ОС;
- мінімальний вплив на продуктивність через обробку на рівні netfilter;
- неможливість обходу обмежень з боку додатків користувача;

- підтримка складних правил фільтрації з використанням всіх можливостей iptables.

Однак такі рішення вимагають root-доступу до пристрою та залежать від реалізації ядра Android. Root-доступ означає зняття захисних механізмів Android, що може створювати ризики безпеки та призводити до втрати гарантії виробника. Крім того, сучасні версії Android (особливо з API level 29+) активно ускладнюють отримання та використання root-прав через механізми Verified Boot, SELinux policies та інші системи захисту цілісності.

У випадках, коли root недоступний або небажаний, користувачі можуть використовувати вбудовані в Android механізми контролю трафіку. Android самостійно пропонує «режим економії даних» (Data Saver) та обмеження фонових даних для додатків, але ці механізми мають значні обмеження функціональності.

Per-app data restrictions дозволяють індивідуально налаштовувати дозволи мережевого доступу для кожного додатку через системні налаштування. Користувач може заборонити конкретному додатку використовувати мобільні дані, Wi-Fi дані, або фонові дані, але не може створювати більш складні правила фільтрації.

Digital Wellbeing та Parental Controls надають базові функції контролю часу використання та обмеження доступу до додатків, але не забезпечують детального контролю мережевого трафіку. Ці механізми працюють на рівні Activity Manager та Package Manager, блокуючи запуск додатків, а не їх мережеву активність.

Основні недоліки вбудованих механізмів:

- відсутність контролю доменів - неможливість блокування доступу до конкретних веб-сайтів;
- брак аналітики - мінімальна інформація про мережеву активність додатків;
- системні обмеження - залежність від політик Google та виробника пристрою.

1.2 Технологія локального VPN для моніторингу та фільтрації трафіку

Android-інтерфейс VpnService надає можливість створити безпечний віртуальний тунель на пристрої без необхідності в root-правах або модифікації системи. Цей API є частиною Android Framework і забезпечує controlled interface для створення VPN-додатків, що працюють в user space.

При ініціалізації VPN сервіс встановлює віртуальний мережевий інтерфейс (TUN), керує йому призначеними IP-адресами та маршрутами, а додаток отримує дескриптор цього інтерфейсу. TUN інтерфейс працює на третьому рівні моделі OSI (Network Layer) і забезпечує доступ до сирих IP-пакетів.

Ключові компоненти VpnService архітектури:

VpnService.Builder- паттерн проектування, що використовується для конфігурації VPN-з'єднання [12]. Builder дозволяє налаштувати:

- віртуальні IP-адреси та підмережі
- DNS-сервери для VPN-інтерфейсу
- MTU (Maximum Transmission Unit) розмір
- Маршрути трафіку (route tables)
- Список додатків для включення/виключення з VPN

ParcelFileDescriptor - дескриптор файлу, що представляє TUN інтерфейс [13]. Через цей дескриптор додаток може читати вихідні пакети та записувати вхідні пакети, використовуючи стандартні операції файлового вводу/виводу.

Network Namespace Isolation - Android створює ізольований мережевий простір для VPN-сервісу, що запобігає циклічному перенаправленню трафіку та забезпечує стабільність роботи [14].

Через TUN інтерфейс програма може читати вихідні пакети (які операційна система надсилає у Інтернет) та записувати вхідні пакети (які повертаються з мережі),

ніби вона безпосередньо працювала з мережею. Таким чином увесь інтернет-трафік пристрою проходить через VPN-додаток. Коли будь-який застосунок на пристрої намагається встановити з'єднання або відправити запит, Android перенаправляє цей трафік у віртуальний інтерфейс, де VPN-програма читає сирі IP-пакети. На цьому етапі пакети парсяться - з них витягується службова інформація: адреса відправника та отримувача, номери портів, тип протоколу (TCP, UDP, ICMP), розмір та вміст. На основі отриманих метаданих і заздалегідь заданих правил фільтрації додаток приймає рішення про подальші дії: трафік може бути дозволено, заблоковано, змінено або перенаправлено. Якщо з'єднання не порушує правил, пакети передаються на справжній мережевий інтерфейс (наприклад, через сокети), після чого запит доходить до сервера призначення. Відповіді з Інтернету надходять назад через цей же інтерфейс, VPN-додаток приймає вхідні пакети, обробляє їх (знову застосовуючи фільтрацію або логування, якщо потрібно) і повертає в TUN інтерфейс, з якого операційна система доставляє їх до програми-ініціатора. Таким чином VPN-додаток контролює повний цикл проходження мережевого трафіку: від моменту ініціації до моменту доставки, дозволяючи фільтрувати як IP-пакети, так і DNS-запити згідно з логікою безпеки [6].

1.3 Огляд інструменті.в, які реалізують подібну функціональність

Існує кілька популярних застосунків, що використовують локальний VPN або інші технології для блокування трафіку. Аналіз цих рішень дозволяє виявити найкращі практики та недоліки, які варто врахувати при розробці нового додатку.

1.3.1 Blokada - універсальний блокувальник реклами

Blokada (рис.1.1) популярний безкоштовний і відкритий блокувальник реклами та трекерів для Android і iOS, що розвивається з 2017 року та вирізняється простотою у використанні й ефективністю фільтрації. Застосунок працює на базі локального VPN-інтерфейсу, фільтруючи DNS-запити на основі попередньо сформованих списків

доменів рекламних мереж і трекерів [15]. Такий підхід дозволяє блокувати рекламу у всіх додатках і браузерах без потреби в root-доступі.

Однак, попри зручність і кросплатформенність, Blokada має низку обмежень. Оскільки він покладається виключно на DNS-фільтрацію, можливості контролю трафіку є досить обмеженими - застосунок не може обробляти IP-пакети повністю. Це спрощує архітектуру і знижує енергоспоживання, але разом з тим зменшує ефективність фільтрації у складніших сценаріях.

Крім того, можливості налаштування в Blokada доволі обмежені: користувачі не можуть створювати гнучкі власні правила фільтрації. Основний акцент зроблено саме на блокування реклами, тому застосунок не надає повного контролю над усім мережевим трафіком. Також DNS-фільтрація не порушує шифрування, але через це не дозволяє аналізувати вміст HTTPS-запитів, що може призвести до пропуску частини небажаного трафіку.

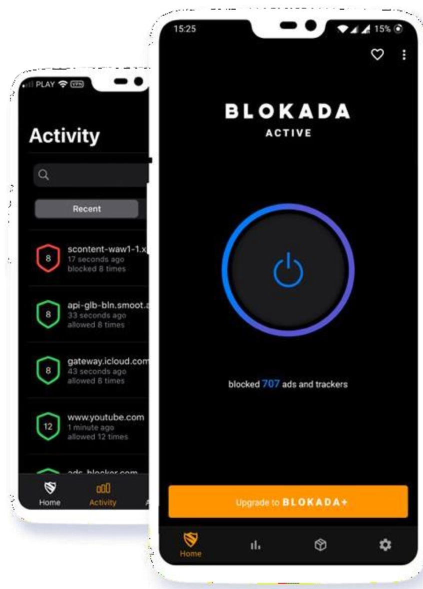


Рисунок 1.1 - Застосунок Blokada

1.3.2 Intra - DNS-захист від Google Jigsaw

Intra це мобільний застосунок від Google/Jigsaw, створений для захисту користувачів від DNS-цензури та атак на рівні DNS. Він є частиною ініціатив Google, спрямованих на покращення інтернет-безпеки та захист свободи доступу до інформації.

З технічного боку, Intra працює через локальний VPN, який шифрує DNS-запити користувача і пересилає їх на публічні DNS-сервери за протоколом DNS-over-HTTPS (DoH). Такий механізм дозволяє уникнути підміни DNS-записів і забезпечує захист від поширених атак, таких як DNS spoofing чи cache poisoning.

Проте, попри свою ефективність у вирішенні конкретного завдання, функціональність Intra доволі вузька. Застосунок не виконує фільтрацію або моніторинг іншого мережевого трафіку, крім DNS-запитів. Це означає, що він не може блокувати рекламу, трекери або забезпечувати детальний контроль над усім інтернет-з’єднанням користувача.

Також Intra не дозволяє використовувати власні списки DNS або змінювати правила фільтрації, що обмежує його адаптивність до різних умов використання. Крім того, хоча застосунок і є легким та простим, залежність від зовнішніх DNS-серверів (Google або Cloudflare) викликає питання щодо повної конфіденційності, особливо для користувачів, які прагнуть повного локального контролю над даними.

Intra добре підходить для обходу DNS-цензури та захисту DNS-запитів, але його функціональність обмежена лише цією сферою, що робить його менш універсальним у порівнянні з більш гнучкими рішеннями.

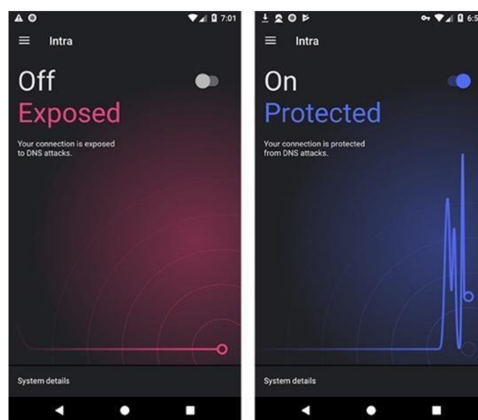


Рисунок 1.2 - Застосунок Intra

1.4 Порівняльний аналіз

Проведений огляд існуючих підходів до контролю мережевого трафіку на Android дозволяє виділити ключові переваги та обмеження кожного методу.

Root-базовані рішення забезпечують найвищий рівень контролю завдяки роботі з ядром Linux та netfilter, гарантуючи абсолютну ефективність фільтрації. Критичним недоліком є необхідність root-доступу, що обмежує аудиторію користувачів, створює ризики безпеки та може призвести до втрати гарантії пристрою. Сучасні версії Android активно протидіють root-доступу через Verified Boot та SELinux.

Вбудовані механізми Android характеризуються простотою використання та повною інтеграцією в ОС, але мають значні обмеження: відсутність фільтрації за доменами, мінімальні аналітичні можливості та залежність від політик виробника роблять їх недостатніми для повноцінного контролю мережевої активності.

Технологія локального VPN представляє найбільш збалансований підхід, поєднуючи переваги попередніх методів при мінімізації недоліків. VpnService API дозволяє досягти глибокого контролю над трафіком без root-прав, забезпечуючи доступ до IP-пакетів та повний контроль їх обробки. Підхід забезпечує максимальну сумісність з Android та не порушує безпекових механізмів ОС.

Аналіз існуючих VPN-рішень показує специфічні обмеження: Blokada фокусується на блокуванні реклами через DNS-фільтрацію, Intra обмежується захистом DNS-запитів. Жоден інструмент не забезпечує повноцінного контролю над усім спектром мережевого трафіку з гнучким налаштуванням правил.

Для розробки комплексного рішення найбільш доцільним є використання технології локального VPN на базі VpnService API.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Формування дерева проблем та дерева цілей

Аналіз проблем за допомогою дерева проблем є дієвий метод для виявлення ключових бар'єрів і труднощів, що виникають при розробці та експлуатації Android VPN-сервісу. Такий підхід дозволяє візуалізувати причинно-наслідкові зв'язки між головною проблемою, її джерелами та наслідками. Це сприяє глибшому розумінню суті проблематики, структуризації завдань та визначенню пріоритетів для подальшого вирішення. У цьому випадку проблемне дерево на рисунку 2.1 розкриває основні виклики, пов'язані з управлінням VPN-з'єднаннями, обробкою трафіку, безпекою та сумісністю з різними версіями Android.

Ключова проблема: Високий рівень складності керування мережевим трафіком та забезпечення безпеки в Android VPN-додатку.

Проблема 1: Ускладнене управління VPN-з'єднаннями

Труднощі з установкою, підтримкою та відновленням VPN-тунелю:

- **Причина 1.1,** часті перебої та нестабільність мережевого з'єднання;
- **Причина 1.2,** складність роботи з різними мережевими типами (Wi-Fi, мобільна мережа, Ethernet);
- **Причина 1.3,** Непослідовність у керуванні життєвим циклом VPN-служби.

Проблема 2: Неефективна обробка великого трафіку

Проблеми з продуктивністю при фільтрації та обробці даних:

- **Причина 2.1,** Відсутність оптимізованих алгоритмів фільтрації;
- **Причина 2.2,** Завантаження основного потоку при зборі логів і статистики;
- **Причина 2.3,** Недостатня оптимізація пам'яті при роботі з великими наборами правил.

Проблема 3: Складність конфігурації фільтрації трафіку

Важко керувати доступом додатків до мережі через неінтуїтивний процес налаштування:

- **Причина 3.1,** Відсутність зручного користувацького інтерфейсу для правил;
- **Причина 3.2,** Ускладнене управління DNS-фільтрацією;
- **Причина 3.3,** Відсутність автоматичного виявлення загроз.

Проблема 4: Проблеми сумісності з різними Android-версіями

Нестабільність додатку на різних версіях операційної системи:

- **Причина 4.1:** Часті зміни в API Android;
- **Причина 4.2:** Відмінності в механізмах надання дозволів;
- **Причина 4.3:** Обмеження у використанні foreground-сервісів на новіших версіях Android.

Проблема 5: Недостатній рівень безпеки та конфіденційності

Загроза витоку даних та небажаного доступу:

- **Причина 5.1,** Відсутність ефективного шифрування трафіку;
- **Причина 5.2,** Ненадійна валідація мережевих сертифікатів;
- **Причина 5.3,** Обмежений контроль над доступом до журналів і аналітики.

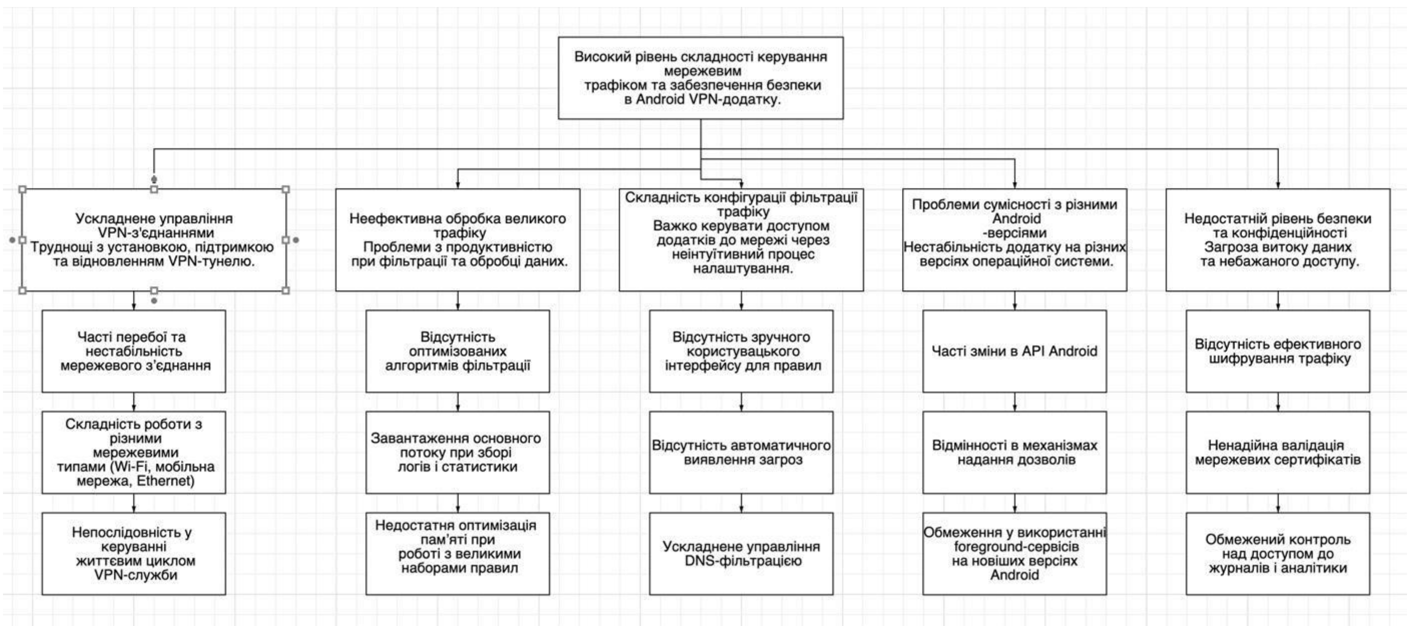


Рисунок 2.1 – Дерево проблем

Дерево цілей

Дерево цілей на рисунку 2.2 відображає зворотну сторону дерева проблем, перетворюючи кожну з виявлених проблем на відповідну ціль. Такий підхід допомагає чітко окреслити шляхи вирішення і формує стратегічний план розробки ефективного Android VPN-сервісу.

Головна ціль: Створення надійного, продуктивного та безпечного Android VPN-додатку з простим і зрозумілим керуванням.

Ціль 1: Забезпечення стабільної роботи VPN-з’єднань

Покращення механізмів створення, підтримки та відновлення тунелю:

- **Завдання 1.1,** Впровадити автоматичне відновлення при розриві з’єднання;
- **Завдання 1.2,** Розробити адаптивну логіку для роботи з різними типами мереж;
- **Завдання 1.3,** Оптимізувати життєвий цикл служби для стабільної роботи у фоновому режимі.

Ціль 2: Оптимізація обробки трафіку

Зниження затримок і підвищення ефективності фільтрації:

- **Завдання 2.1,** Створити високопродуктивні алгоритми фільтрації з використанням нативного коду;
- **Завдання 2.2,** Впровадити асинхронну обробку журналів і статистичних даних;
- **Завдання 2.3,** Реалізувати кешування та зменшення витрат пам’яті при обробці правил.

Ціль 3: Спрощення управління фільтрацією

Надання зручних засобів для налаштування правил доступу:

- **Завдання 3.1,** Розробити інтуїтивний інтерфейс для створення та редагування правил;
- **Завдання 3.2,** Автоматизувати процес DNS-фільтрації та оновлення блокувальних списків;
- **Завдання 3.3,** Впровадити механізми автоматичного виявлення і блокування загроз.

Ціль 4: Забезпечення широкої сумісності з Android

Гарантія стабільної роботи на більшості пристроїв:

- **Завдання 4.1,** Впровадити адаптивні рішення для роботи з різними версіями Android API;
- **Завдання 4.2,** Створити універсальну систему обробки дозволів;
- **Завдання 4.3,** Оптимізувати функціонування foreground-сервісів під вимоги нових версій ОС.

Ціль 5: Підвищення рівня безпеки та захисту конфіденційності

Захист персональних даних користувачів та мережевого трафіку:

- **Завдання 5.1,** Реалізувати повне шифрування трафіку;
- **Завдання 5.2,** Впровадити перевірку сертифікатів і механізми захисту від атак типу "людина посередині" (MITM);
- **Завдання 5.3,** Створити механізм обмеження доступу до логів і забезпечити їх анонімізацію.

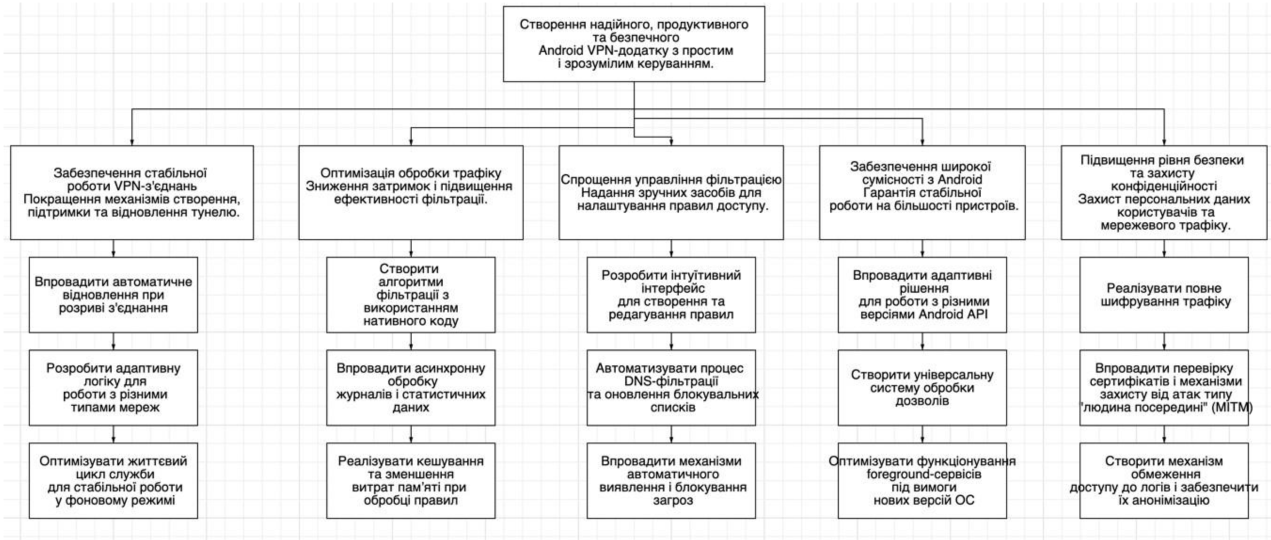


Рисунок 2.2 – Дерево цілей

2.2 Об’єкти дослідження та предметна область

Розробка мобільного додатка SafeBarrier для блокування доступу до інтернету потребує детального аналізу об’єктів предметної області та встановлення зв’язків між ними. Основними об’єктами дослідження в рамках даної системи є користувач мобільного пристрою, застосунок SafeBarrier, мережеві запити, служби операційної системи Android, правила фільтрації та дані журналування.

Користувач мобільного пристрою представляє собою кінцевий об’єкт взаємодії з системою, який виконує налаштування параметрів блокування через користувацький інтерфейс. Цей об’єкт характеризується можливістю управління правилами блокування для окремих застосунків, перегляду історії мережевої активності та контролю стану VPN-з’єднання. Користувач взаємодіє з системою через компоненти MainActivity та відповідні ViewModel класи, які забезпечують реактивне оновлення інтерфейсу при зміні стану системи.

Застосунок SafeBarrier являє собою комплексну програмну систему, побудовану на принципах MVVM архітектури з використанням патерну Repository для управління даними. Основними компонентами застосунку є користувацький інтерфейс, реалізований через MainActivity та фрагменти, система ViewModels для управління станом додатка, VPN сервіс ServiceFirewall для обробки мережевого трафіку та модуль DatabaseHelper для зберігання конфігурацій та логів [9].

Мережеві запити представляють потік IP-пакетів, які проходять через створений локальний VPN-інтерфейс і підлягають аналізу та фільтрації. Ці запити включають TCP/UDP з’єднання від встановлених на пристрої застосунків, DNS-запити для резолюції доменних імен, ICMP повідомлення для діагностики мережі та метадані з’єднань, що містять інформацію про IP-адреси, порти та протоколи [11].

Служби операційної системи Android формують інфраструктурний рівень системи та включають VpnService як базовий клас для реалізації VPN-функціональності, ConnectivityManager для моніторингу стану мережевих з’єднань,

PackageManager для отримання інформації про встановлені застосунки та TelephonyManager для відстеження стану телефонних з'єднань.

Правила фільтрації та блокування містять конфігураційні дані у вигляді об'єктів Rule, які визначають поведінку системи для кожного встановленого застосунку. Ці правила зберігають налаштування для WiFi та мобільної мережі, параметри блокування при заблокованому екрані, налаштування для роумінгу та режиму lockdown. Правила зберігаються як у SharedPreferences для швидкого доступу, так і в SQLite базі даних для складніших запитів.

Дані журналу мережевої активності представляють інформаційні об'єкти, що фіксують всі події обробки пакетів системою. Ці дані включають логи пакетів через об'єкти Packet, статистику використання мережі через Usage об'єкти, записи про DNS резолюцію та метрики продуктивності VPN сервісу. Журналування реалізується асинхронно через LogHandler для мінімізації впливу на продуктивність [7].

2.3 Інформаційна модель системи

Інформаційна модель системи зображена на рисунку 2.3 представляє структуру даних та взаємозв'язки між компонентами системи у вигляді концептуальної схеми та діаграм сутність-зв'язок. Модель описує як статичні аспекти структури даних, так і динамічні аспекти взаємодії компонентів.

Концептуальна архітектура системи базується на чотирирівневій структурі, де кожен рівень має чітко визначені обов'язки та інтерфейси взаємодії. Презентаційний рівень включає компоненти MainActivity, фрагменти користувацького інтерфейсу та адаптери для відображення списків даних. Цей рівень відповідає за обробку користувацького вводу та відображення стану системи.

Рівень ViewModel містить класи MainViewModel, AppsViewModel та LogViewModel, які служать посередниками між презентаційним рівнем та бізнес-логікою. Ці компоненти управляють станом користувацького інтерфейсу, обробляють

команди користувача та забезпечують реактивне оновлення даних через LiveData механізм.

Доменний рівень представлений Use Cases та Repository класами, які інкапсулюють бізнес-логіку застосунку. Цей рівень визначає правила обробки VPN з'єднань, алгоритми фільтрації пакетів та політики логування мережевої активності.

Сервісний рівень включає ServiceFirewall як основний VPN сервіс та його внутрішні компоненти CommandHandler, LogHandler та StatsHandler. Ці компоненти забезпечують низькорівневу обробку мережевого трафіку та взаємодію з операційною системою Android.

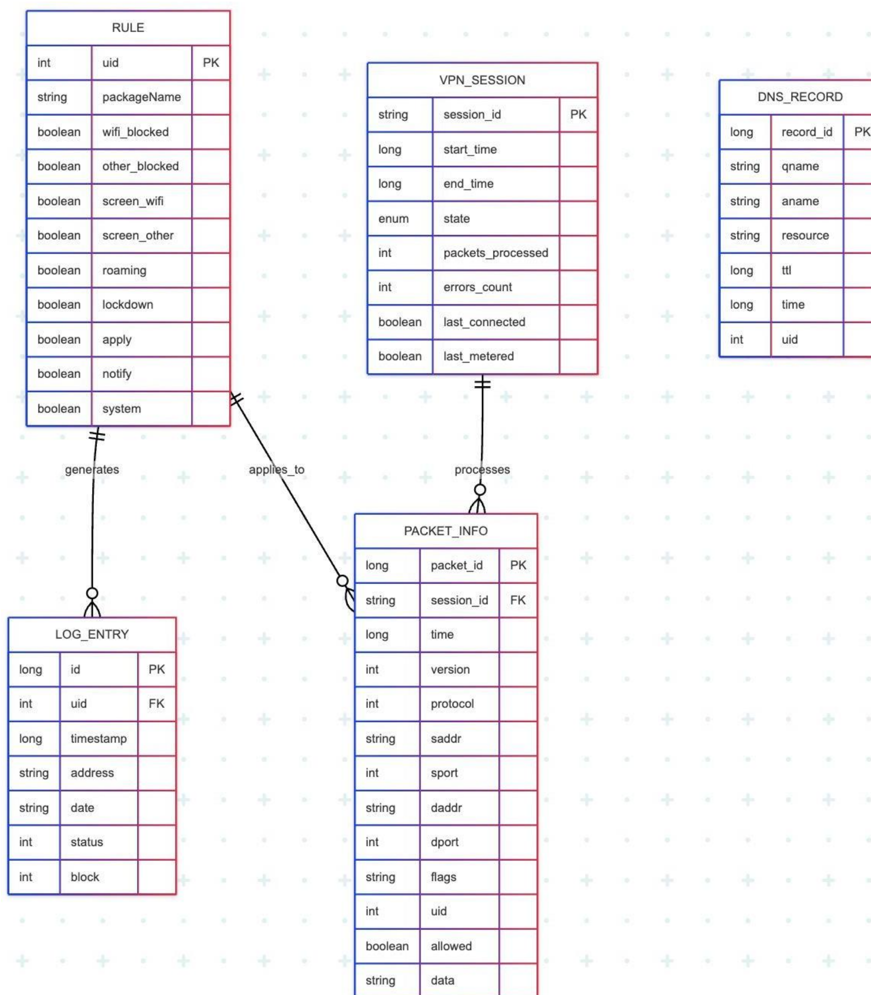


Рисунок 2.3 - Інформаційна модель програмного забезпечення

Основні зв’язки між сутностями визначаються наступним чином: сутність RULE знаходиться у відношенні "один до багатьох" з LOG_ENTRY, оскільки одне правило фільтрації може породжувати множину записів у журналі. VPN_SESSION пов’язана з PACKET_INFO також у відношенні "один до багатьох", де одна сесія VPN може обробляти тисячі пакетів. DNS_RECORD існує як незалежна сутність для кешування результатів DNS запитів.

2.4 Обмеження на вхідні та вихідні дані

Система функціонує в рамках певних обмежень, які визначаються як технічними характеристиками платформи Android, так і архітектурними рішеннями застосунку. Вхідні дані системи включають налаштування користувача, список встановлених застосунків та мережеві пакети.

Налаштування користувача приймаються у вигляді булевих значень для параметрів wifi_blocked та other_blocked, які визначають поведінку блокування для WiFi та мобільних мереж відповідно. Ці дані валідуються на предмет коректності package name та зберігаються в SharedPreferences з ключами "wifi", "other", "apply" тощо. Система підтримує також складні налаштування, такі як screen_wifi та screen_other для поведінки при заблокованому екрані.

Список встановлених застосунків отримується через PackageManager.getInstalledPackages() та містить об’єкти ApplicationInfo з метаданими. Система автоматично фільтрує системні застосунки залежно від налаштувань користувача та оновлюється при подіях install/uninstall через BroadcastReceiver.

Мережеві пакети надходять через TUN інтерфейс у форматі стандартних IP пакетів з полями version (4 або 6), protocol (6 для TCP, 17 для UDP), адресами джерела та призначення, портами та додатковими метаданими. Кожен пакет асоціюється з UID застосунку, що його згенерував, для подальшого застосування правил фільтрації.

Вихідні дані системи представлені станом VPN-з’єднання, звітами про блокування та метриками продуктивності. Стан VPN описується енумом StartState з значеннями READY, CONNECTING та CONNECTED, що відображає поточний етап життєвого циклу з’єднання.

Звіти про блокування генеруються у форматі LogEntry об’єктів, які зберігаються в SQLite базі даних з автоматичною ротацією записів старше трьох днів. Система підтримує експорт даних у CSV та JSON форматах для подальшого аналізу.

Метрики продуктивності отримуються з нативного рівня через JNI інтерфейс у вигляді масиву цілих чисел, де індекси відповідають кількості оброблених пакетів, заблокованих з’єднань, переданих байтів та помилок з’єднання.

Технічні обмеження системи визначаються вимогами платформи Android та включають мінімальну версію API 21 (Android 5.0) для забезпечення сумісності з VpnService.Builder функціональністю. Система вимагає дозволів BIND_VPN_SERVICE та FOREGROUND_SERVICE для коректного функціонування.

Функціональні обмеження включають неможливість блокування критичних системних служб Android, обмеження на один активний VPN на пристрої та принципове рішення про локальну обробку без передачі даних на зовнішні сервери для забезпечення приватності користувачів.

Ресурсні обмеження залежать від характеристик конкретного пристрою та включають обмеження на кількість правил фільтрації, період зберігання логів та розмір буферів обробки пакетів. Система адаптивно налаштовується залежно від доступної пам’яті пристрою, обмежуючи максимальну кількість правил від 100 для бюджетних пристроїв до 1000 для флагманських моделей [3].

2.5 Концептуальне проектування системи

Концептуальне проектування системи SafeBarrier базується на принципах MVVM архітектури з елементами Clean Architecture для забезпечення тестованості,

масштабованості та підтримуваності коду зображено на рис. 2.4 на рис. 2.5. Архітектура розділена на чітко визначені шари з односпрямованими залежностями та абстрактними інтерфейсами взаємодії.

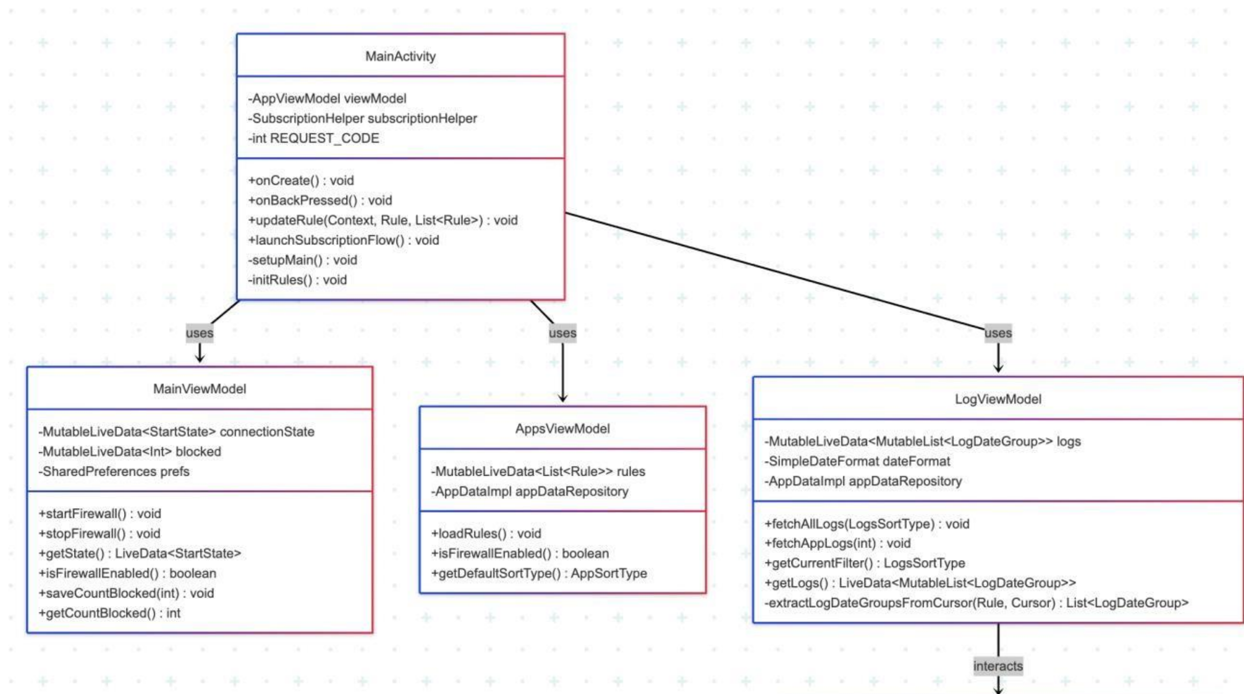


Рисунок 2.4 - Перша частина UML діаграми

Ключовими компонентами є класи ViewModel, які забезпечують зв’язок між інтерфейсом користувача та бізнес-логікою. У центрі взаємодії користувача з додатком знаходиться MainActivity, яка ініціалізує ViewModel’и та взаємодіє з ними для управління станом додатку. Зокрема, MainActivity використовує MainViewModel для запуску та зупинки фаєрволу, відслідковування стану з’єднання та збереження статистики заблокованих з’єднань. Вона також використовує AppsViewModel для завантаження правил блокування трафіку для окремих застосунків, а LogViewModel - для отримання журналів заблокованих підключень.

MainViewModel відповідає за логіку керування VPN-з’єднанням і зберігає стан фаєрволу у вигляді LiveData-об’єктів. Він також взаємодіє з локальним сховищем (SharedPreferences) для збереження даних про кількість заблокованих з’єднань.

AppsViewModel взаємодіє з AppDataImpl - класом, що виконує роль репозиторію - для отримання правил блокування (Rule). Він дозволяє фільтрувати та сортувати список програм залежно від обраного типу сортування.

LogViewModel також використовує AppDataImpl та відповідає за завантаження логів. Він оперує списками LogDateGroup, які групують журнали за датою. Для обробки сирих даних з бази даних використовується курсор, який перетворюється у структуровану модель.

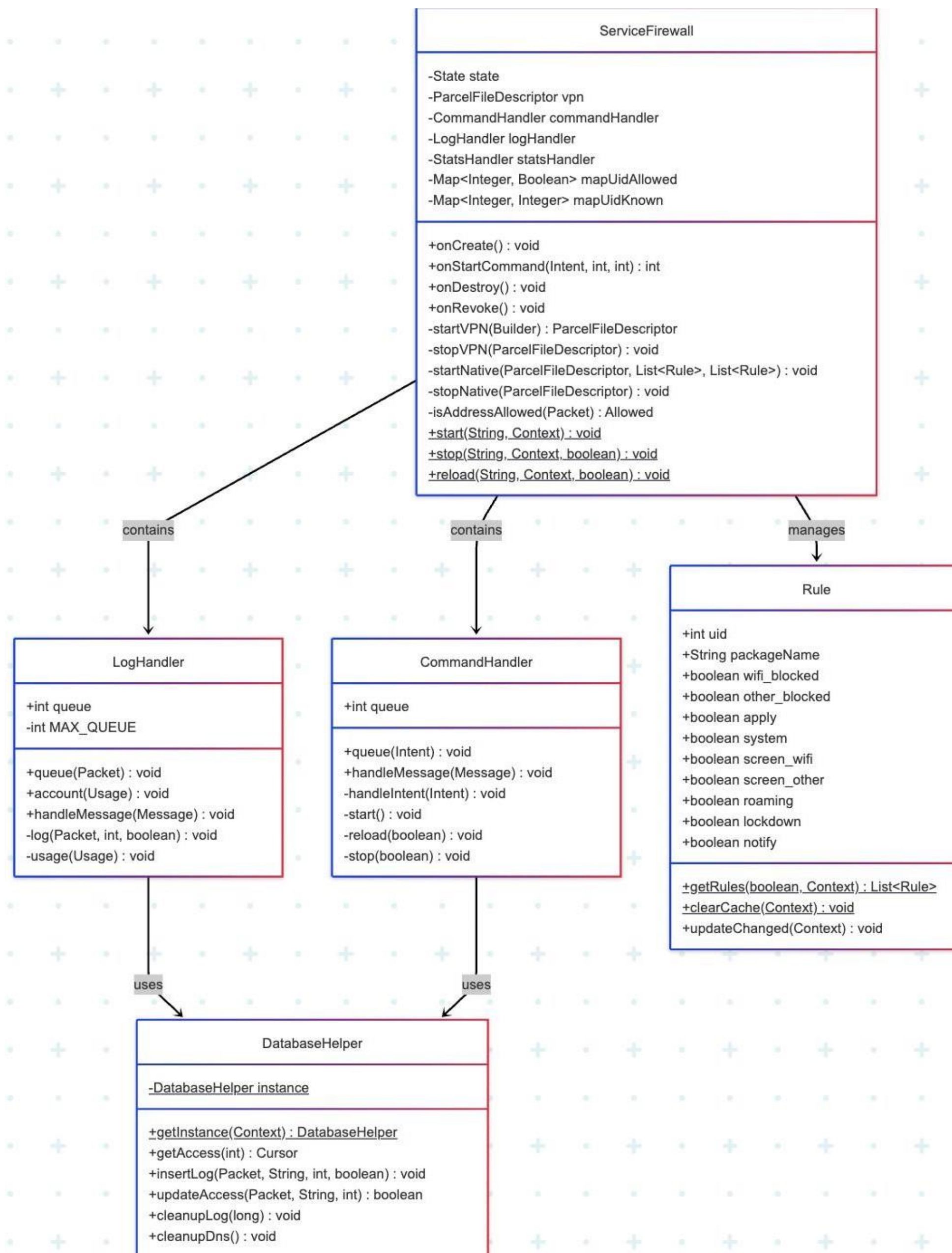


Рисунок 2.5 - Друга частина UML діаграми

Ключовим компонентом фаєрволу є сервіс ServiceFirewall, який реалізує логіку локального VPN. Він ініціалізує VPN-з'єднання, керує трафіком на низькому рівні за допомогою нативного коду, перевіряє дозволи для UID, та передає пакети у відповідні обробники.

Клас Rule є моделлю, яка представляє налаштування блокування для конкретного UID (застосунку), зокрема, чи дозволено доступ до Wi-Fi або мобільного інтернету, чи застосовується режим блокування лише при вимкненому екрані, тощо. Він також містить статичні методи для отримання правил і оновлення кешу.

Для роботи з журналами використовується DatabaseHelper, який реалізує збереження, оновлення та очищення логів. Як LogHandler, так і CommandHandler взаємодіють з DatabaseHelper для запису змін у базу даних.

Таким чином, дана діаграма класів описує повноцінну архітектуру застосунку, яка охоплює рівні UI (MainActivity), логіки (ViewModel), сервісів (ServiceFirewall) та доступу до даних (DatabaseHelper, Rule), забезпечуючи розділення відповідальностей та гнучкість розширення функціональності.

2.6 Реалізація математичних і алгоритмічних методів

Математичне забезпечення охоплює кілька ключових напрямів: класифікація мережевих пакетів, прийняття рішень на основі правил доступу, статистичний облік заблокованого трафіку, а також оптимізація продуктивності при високих навантаженнях.

Основу системи прийняття рішень становить алгоритм класифікації вхідних та вихідних мережевих пакетів згідно з визначеними правилами. Кожен пакет характеризується унікальним ідентифікатором процесу (UID), IP-адресою, портом, протоколом передачі (TCP/UDP) та типом з'єднання (Wi-Fi або мобільна мережа). На основі цих характеристик виконується перевірка умов, заданих у правилі типу:

- заборонити доступ до Wi-Fi;

- заборонити доступ до мобільного інтернету;
- заборонити підключення при вимкненому екрані;
- увімкнути режим жорсткої ізоляції (lockdown);
- блокувати в режимі роумінгу.

Ці правила інтерпретуються як булеві умови. Таким чином, класифікація пакета реалізується у вигляді логічного виразу з булевих змінних, що відповідають умовам

Для побудови рішень про дозволи/заборони на основі множини правил використовується теорія множин. Усі пакети, що надходять до VPN-ядра, ідентифікуються за UID. Кожен UID має відповідний набір дозволених або заборонених дій, що утворює множину дозволених з’єднань: A = {UID_i | доступ дозволено}, B = {UID_j | доступ заборонено}

Під час обробки пакету перевіряється його приналежність до множини B. Якщо UID ∈ B, трафік блокується. Для оптимізації ця перевірка реалізована через хеш-мапу mapUidAllowed.

Система також реалізує модуль статистичного аналізу, який обліковує кількість заблокованих з’єднань по кожному застосунку. Для цього використовується структура даних типу словника Map<UID, Count>. При кожному блокуванні значення лічильника інкрементується. Після досягнення певного інтервалу часу або кількості подій, статистика агрегується та записується у базу даних.

У випадках великої кількості з’єднань використовується алгоритмічна оптимізація у вигляді кешування останніх рішень для UID, що часто зустрічаються. Це дозволяє уникнути повторної перевірки тих самих умов.

2.7 Використання нативного коду для фільтрації мережевого трафіку

Обробка мережевого трафіку в режимі реального часу на мобільних пристроях потребує максимальної продуктивності та мінімального споживання ресурсів.

Незважаючи на значні оптимізації віртуальної машини Android Runtime, критичні шляхи обробки пакетів вимагають використання нативного коду для досягнення необхідної пропускної здатності та зменшення затримок.

Аналіз продуктивності показує, що чиста Java реалізація здатна обробляти приблизно 5000-8000 пакетів на секунду на середньому Android пристрої, тоді як гібридна архітектура з нативним ядром досягає показників 15000-25000 пакетів на секунду. Це критично важливо для пристроїв з високим мережевим навантаженням або при використанні ресурсоємних застосунків.

Основними факторами, що обумовлюють перевагу нативного коду, є відсутність накладних витрат на garbage collection для критичних операцій, пряме управління пам’яттю без посередництва runtime heap, можливість використання SIMD інструкцій для паралельної обробки множинних пакетів та оптимізація на рівні машинних інструкцій через сучасні компілятори C++.

Архітектурне рішення системи SafeBarrier базується на принципі гібридної реалізації, де логіка управління та користувацький інтерфейс залишаються на Java рівні для забезпечення інтеграції з Android framework, а критичні операції обробки пакетів переносяться на нативний рівень для максимальної продуктивності.

Вибір мови C++ для нативної реалізації обумовлений її поєднанням високої продуктивності з відносною безпекою порівняно з чистим C. Сучасні можливості C++17 та C++20, такі як smart pointers, thread-safe контейнери та structured bindings, дозволяють писати безпечний та ефективний код для роботи з мережевими протоколами.

Java Native Interface служить мостом між Java кодом системи SafeBarrier та нативними компонентами обробки пакетів. Архітектура JNI взаємодії оптимізована для мінімізації накладних витрат на перехід між runtime середовищами та забезпечення thread-safety в багатопоточному середовищі [1].

Основним принципом архітектури є асиметричний розподіл відповідальностей, де Java сторона відповідає за управління життєвим циклом нативних ресурсів,

конфігурацію та координацію між компонентами, тоді як нативна сторона зосереджена на високопродуктивній обробці даних пакетів.

Нативний контекст представлений структурою, що інкапсулює всі ресурси, необхідні для обробки VPN трафіку. Контекст містить файловий дескриптор TUN інтерфейсу, конфігураційні параметри системи, кеші для оптимізації та статистичні лічильники продуктивності.

Ініціалізація контексту виконується атомарно для забезпечення консистентності даних у багатопоточному середовищі. Використання RAII принципів C++ гарантує автоматичне звільнення ресурсів при завершенні роботи або виникненні помилок.

Дизайн контексту передбачає можливість існування множинних ізольованих інстансів для підтримки advanced сценаріїв використання, таких як одночасна обробка трафіку різних VPN профілів або A/B тестування алгоритмів фільтрації.

Публічний JNI інтерфейс містить мінімальний набір методів для управління життєвим циклом нативних компонентів та передачі даних. Кожен метод спроектований для виконання атомарних операцій без блокування викликаючого потоку.

Метод jni_init створює та ініціалізує нативний контекст, приймаючи версію Android SDK для адаптації до специфічних особливостей різних версій платформи. Метод повертає унікальний ідентифікатор контексту для подальшого використання в інших операціях.

Методи jni_start та jni_stop управляють активацією та деактивацією обробки пакетів. Ці методи є асинхронними та повертають управління негайно, запускаючи внутрішні робочі потоки для фактичної обробки трафіку.

Метод jni_run представляє основний цикл обробки пакетів та виконується у вигляді blocking операції в окремому потоці. Цей підхід дозволяє Java коду продовжувати обробку користувацького інтерфейсу та управління системою, поки нативний код займається інтенсивною обробкою мережевих даних.

Зворотний виклик з нативного коду до Java реалізований через систему callback методів, що дозволяє повідомляти про події обробки пакетів, помилки та статистичні дані. Callback методи оптимізовані для мінімального впливу на продуктивність основного циклу обробки.

Асинхронні callback використовуються для подій, що не потребують негайної обробки, таких як статистичні оновлення або логування неблокуючих подій. Ці виклики ставляться у чергу та обробляються пакетно для зменшення накладних витрат.

Синхронні callback застосовуються для критичних рішень, таких як перевірка дозволів для пакетів або резолюція DNS запитів. Ці операції вимагають негайної відповіді для продовження обробки пакету.

Ефективне управління пам’яттю є критичним аспектом нативної реалізації для забезпечення стабільності та продуктивності системи. Архітектура управління пам’яттю базується на принципах детерміністичного allocation, object pooling та мінімізації memory fragmentation.

Object pool реалізований як template клас C++, що дозволяє створювати type-safe пули для різних типів об’єктів. Пул використовує lock-free алгоритми для allocation та deallocation операцій у багатопоточному середовищі.

Pre-allocation стратегія виділяє всі необхідні об’єкти на етапі ініціалізації системи для уникнення dynamic allocation під час обробки пакетів. Це забезпечує детермінований час виконання критичних операцій.

Memory alignment оптимізації вирівнюють об’єкти по cache line boundaries для мінімізації false sharing між потоками. На сучасних ARM архітектурах cache line size становить 64 байти, що враховується при розрахунку layout об’єктів.

Reference counting автоматично відстежує використання об’єктів та повертає їх у пул після завершення обробки. Weak references використовуються для додаткових посилань без впливу на lifetime управління.

Великі структури даних, такі як таблиці маршрутизації або кеші DNS, реалізовані через memory mapping для ефективного використання virtual memory. Це дозволяє операційній системі управляти paging та caching на оптимальному рівні.

Anonymous mapping використовується для runtime структур даних, що не потребують persistent storage. File-backed mapping застосовується для конфігураційних даних, що можуть бути збережені між сеансами роботи.

Copy-on-write семантика дозволяє ефективно клонувати великі структури даних для read-only доступу з множинних потоків. Модифікації автоматично створюють приватні копії без впливу на інші потоки.

Memory advise системні виклики оптимізують поведінку virtual memory manager для специфічних паттернів доступу до даних. Sequential access patterns використовують MADV_SEQUENTIAL, тоді як random access застосовує MADV_RANDOM.

Resource Acquisition Is Initialization принципи забезпечують автоматичне управління ресурсами через конструктори та деструктори C++ об'єктів. Це гарантує звільнення ресурсів навіть при виникненні exceptional situations.

Smart pointers (unique_ptr, shared_ptr) використовуються для автоматичного management memory та інших ресурсів. Custom deleters дозволяють інтегрувати object pools та інші спеціалізовані механізми управління.

Exception safety guarantees забезпечують consistent state системи при виникненні помилок. Strong exception safety використовується для критичних операцій, де відкат до попереднього стану є обов'язковим.

RAII wrappers створені для всіх system resources, таких як file descriptors, mutex objects та thread handles. Це забезпечує automatic cleanup навіть при unexpected termination процесу.

Взаємодія з Java garbage collector мінімізована через обмеження кількості JNI object allocations та використання direct memory access. Local references звільняються явно після завершення використання [2].

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Структура проекту та середовище розробки

Розробка мобільного фаєрволу SafeBarrier здійснювалася в інтегрованому середовищі Android Studio (рис. 3.1), яке забезпечує повний набір інструментів для створення Android-додатків. Проєкт організований за стандартною структурою Android-розробки з використанням системи збірки Gradle та підтримкою як Java і Kotlin, так і нативного C/C++ коду.

Основна логіка взаємодії з мережею реалізована на Java, тоді як Kotlin використовується для створення користувацького інтерфейсу.

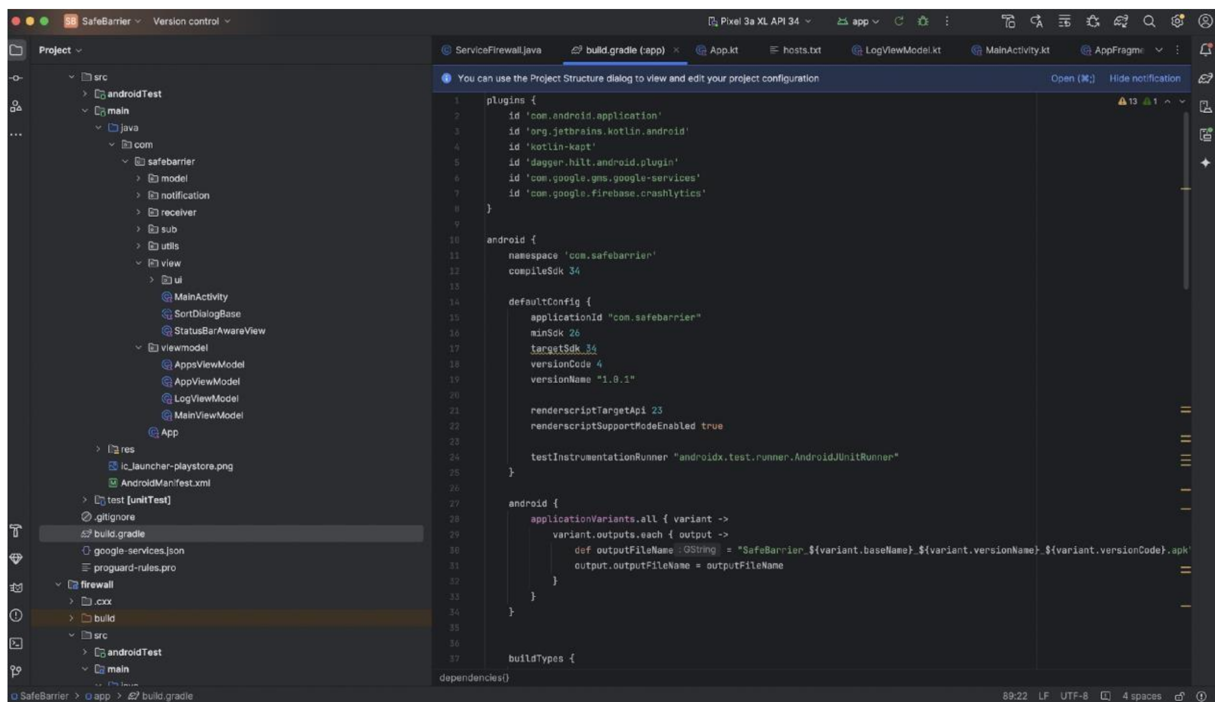


Рисунок 3.1 - Друга частина UML діаграми

Структура проекту включає декілька основних директорій: app для основного коду додатку, libs для зовнішніх бібліотек, build для скомпільованих файлів та src для

вихідного коду. Така організація забезпечує логічне розділення різних типів ресурсів та полегшує процес розробки й підтримки проекту.

Система збірки проекту базується на Gradle, що забезпечує автоматичне управління залежностями та гнучкі можливості конфігурації. Файл build.gradle (рис. 3.2) містить всі критично важливі налаштування проекту, включаючи плагіни, версії SDK та параметри компіляції.

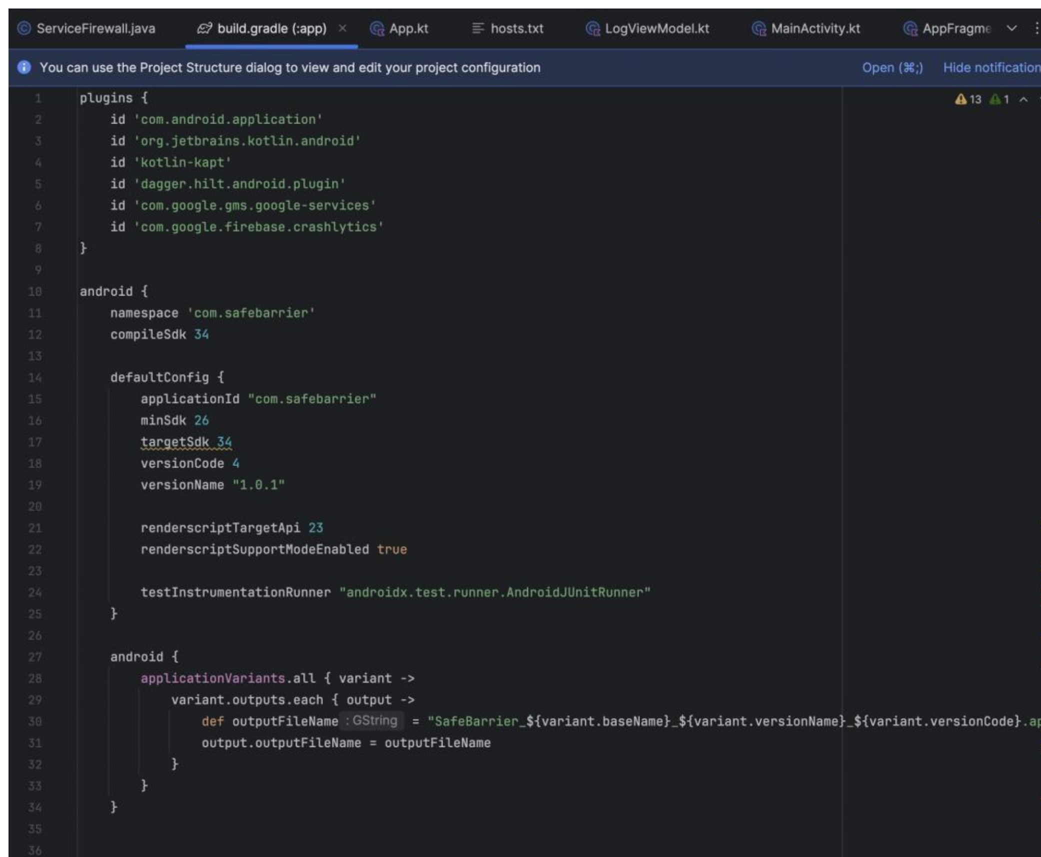


Рисунок 3.2 - Файл build.gradle

Конфігурація включає підтримку сучасних Android API (compileSdk 34), мінімальну версію для забезпечення сумісності (minSdk 26) та цільову версію для оптимізації функціональності. Використання множинних плагінів забезпечує інтеграцію з Kotlin, Dagger Hilt для dependency injection, Google Services та Firebase для аналітики та crash reporting.

Java компоненти організовані в пакеті com.net.firewall (рис. 3.3), який містить логічно структуровані класи для різних аспектів функціональності фаєрволу. Основні компоненти включають модельні класи для представлення даних, сервісні класи для бізнес-логіки та утилітарні класи для допоміжних функцій.

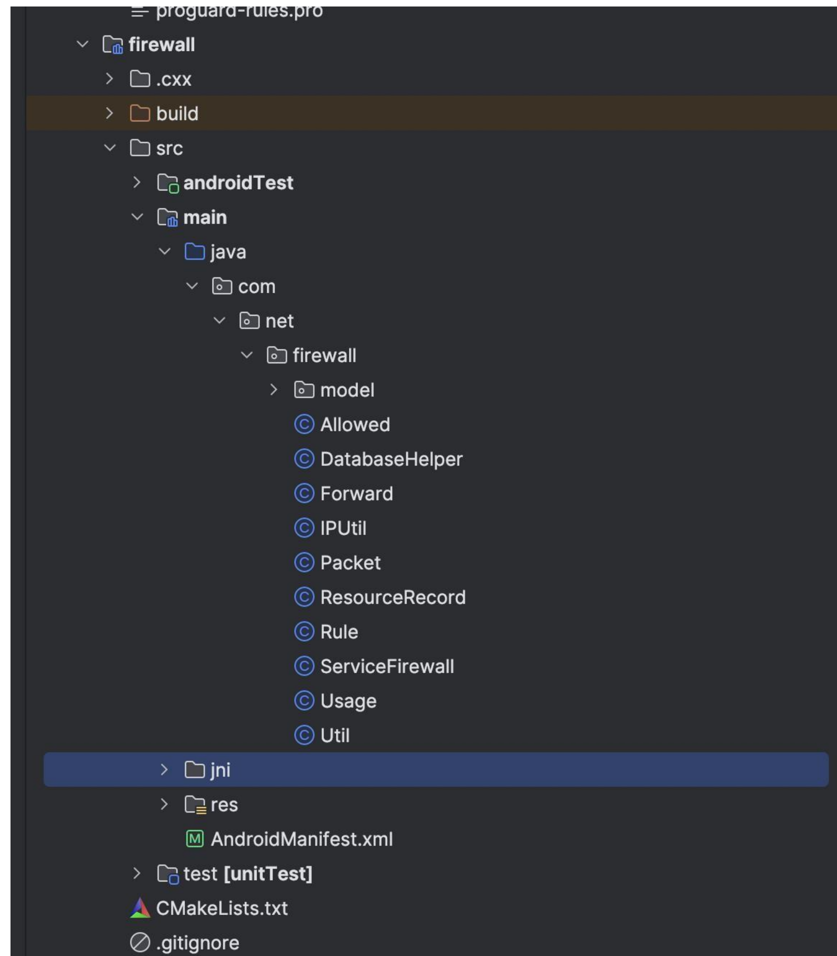


Рисунок 3.3 - Організація Java класів та нативних компонентів у проекті

Java компоненти організовані в пакеті com.net.firewall, який містить логічно структуровані класи для різних аспектів функціональності фаєрволу. Основні компоненти включають модельні класи для представлення даних, сервісні класи для бізнес-логіки та утилітарні класи для допоміжних функцій.

Клас ServiceFirewall є центральним компонентом архітектури, який розширює VpnService та реалізує основну функціональність перехоплення та фільтрації

мережевого трафіку. Клас DatabaseHelper забезпечує роботу з локальною базою даних SQLite для зберігання правил фільтрації та журналів активності. Клас Rule представляє правила фільтрації для окремих додатків, а Packet інкапсулює інформацію про мережеві пакети.

Директорія res зображена на рисунку 3.4 містить всі ресурси додатку, включаючи макети інтерфейсу користувача, графічні ресурси, рядки для локалізації та файли конфігурації. Структура ресурсів організована за типами та призначенням: layout для XML макетів екранів, drawable для векторних та растрових зображень, values для рядків, кольорів та розмірів, menu для меню додатку та xml для різноманітних конфігураційних файлів.

Макети інтерфейсу в директорії layout містять опис екранів додатку з використанням ConstraintLayout та LinearLayout для забезпечення адаптивності на різних розмірах екранів. Кожен макет оптимізований для швидкого завантаження та ефективного використання пам’яті, що критично важливо для системного додатку, який постійно працює у фоні.

Графічні ресурси включають іконки додатку різних розмірів для підтримки різних щільностей екранів (mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi), векторні drawable для масштабованих елементів інтерфейсу та спеціальні індикатори стану фаєрволу. Використання векторної графіки дозволяє забезпечити чіткість зображень на всіх типах дисплеїв при мінімальному розмірі APK файлу.

Файли локалізації в директорії values містять рядки українською та англійською мовами. Кольорова схема додатку визначена в colors.xml з урахуванням принципів Material Design. Розміри та відступи стандартизовані в dimens.xml.

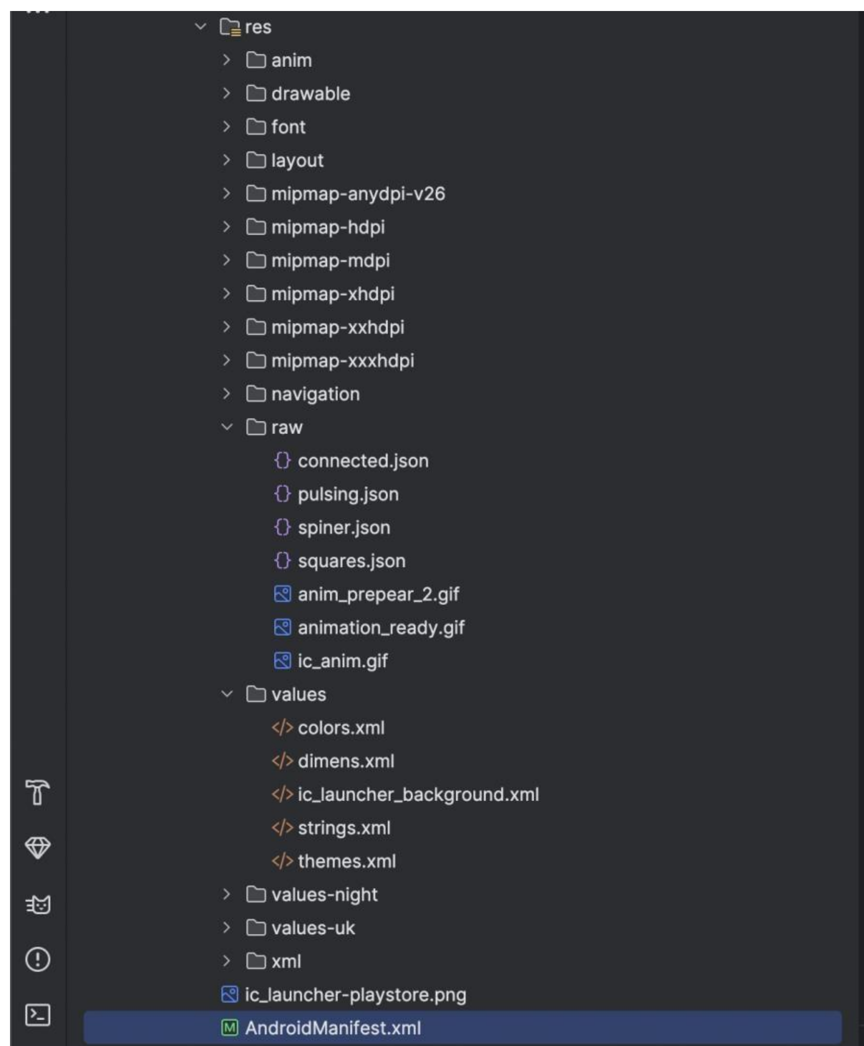


Рисунок 3.4 - Директорія res застосунку

Файл AndroidManifest.xml на рис. 3.5 визначає основні характеристики додатку, необхідні дозволи та компоненти системи.

Маніфест також містить декларації BroadcastReceiver для обробки системних подій, Activity компонентів для користувацького інтерфейсу та провайдерів для обміну даними з іншими додатками. Правильна конфігурація дозволів забезпечує функціональність фаєрволу при дотриманні принципів безпеки Android.

```
ServiceFirewall.java   build.gradle (:app)   AndroidManifest.xml ×   App.kt   LogViewModel.kt
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
23      <uses-permission
24          android:name="android.permission.QUERY_ALL_PACKAGES"
25          tools:ignore="QueryAllPackagesPermission" />
26
27      <application
28          android:name=".App"
29          android:allowBackup="true"
30          android:dataExtractionRules="@xml/data_extraction_rules"
31          android:fullBackupContent="@xml/backup_rules"
32          android:icon="@mipmap/ic_launcher"
33          android:label="SafeBarrier"
34          android:roundIcon="@mipmap/ic_launcher_round"
35          android:supportsRtl="true"
36          android:theme="@style/Theme.SafeBarrier"
37          tools:targetApi="31" >
38
39          <activity
40              android:name=".view.MainActivity"
41              android:screenOrientation="portrait"
42              android:exported="true" >
43              <intent-filter>
44                  <action android:name="android.intent.action.MAIN" />
45                  <category android:name="android.intent.category.LAUNCHER" />
46              </intent-filter>
47          </activity>
48
49          <service
50              android:name="com.net.firewall.ServiceFirewall"
51              android:exported="true"
52              android:foregroundServiceType="dataSync"
53              android:label="SafeBarrier"
54              android:permission="android.permission.BIND_VPN_SERVICE">
55              <intent-filter>
56                  <action android:name="android.net.VpnService" />
57              </intent-filter>
58          </service>
59
```

Рисунок 3.5 - Фрагмент коду ServiceFirewall в AndroidManifest.xml

Файл AndroidManifest.xml містить список необхідних дозволів для роботи фаєрволу. Кожен дозвіл надає доступ до певних функцій Android системи.

Мережеві дозволи:

- INTERNET - доступ до інтернету
- ACCESS_NETWORK_STATE - інформація про стан мережі
- ACCESS_WIFI_STATE - інформація про Wi-Fi підключення

Системна інформація:

- READ_PHONE_STATE - читання стану телефону для розрізнення типів з’єднань
- QUERY_ALL_PACKAGES - отримання списку всіх встановлених додатків

Фонова робота:

- FOREGROUND_SERVICE - запуск сервісів переднього плану
- FOREGROUND_SERVICE_DATA_SYNC - синхронізація даних
- WAKE_LOCK - утримання процесора в активному стані

Автозапуск:

- RECEIVE_BOOT_COMPLETED - автозапуск після перезавантаження
- QUICKBOOT_POWERON - сумісність з швидким запуском деяких пристроїв

Інтерфейс користувача:

- POST_NOTIFICATIONS - показ системних сповіщень
- VIBRATE - вібрація для сповіщень

Дозволи забезпечують повноцінну роботу фаєрволу: перехоплення мережевого трафіку, контроль додатків, фонову роботу та взаємодію з користувачем через сповіщення.

Для тестування і відлагодження було використано вбудований емулятор Android, який є частиною Android Studio. Емулятор дозволив перевірити роботу додатку на різних версіях операційної системи Android (від Android 8 до Android 14), а також на різних типах пристроїв - смартфонах, планшетах і пристроях з різною роздільною здатністю екрана.

За допомогою емулятора була змодельована робота додатку в умовах, максимально наближених до реальних: зміна мережевого з’єднання, перемикання між Wi-Fi та мобільною мережею, емуляція повільного інтернету, розрядженої батареї, призупинення додатку тощо. Це дозволило перевірити стабільність роботи мережевого фаєрволу в різних сценаріях використання.

Крім того, емульований пристрій підтримує налаштування рівня доступу до системних сервісів, що було корисно для перевірки коректної обробки дозволів (Permissions) і поведінки додатку при їх зміні.

Тестування на емуляторі дало змогу виявити і усунути помилки ще до запуску на реальних пристроях, що значно пришвидшило процес розробки та підвищило загальну якість рішення.

3.2 Функціонал розробленого додатку

При відкриті додатку користувач зразу попадає в екран додатку (рис. 3.6-3.7) реалізує дворежимну систему управління станом фаєрволу через елегантний кільцевий інтерфейс. У активному режимі концентричні кільця заповнюються помаранчевим градієнтом, візуально демонструючи рівень захисту системи. Центральна іконка живлення змінює свій стан відповідно до поточного режиму роботи.

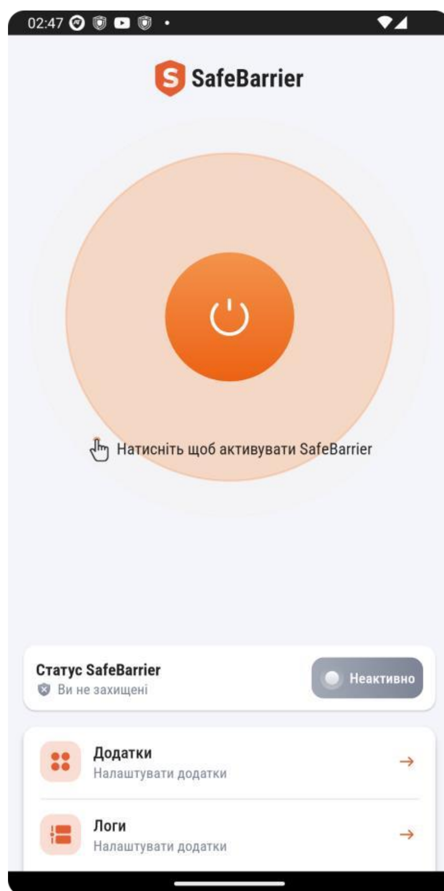


Рисунок 3.6 - Неактивний головний екран додатка

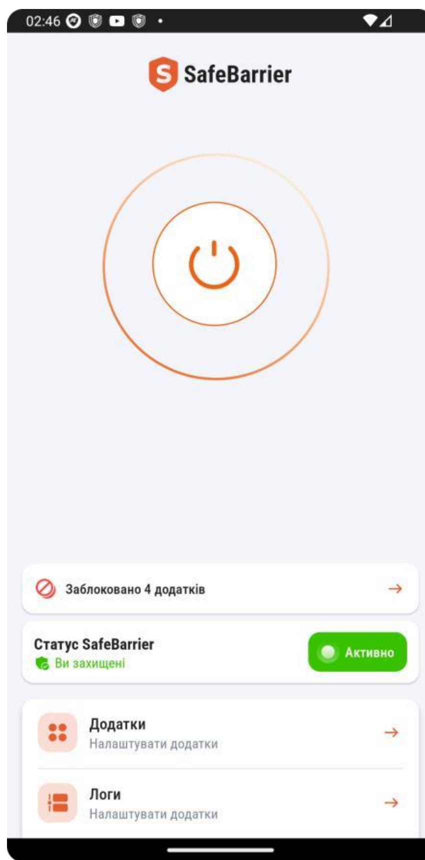


Рисунок 3.7 - Активний головний екран додатка

Статусна інформація включає текстовий індикатор "Ви захищені" у верхній частині та зелену кнопку "Активно" у нижній секції для активного режиму. При деактивованому стані система відображає підказку "Натисніть щоб активувати SafeBarrier" з сірим індикатором "Неактивно", що забезпечує чітке розуміння поточного стану захисту.

Лічильник заблокованих з'єднань розміщений у вигляді червоної плашки з іконкою заборони, демонструючи кількість перехоплених мережевих запитів. Ця метрика оновлюється в реальному часі та надає користувачу миттєву оцінку ефективності фільтрації.

Екран Додатки (рис. 3.8) представляє комплексну систему управління мережевими дозволами на рівні окремих програм. Інтерфейс відображає повний список

встановленого програмного забезпечення, отриманий через системний API PackageManager з використанням дозволу QUERY_ALL_PACKAGES.

Кожен елемент списку містить:

- оригінальну іконку додатку, завантажену з системних ресурсів;
- відображувану назву програми та технічний ідентифікатор пакету;
- чотири індикатори стану мережевого доступу у вигляді іконок сигналу.

Система індикаторів розділена на дві пари: перша відповідає за мобільні дані (LTE/5G), друга - за Wi-Fi підключення. Кожна пара включає окремі кольорові індикатори для вхідного та вихідного трафіку. Кольорові індикатори дозволяє швидко орієнтуватися у статусі з’єднань: зелений колір вказує на дозволені з’єднання, а сірий - на заблоковані. Для зручності перегляду список можна впорядкувати за допомогою кнопки сортування, розташованої у верхньому правому куті.

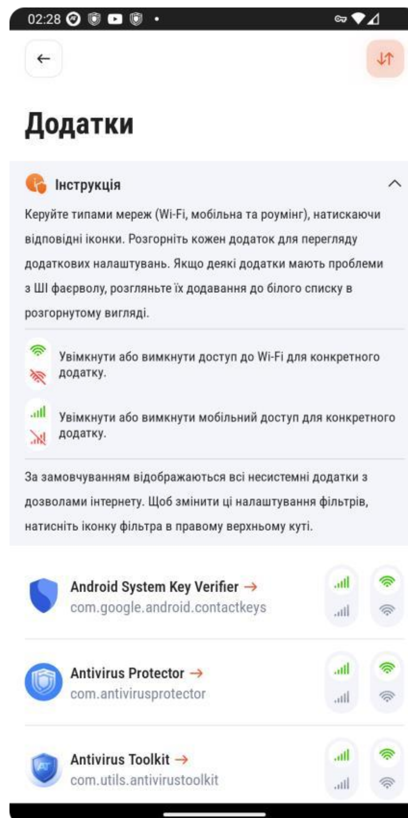


Рисунок 3.8 - Екран “Додатки”

Екран детальних налаштувань окремого додатку (рис. 3.9) надає розширені можливості для точного управління мережевими дозволами. Інтерфейс структурований у вигляді двох основних секцій з незалежними елементами управління.

Секція "Мобільний доступ" контролює використання стільникових мереж (2G/3G/4G/5G) з відображенням поточного стану через зелений індикатор "Увімкнено". Аналогічна секція "Доступ до Wi-Fi" керує підключеннями через бездротові локальні мережі з власним незалежним статусом.

Нижня частина екрану містить хронологічну панель активності, що відображає історію мережевих з’єднань обраного додатку. Таймлайн використовує вертикальну лінію з точками-маркерами для позначення конкретних подій, поруч з якими вказані дати та час активності. Ця функція дозволяє аналізувати паттерни використання мережі та виявляти підозрілу активність.

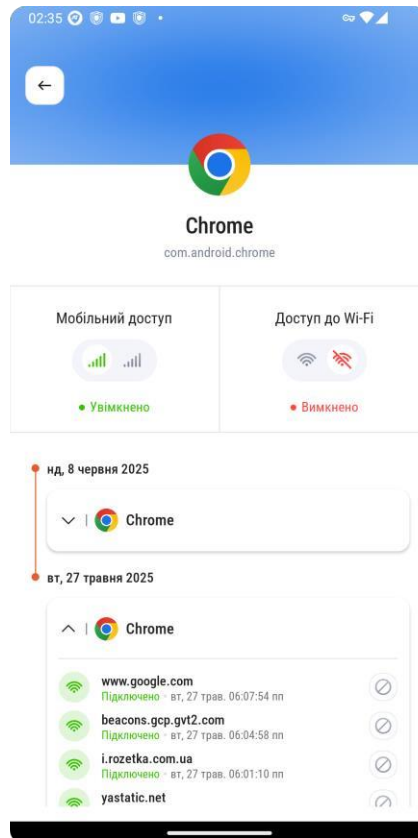


Рисунок 3.9 - Екран детальних налаштувань окремого додатку

Екран "Журнали" (рис. 3.11) відповідає за збереження та відображення інформації про мережеву активність додатків. У ньому реалізовано зручну систему аудиту подій фільтрації, яка дозволяє користувачеві бачити всі мережеві запити в хронологічному порядку. Інтерфейс побудований у вигляді вертикального таймлайну, що спрощує візуальне сприйняття подій:

- Кожен запис у журналі містить;
- Точну мітку часу (дата і година);
- Іконку додатку, який ініціював з'єднання;
- Позначку типу події - чи було з'єднання дозволено або заблоковано;
- Стрілку переходу до докладної інформації про подію.

Система журналювання працює в режимі реального часу, фіксуючи всі спроби з'єднання через VPN-інтерфейс. Усі події зберігаються у локальній базі даних SQLite, що дає змогу застосовувати фільтри за додатком, типом запиту або часовим проміжком.

Крім перегляду, користувач має можливість керувати фільтрацією безпосередньо з журналу: наприклад, заблокувати подібні запити для конкретного додатку або, навпаки, дозволити їх у майбутньому. Додаткову інформації зображено на рисунку 3.10.

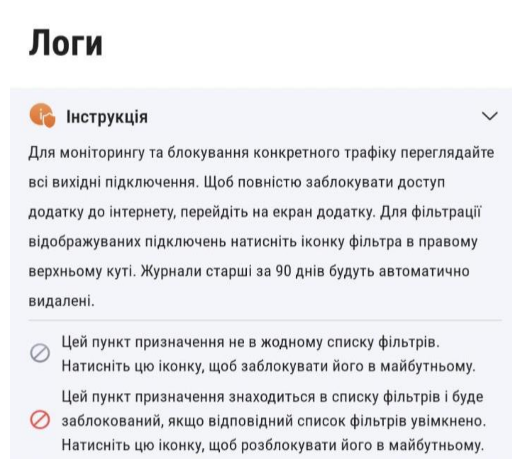


Рисунок 3.10 - Екран "Журнали"

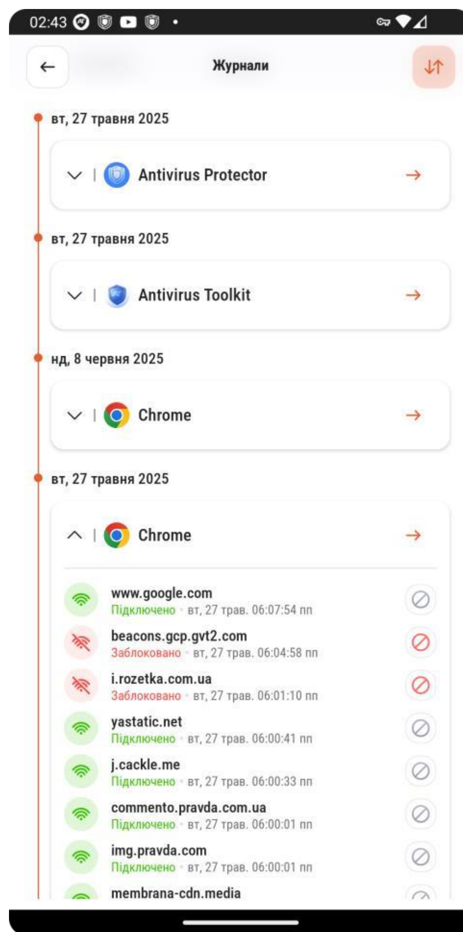


Рисунок 3.11 - Екран “Журнали”

3.3 Апаратне і програмне забезпечення

Додаток розроблений для операційної системи Android з мінімальною версією API рівня 21 (Android 5.0 Lollipop) та цільовою версією API 34 (Android 14.0). Така широка підтримка версій забезпечує сумісність з 95% активних Android пристроїв на ринку.

Мінімальні апаратні вимоги включають:

- оперативна пам’ять: 2 ГБ RAM;
- вільне місце: 50 МБ для установки додатку;
- архітектура процесора: ARM64-v8a (основна), ARM v7a (підтримується);

- мережеві інтерфейси: Wi-Fi 802.11 b/g/n/ac, LTE/5G модем.

ВИСНОВОК

У рамках бакалаврської дипломної роботи було створено мобільний додаток SafeBarrier, що є комплексним технологічним рішенням для контролю доступу до Інтернету на пристроях Android. Додаток успішно поєднує передові методи розробки з ефективними принципами мережевої фільтрації, використовуючи для цього технологію локального VPN. Такий підхід забезпечує контрольований доступ до мережевих ресурсів без необхідності отримувати root-права або залежати від зовнішніх VPN-серверів.

Основні функціональні можливості:

- вибіркове блокування: Можливість блокувати доступ до Інтернету для окремих, обраних користувачем застосунків;
- моніторинг трафіку: Вбудована система для відстеження мережевої активності з веденням детальних журналів (логів);
- гнучкі правила фільтрації: Система дозволяє гнучко налаштовувати правила для контролю трафіку;
- підтримка різних мереж: Додаток коректно працює як з Wi-Fi, так і з мобільним Інтернетом, дозволяючи встановлювати різні правила для кожного типу з’єднання.

Додаток може ефективно використовуватися для батьківського контролю та обмеження доступу дітей до небажаного контенту, забезпечення корпоративної безпеки через контроль мережевої активності службових пристроїв, захисту особистої приватності користувачів та економії мобільного трафіку через блокування фонової активності додатків.

Розроблений мобільний додаток представляє собою комплексне технологічне рішення, яке успішно поєднує передові методи Android-розробки з ефективними принципами мережевої фільтрації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **Android Developers.** Kotlin and Android [Електронний ресурс]. – Режим доступу: https://developer.android.com/kotlin (дата звернення: 06.06.2025).
2. **Android Developers.** VpnService | Android Developers [Електронний ресурс]. – Режим доступу: https://developer.android.com/reference/android/net/VpnService (дата звернення: 06.06.2025).
3. **Android Developers.** Develop for Android | VPN [Електронний ресурс]. – Режим доступу: https://developer.android.com/develop/connectivity/vpn (дата звернення: 06.06.2025).
4. **JetBrains.** Kotlin Programming Language [Електронний ресурс]. – Режим доступу: https://kotlinlang.org/ (дата звернення: 06.06.2025).
5. **JetBrains.** Kotlin for Android | Kotlin Documentation [Електронний ресурс]. – Режим доступу: https://kotlinlang.org/docs/android-overview.html (дата звернення: 06.06.2025).
6. **IEEE Xplore.** Research on network security of VPN technology [Електронний ресурс]. – Режим доступу: https://ieeexplore.ieee.org/document/9418865 (дата звернення: 06.06.2025).
7. **Weichbroth P.** Mobile Security: Threats and Best Practices // *Mobile Information Systems*. – 2020 [Електронний ресурс]. – Режим доступу: https://www.hindawi.com/journals/misy/2020/8828078/ (дата звернення: 06.06.2025).
8. **CSIRO Research.** An Analysis of the Privacy and Security Risks of Android VPN Applications [Електронний ресурс]. – Режим доступу: https://research.csiro.au/isp/wp-content/uploads/sites/106/2016/08/paper-1.pdf (дата звернення: 06.06.2025).
9. **Шевченко Н. О.** Розробка мобільних додатків для Android-платформи: навчальний посібник. – Харків: ХНУРЕ, 2020. – 215 с.

10. **Костюченко В. П.** Основи комп’ютерних мереж і безпеки: підручник. – Київ: КНУ, 2021. – 312 с.
11. **Таненбаум А. С., Уезеролл Д. Дж.** Комп’ютерні мережі. – 5-те вид. – Київ: Діалектика, 2018. – 960 с.
12. VpnService.Builder.Establish Method [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/api/android.net.vpnservice.builder.establish?view=net-android-35.0 (дата звернення: 06.06.2025).
13. ParcelFileDescriptor [Електронний ресурс]. – Режим доступу: https://developer.android.com/reference/android/os/ParcelFileDescriptor (дата звернення: 06.06.2025).
14. Namespace Isolation [Електронний ресурс]. – Режим доступу: https://cycle.io/learn/namespace-isolation (дата звернення: 06.06.2025).
15. **Blokada** - the popular mobile adblocker and VPN for Android [Електронний ресурс]. – Режим доступу: https://blokada.org/#cloud (дата звернення: 06.06.2025).

ДОДАТКИ

ДОДАТОК А

ServiceFirewall.java

```
package com.net.firewall;

import android.annotation.TargetApi;
import android.app.*;
import android.content.*;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.net.*;
import android.os.*;
import android.telephony.*;
import android.text.TextUtils;
import android.util.Log;
import android.util.TypedValue;
import androidx.core.app.NotificationCompat;
import androidx.core.content.ContextCompat;
import androidx.localbroadcastmanager.content.LocalBroadcastManager;
import androidx.preference.PreferenceManager;

import java.io.*;
import java.net.*;
import java.util.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.ReentrantReadWriteLock;

import com.net.firewall.model.Data;

public class ServiceFirewall extends VpnService implements SharedPreferences.OnSharedPreferenceChangeListener {
    private static final String TAG = "ServiceFirewall";

    // Notification IDs
    private static final int NOTIFY_ENFORCING = 1;
    private static final int NOTIFY_WAITING = 2;
    private static final int NOTIFY_DISABLED = 3;
    private static final int NOTIFY_ERROR = 6;

    // Command extras
    public static final String EXTRA_COMMAND = "Command";
    private static final String EXTRA_REASON = "Reason";
    public static final String EXTRA_NETWORK = "Network";
    public static final String EXTRA_UID = "UID";
    public static final String EXTRA_PACKAGE = "Package";
    public static final String EXTRA_BLOCKED = "Blocked";
    public static final String EXTRA_INTERACTIVE = "Interactive";
    public static final String EXTRA_TEMPORARY = "Temporary";

    // State variables
    private State state = State.none;
    private boolean user_foreground = true;
    private boolean last_connected = false;
    private boolean last_metered = true;
```

```
private boolean last_interactive = false;
private boolean temporarilyStopped = false;

// VPN components
private Thread tunnelThread = null;
private Builder last_builder = null;
private ParcelFileDescriptor vpn = null;
private static long jni_context = 0;
private static final Object jni_lock = new Object();

// Data structures
private final Map<String, Boolean> mapHostsBlocked = new HashMap<>();
private final Map<Integer, Boolean> mapUidAllowed = new HashMap<>();
private final Map<Integer, Integer> mapUidKnown = new HashMap<>();
private final ReentrantReadWriteLock lock = new ReentrantReadWriteLock(true);

// Handlers
private volatile CommandHandler commandHandler;
private volatile LogHandler logHandler;
private volatile StatsHandler statsHandler;
private volatile Looper commandLooper, logLooper, statsLooper;

// Receivers
private boolean registeredUser = false;
private boolean registeredConnectivityChanged = false;
private boolean registeredPackageChanged = false;

private ExecutorService executor = Executors.newCachedThreadPool();

public enum Command { run, start, reload, stop, stats, set, householding, watchdog }
private enum State { none, waiting, enforcing, stats }

// Native methods
private native long jni_init(int sdk);
private native void jni_start(long context, int loglevel);
private native void jni_run(long context, int tun, boolean fwd53, int rcode);
private native void jni_stop(long context);
private native void jni_clear(long context);
private native int jni_get_mtu();
private native void jni_done(long context);

@Override
public void onCreate() {
   Log.i(TAG, "Creating ServiceFirewall");
   startForeground(NOTIFY_WAITING, createWaitingNotification());

   initializeNative();
   initializeHandlers();
   registerReceivers();
   setupNetworkMonitoring();

   SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
   prefs.registerOnSharedPreferenceChangeListener(this);
}

private void initializeNative() {
   if (jni_context != 0) {
      Log.w(TAG, "Cleaning existing context=" + jni_context);
```

```
            synchronized (jni_lock) {
                jni_done(jni_context);
                jni_context = 0;
            }
        }
        jni_context = jni_init(Build.VERSION.SDK_INT);
        Log.i(TAG, "Created native context=" + jni_context);
    }

    private void initializeHandlers() {
        HandlerThread commandThread = new HandlerThread("Command", Process.THREAD_PRIORITY_FOREGROUND);
        HandlerThread logThread = new HandlerThread("Log", Process.THREAD_PRIORITY_BACKGROUND);
        HandlerThread statsThread = new HandlerThread("Stats", Process.THREAD_PRIORITY_BACKGROUND);

        commandThread.start();
        logThread.start();
        statsThread.start();

        commandLooper = commandThread.getLooper();
        logLooper = logThread.getLooper();
        statsLooper = statsThread.getLooper();

        commandHandler = new CommandHandler(commandLooper);
        logHandler = new LogHandler(logLooper);
        statsHandler = new StatsHandler(statsLooper);
    }

    private void registerReceivers() {
        // Register package changes
        IntentFilter packageFilter = new IntentFilter();
        packageFilter.addAction(Intent.ACTION_PACKAGE_ADDED);
        packageFilter.addAction(Intent.ACTION_PACKAGE_REMOVED);
        packageFilter.addDataScheme("package");
        ContextCompat.registerReceiver(this, packageChangedReceiver, packageFilter,
ContextCompat.RECEIVER_NOT_EXPORTED);
        registeredPackageChanged = true;

        // Register user changes
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR1) {
            IntentFilter userFilter = new IntentFilter();
            userFilter.addAction(Intent.ACTION_USER_BACKGROUND);
            userFilter.addAction(Intent.ACTION_USER_FOREGROUND);
            ContextCompat.registerReceiver(this, userReceiver, userFilter, ContextCompat.RECEIVER_NOT_EXPORTED);
            registeredUser = true;
        }
    }

    private void setupNetworkMonitoring() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            listenNetworkChanges();
        } else {
            listenConnectivityChanges();
        }
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        if (state == State.enforcing) {
```

```
            startForeground(NOTIFY_ENFORCING, createEnforcingNotification(-1, -1));
        } else {
            startForeground(NOTIFY_WAITING, createWaitingNotification());
        }

        Log.i(TAG, "Received command: " + intent);

        if (intent != null && intent.hasExtra(EXTRA_COMMAND) &&
                intent.getSerializableExtra(EXTRA_COMMAND) == Command.set) {
            handleSetCommand(intent);
            return START_STICKY;
        }

        // Handle service restart
        if (intent == null) {
            SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
            boolean enabled = prefs.getBoolean("enabled", false);
            intent = new Intent(this, ServiceFirewall.class);
            intent.putExtra(EXTRA_COMMAND, enabled ? Command.start : Command.stop);
        }

        commandHandler.queue(intent);
        return START_STICKY;
    }

    private void handleSetCommand(Intent intent) {
        int uid = intent.getIntExtra(EXTRA_UID, 0);
        String network = intent.getStringExtra(EXTRA_NETWORK);
        String pkg = intent.getStringExtra(EXTRA_PACKAGE);
        boolean blocked = intent.getBooleanExtra(EXTRA_BLOCKED, false);

        Log.i(TAG, "Set " + pkg + " " + network + "=" + blocked);

        SharedPreferences prefs = getSharedPreferences(network, Context.MODE_PRIVATE);
        if (blocked) {
            prefs.edit().putBoolean(pkg, blocked).apply();
        } else {
            prefs.edit().remove(pkg).apply();
        }

        reload("notification", this, false);

        Intent ruleset = new Intent(Data.ACTION_RULES_CHANGED);
        LocalBroadcastManager.getInstance(this).sendBroadcast(ruleset);
    }

    // Command Handler
    private final class CommandHandler extends Handler {
        public CommandHandler(Looper looper) { super(looper); }

        public void queue(Intent intent) {
            Command cmd = (Command) intent.getSerializableExtra(EXTRA_COMMAND);
            Message msg = obtainMessage(cmd.ordinal(), intent);
            sendMessage(msg);
        }

        @Override
        public void handleMessage(Message msg) {
```

```
    try {
        synchronized (ServiceFirewall.this) {
            handleIntent((Intent) msg.obj);
        }
    } catch (Throwable ex) {
        Log.e(TAG, "Command error: " + ex);
        showErrorNotification(ex.toString());
    }
}

private void handleIntent(Intent intent) {
    Command cmd = (Command) intent.getSerializableExtra(EXTRA_COMMAND);
    String reason = intent.getStringExtra(EXTRA_REASON);

    Log.i(TAG, "Executing command=" + cmd + " reason=" + reason);

    switch (cmd) {
        case start:
            startVpn();
            break;
        case reload:
            reloadVpn(intent.getBooleanExtra(EXTRA_INTERACTIVE, false));
            break;
        case stop:
            stopVpn(intent.getBooleanExtra(EXTRA_TEMPORARY, false));
            break;
        case stats:
            statsHandler.sendEmptyMessage(1); // MSG_STATS_RESTART
            break;
        default:
            Log.w(TAG, "Unknown command: " + cmd);
    }
}

private void startVpn() {
    if (vpn != null) return;

    Log.i(TAG, "Starting VPN");
    startForeground(NOTIFY_ENFORCING, createEnforcingNotification(-1, -1));
    state = State.enforcing;

    List<Rule> rules = Rule.getRules(true, ServiceFirewall.this);
    List<Rule> allowedRules = getAllowedRules(rules);

    last_builder = createVpnBuilder(allowedRules, rules);
    vpn = establishVpn(last_builder);

    if (vpn == null) {
        throw new IllegalStateException("Failed to start VPN");
    }

    startNative(vpn, allowedRules, rules);
}

private void reloadVpn(boolean interactive) {
    Log.i(TAG, "Reloading VPN, interactive=" + interactive);

    List<Rule> rules = Rule.getRules(true, ServiceFirewall.this);
```

```
        List<Rule> allowedRules = getAllowedRules(rules);
        Builder builder = createVpnBuilder(allowedRules, rules);

        if (vpn != null) {
            stopNative(vpn);
            stopVPN(vpn);
        }

        vpn = establishVpn(builder);
        if (vpn == null) {
            throw new IllegalStateException("Failed to reload VPN");
        }

        startNative(vpn, allowedRules, rules);
        last_builder = builder;
    }

    private void stopVpn(boolean temporary) {
        Log.i(TAG, "Stopping VPN, temporary=" + temporary);

        if (vpn != null) {
            stopNative(vpn);
            stopVPN(vpn);
            vpn = null;
        }

        if (state == State.enforcing && !temporary) {
            stopForeground(true);
            state = State.waiting;
            startForeground(NOTIFY_WAITING,  createWaitingNotification());
        }

        temporarilyStopped = temporary;
    }
}

// Log Handler
private final class LogHandler extends Handler {
    private static final int MSG_PACKET = 1;
    private static final int MSG_USAGE = 2;

    public LogHandler(Looper looper) { super(looper); }

    public void queue(Packet packet) {
        Message msg = obtainMessage(MSG_PACKET, packet);
        sendMessage(msg);
    }

    @Override
    public void handleMessage(Message msg) {
        try {
            switch (msg.what) {
                case MSG_PACKET:
                    logPacket((Packet) msg.obj);
                    break;
                case MSG_USAGE:
                    logUsage((Usage) msg.obj);
                    break;
```

```
            }
        } catch (Throwable ex) {
            Log.e(TAG, "Log error: " + ex);
        }
    }

    private void logPacket(Packet packet) {
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(ServiceFirewall.this);
        if (prefs.getBoolean("log", false)) {
            DatabaseHelper.getInstance(ServiceFirewall.this).insertLog(packet, null, 0, false);
        }
    }

    private void logUsage(Usage usage) {
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(ServiceFirewall.this);
        if (prefs.getBoolean("track_usage", false)) {
            DatabaseHelper.getInstance(ServiceFirewall.this).updateUsage(usage, null);
        }
    }
}

// Stats Handler
private final class StatsHandler extends Handler {
    public StatsHandler(Looper looper) { super(looper); }

    @Override
    public void handleMessage(Message msg) {
        // Stats handling implementation
    }
}

// VPN Builder and Configuration
private Builder createVpnBuilder(List<Rule> allowedRules, List<Rule> allRules) {
    SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);

    Builder builder = new Builder();
    builder.setSession(getString(R.string.app_name));

    // Set VPN addresses
    builder.addAddress("10.1.10.1", 32);
    if (prefs.getBoolean("ip6", true)) {
        builder.addAddress("fd00:1:fd00:1:fd00:1:fd00:1", 128);
    }

    // Add DNS servers
    if (prefs.getBoolean("filter", false)) {
        for (InetAddress dns : getDnsServers()) {
            builder.addDnsServer(dns);
        }
    }

    // Add routes
    builder.addRoute("0.0.0.0", 0);
    if (prefs.getBoolean("ip6", true)) {
        builder.addRoute("2000::", 3);
    }

    // Set MTU
```

```
        builder.setMtu(jni_get_mtu());

        // Configure allowed/disallowed applications
        configureApplications(builder, allowedRules, allRules);

        return builder;
    }

    private void configureApplications(Builder builder, List<Rule> allowedRules, List<Rule> allRules) {
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
        boolean filter = prefs.getBoolean("filter", false);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            try {
                builder.addDisallowedApplication(getPackageName());

                if (filter) {
                    for (Rule rule : allRules) {
                        if (!rule.apply) {
                            builder.addDisallowedApplication(rule.packageName);
                        }
                    }
                }
            } catch (PackageManager.NameNotFoundException ex) {
                Log.e(TAG, "Package not found: " + ex);
            }
        }
    }

    private ParcelFileDescriptor establishVpn(Builder builder) {
        try {
            return builder.establish();
        } catch (Throwable ex) {
            Log.e(TAG, "Failed to establish VPN: " + ex);
            return null;
        }
    }

    private void startNative(ParcelFileDescriptor vpn, List<Rule> allowedRules, List<Rule> allRules) {
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
        boolean log = prefs.getBoolean("log", false);
        boolean filter = prefs.getBoolean("filter", false);

        if (filter) {
            prepareUidFilters(allowedRules, allRules);
            prepareHostsBlocked();
        }

        if (log || filter) {
            if (tunnelThread == null) {
                jni_start(jni_context, Log.WARN);

                tunnelThread = new Thread(() -> {
                    Log.i(TAG, "Starting tunnel");
                    jni_run(jni_context, vpn.getFd(), false, 3);
                    Log.i(TAG, "Tunnel stopped");
                    tunnelThread = null;
                });
```

```
            tunnelThread.start();
        }
    }
}

private void stopNative(ParcelFileDescriptor vpn) {
    if (tunnelThread != null) {
        jni_stop(jni_context);

        try {
            tunnelThread.join(5000);
        } catch (InterruptedException ignored) {
        }

        jni_clear(jni_context);
        tunnelThread = null;
    }
}

private void stopVPN(ParcelFileDescriptor pfd) {
    try {
        pfd.close();
    } catch (IOException ex) {
        Log.e(TAG, "Error closing VPN: " + ex);
    }
}

// Helper Methods
private List<InetAddress> getDnsServers() {
    List<InetAddress> dns = new ArrayList<>();
    try {
        dns.add(InetAddress.getByName("8.8.8.8"));
        dns.add(InetAddress.getByName("8.8.4.4"));
    } catch (UnknownHostException ex) {
        Log.e(TAG, "DNS error: " + ex);
    }
    return dns;
}

private List<Rule> getAllowedRules(List<Rule> rules) {
    List<Rule> allowed = new ArrayList<>();

    boolean connected = Util.isConnected(this);
    boolean metered = Util.isMeteredNetwork(this);

    last_connected = connected;
    last_metered = metered;

    if (connected) {
        for (Rule rule : rules) {
            boolean blocked = metered ? rule.other_blocked : rule.wifi_blocked;
            if (!blocked) {
                allowed.add(rule);
            }
        }
    }
```

```
    return allowed;
}

private void prepareUidFilters(List<Rule> allowedRules, List<Rule> allRules) {
    lock.writeLock().lock();
    try {
        mapUidAllowed.clear();
        mapUidKnown.clear();

        for (Rule rule : allowedRules) {
            mapUidAllowed.put(rule.uid, true);
        }

        for (Rule rule : allRules) {
            mapUidKnown.put(rule.uid, rule.uid);
        }
    } finally {
        lock.writeLock().unlock();
    }
}

private void prepareHostsBlocked() {
    SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
    boolean useHosts = prefs.getBoolean("filter", false) && prefs.getBoolean("use_hosts", false);

    lock.writeLock().lock();
    try {
        mapHostsBlocked.clear();

        if (useHosts) {
            try (InputStream is = getResources().openRawResource(R.raw.hosts);
                 BufferedReader br = new BufferedReader(new InputStreamReader(is))) {

                String line;
                while ((line = br.readLine()) != null) {
                    line = line.trim();
                    if (line.length() > 0 && !line.startsWith("#")) {
                        String[] parts = line.split("\\s+");
                        if (parts.length >= 2) {
                            mapHostsBlocked.put(parts[1], true);
                        }
                    }
                }
            } catch (IOException ex) {
                Log.e(TAG, "Error reading hosts file: " + ex);
            }
        }
    } finally {
        lock.writeLock().unlock();
    }
}

// Notification Methods
private Notification createEnforcingNotification(int allowed, int blocked) {
    Intent main = new Intent();
    main.setClassName("com.myfirewall",  "com.myfirewall.app.MainActivity");

    PendingIntent pi = PendingIntent.getActivity(this, 0, main,
```

```
        PendingIntent.FLAG_UPDATE_CURRENT |
        (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S ? PendingIntent.FLAG_IMMUTABLE : 0));

    NotificationCompat.Builder builder = new NotificationCompat.Builder(this, "foreground")
            .setSmallIcon(R.drawable.ic_app_logo_small)
            .setContentIntent(pi)
            .setOngoing(true)
            .setAutoCancel(false)
            .setContentTitle(getString(R.string.app_name))
            .setContentText("VPN Active");

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        builder.setCategory(NotificationCompat.CATEGORY_STATUS)
                .setVisibility(NotificationCompat.VISIBILITY_SECRET)
                .setPriority(NotificationCompat.PRIORITY_MIN);
    }

    return builder.build();
}

private Notification createWaitingNotification() {
    Intent main = new Intent();
    main.setClassName("com.myfirewall", "com.myfirewall.app.MainActivity");

    PendingIntent pi = PendingIntent.getActivity(this, 0, main,
        PendingIntent.FLAG_UPDATE_CURRENT |
        (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S ? PendingIntent.FLAG_IMMUTABLE : 0));

    return new NotificationCompat.Builder(this, "foreground")
            .setSmallIcon(R.drawable.ic_app_logo_small)
            .setContentIntent(pi)
            .setOngoing(true)
            .setAutoCancel(false)
            .setContentTitle(getString(R.string.app_name))
            .setContentText("Waiting...")
            .build();
}

private void showErrorNotification(String message) {
    Intent main = new Intent();
    main.setClassName("com.myfirewall", "com.myfirewall.app.MainActivity");

    PendingIntent pi = PendingIntent.getActivity(this, 0, main,
        PendingIntent.FLAG_UPDATE_CURRENT |
        (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S ? PendingIntent.FLAG_IMMUTABLE : 0));

    NotificationCompat.Builder builder = new NotificationCompat.Builder(this, "notify")
            .setSmallIcon(R.drawable.ic_error_white_24dp)
            .setContentTitle(getString(R.string.app_name))
            .setContentText("Error: " + message)
            .setContentIntent(pi)
            .setAutoCancel(true);

    NotificationManagerCompat.from(this).notify(NOTIFY_ERROR, builder.build());
}

// Native callbacks
private void logPacket(Packet packet) {
```

```
        logHandler.queue(packet);
    }

    private boolean isDomainBlocked(String domain) {
        lock.readLock().lock();
        try {
            return mapHostsBlocked.containsKey(domain) && mapHostsBlocked.get(domain);
        } finally {
            lock.readLock().unlock();
        }
    }

    private Allowed isAddressAllowed(Packet packet) {
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);

        lock.readLock().lock();
        try {
            packet.allowed = false;

            if (prefs.getBoolean("filter", false)) {
                boolean isKnown = mapUidKnown.containsKey(packet.uid);
                boolean isAllowed = mapUidAllowed.containsKey(packet.uid) && mapUidAllowed.get(packet.uid);

                if (packet.uid == Process.myUid()) {
                    packet.allowed = true;
                } else if (isKnown && !isAllowed) {
                    packet.allowed = false;
                } else if (packet.uid < 2000) {
                    packet.allowed = true; // System apps
                } else {
                    packet.allowed = isAllowed;
                }
            }

            return packet.allowed ? new Allowed() : null;
        } finally {
            lock.readLock().unlock();
        }
    }

    // Network monitoring
    @TargetApi(Build.VERSION_CODES.LOLLIPOP)
    private void listenNetworkChanges() {
        ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkRequest.Builder builder = new NetworkRequest.Builder()
                .addCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET);

        ConnectivityManager.NetworkCallback callback = new ConnectivityManager.NetworkCallback() {
            @Override
            public void onAvailable(Network network) {
                reload("network available", ServiceFirewall.this, false);
            }

            @Override
            public void onLost(Network network) {
                reload("network lost", ServiceFirewall.this, false);
            }
        };
```

```
        cm.registerNetworkCallback(builder.build(),  callback);
    }

    private void listenConnectivityChanges() {
        IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
        ContextCompat.registerReceiver(this, connectivityChangedReceiver, filter,
ContextCompat.RECEIVER_NOT_EXPORTED);
        registeredConnectivityChanged = true;
    }

    // Broadcast Receivers
    private final BroadcastReceiver userReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            user_foreground = Intent.ACTION_USER_FOREGROUND.equals(intent.getAction());
            Log.i(TAG, "User foreground=" + user_foreground);
        }
    };

    private final BroadcastReceiver connectivityChangedReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            reload("connectivity changed", ServiceFirewall.this, false);
        }
    };

    private final BroadcastReceiver packageChangedReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Log.i(TAG, "Package changed: " + intent.getAction());
            reload("package changed", context, false);
        }
    };

    @Override
    public void onSharedPreferenceChanged(SharedPreferences prefs, String key) {
        if ("enabled".equals(key)) {
            if (prefs.getBoolean("enabled", false)) {
                start("preference", this);
            } else {
                stop("preference", this, false);
            }
        }
    }

    @Override
    public void onRevoke() {
        Log.i(TAG, "VPN permission revoked");
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
        prefs.edit().putBoolean("enabled", false).apply();
        super.onRevoke();
    }

    @Override
    public void onDestroy() {
        Log.i(TAG, "Destroying service");
```

```
        synchronized (this) {
            // Stop handlers
            if (commandLooper != null) commandLooper.quit();
            if (logLooper != null) logLooper.quit();
            if (statsLooper != null) statsLooper.quit();

            // Unregister receivers
            if (registeredUser) unregisterReceiver(userReceiver);
            if (registeredConnectivityChanged) unregisterReceiver(connectivityChangedReceiver);
            if (registeredPackageChanged) unregisterReceiver(packageChangedReceiver);

            // Clean up VPN
            if (vpn != null) {
                stopNative(vpn);
                stopVPN(vpn);
                vpn = null;
            }

            // Clean up native context
            synchronized (jni_lock) {
                if (jni_context != 0) {
                    jni_done(jni_context);
                    jni_context = 0;
                }
            }
        }

        super.onDestroy();
    }

    // Static helper methods
    public static void start(String reason, Context context) {
        Intent intent = new Intent(context, ServiceFirewall.class);
        intent.putExtra(EXTRA_COMMAND, Command.start);
        intent.putExtra(EXTRA_REASON, reason);
        ContextCompat.startForegroundService(context, intent);
    }

    public static void stop(String reason, Context context, boolean temporary) {
        Intent intent = new Intent(context, ServiceFirewall.class);
        intent.putExtra(EXTRA_COMMAND, Command.stop);
        intent.putExtra(EXTRA_REASON, reason);
        intent.putExtra(EXTRA_TEMPORARY, temporary);
        ContextCompat.startForegroundService(context, intent);
    }

    public static void reload(String reason, Context context, boolean interactive) {
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
        if (prefs.getBoolean("enabled", false)) {
            Intent intent = new Intent(context, ServiceFirewall.class);
            intent.putExtra(EXTRA_COMMAND, Command.reload);
            intent.putExtra(EXTRA_REASON, reason);
            intent.putExtra(EXTRA_INTERACTIVE, interactive);
            ContextCompat.startForegroundService(context, intent);
        }
    }
}
```

ДОДАТОК Б

MainViewModel.kt

```
package com.safebarrier.viewmodel

import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope

@HiltViewModel
class MainViewModel @Inject constructor(
    @ApplicationContext
    private val context: Context
) : ViewModel() {

    private val _connectionState = MutableLiveData<StartState>(StartState.READY)
    private val _blocked = MutableLiveData<Int>(0)
    private val prefs = PreferenceManager.getDefaultSharedPreferences(context)

    private val INFO_KEY = "info_dialog"
    private val COUNT_BLOCKED_KEY = "blocked_apps"
    private val ENABLED_KEY = "enabled"

    val DEBUG_PREPARE_TAG = "prepared"
    val DEBUG_STOP_TAG = "stop"

    fun setState (state: StartState) {
        _connectionState.value = state
    }

    fun startFirewall () {
        viewModelScope.launch {
            _connectionState.value = StartState.CONNECTING
            delay(3000)
            prefs.edit().putBoolean("enabled", true).apply()
            prefs.edit().putBoolean("filter", true).apply()
            prefs.edit().putBoolean("use_hosts", false).apply()
            prefs.edit().putBoolean("filter_udp", false)
```

```
            prefs.edit().putBoolean("log", false).apply()
            prefs.edit().putBoolean("log_app", true).apply()
            ServiceFirewall.start(DEBUG_PREPARE_TAG, context)
            _connectionState.value = StartState.CONNECTED
        }
    }

    fun stopFirewall () {
        viewModelScope.launch {
            ServiceFirewall.stop(DEBUG_STOP_TAG, context, false)
            prefs.edit().putBoolean("enabled", false).apply()
            _connectionState.value = StartState.READY
        }
    }

    fun getState () : LiveData<StartState> {
        return _connectionState
    }

    fun infoShown () : Boolean {
        return prefs.getBoolean(INFO_KEY, false)
    }

    fun setInfoShown (shown:Boolean) {
        prefs.edit().putBoolean(INFO_KEY, shown).apply()
    }

    fun saveCountBlocked (count:Int) {
        prefs.edit().putInt(COUNT_BLOCKED_KEY, count).apply()
    }

    fun getCountBlocked () : Int {
       return prefs.getInt(COUNT_BLOCKED_KEY, 0)
    }
    fun isFirewallEnabled () : Boolean {
        return prefs.getBoolean(ENABLED_KEY, false)
    }

}
```

ДОДАТОК В

LogViewModel.kt

```
package com.safebarrier.viewmodel

import android.content.Context
import android.database.Cursor
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.net.firewall.DatabaseHelper
import com.net.firewall.Rule
import com.safebarrier.model.LogDateGroup
import com.safebarrier.model.LogEntry
import com.safebarrier.model.LogsSortType
import com.safebarrier.model.repository.AppDataImpl
import dagger.hilt.android.lifecycle.HiltViewModel
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale
import javax.inject.Inject

@HiltViewModel
class LogViewModel @Inject constructor(
    @ApplicationContext
    private val context: Context
) : ViewModel() {

    private val dateFormat = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())
    private val _logs = MutableLiveData<MutableList<LogDateGroup>>()
    private val appDataRepository = AppDataImpl(context)

    fun fetchAllLogs (filter:LogsSortType) {
        viewModelScope.launch(Dispatchers.IO)  {
            val rules = Rule.getRules(false, context)
```

```
        val allLogs = mutableListOf<LogDateGroup>()
        for (rule in rules) {
            val cursor = DatabaseHelper.getInstance(context).getAccess(rule.uid)
            val appLogs = extractLogDateGroupsFromCursor(rule, cursor)
            if (appLogs.isNotEmpty()) {
                allLogs.addAll(appLogs)
            }
        }
        val filteredList = when (filter) {
            LogsSortType.NONE -> allLogs
            LogsSortType.NEWER_OLDER -> allLogs.sortedByDescending { it.date }
            LogsSortType.OLDER_NEWER -> allLogs.sortedBy { it.date }
        }
        //allLogs.sortBy { it.date }
        _logs.postValue(filteredList.toMutableList())
    }
}

fun getCurrentFilter () : LogsSortType {
    return appDataRepository.getLogsSortType()
}

fun fetchAppLogs (uid:Int) {
    viewModelScope.launch(Dispatchers.IO)  {
        val allLogs = mutableListOf<LogDateGroup>()
        val cursor = DatabaseHelper.getInstance(context).getAccess(uid)
        val currentRule = Rule.getRules(false, context).find { it.uid == uid }
        if (currentRule != null) {
            val appLogs = extractLogDateGroupsFromCursor(currentRule, cursor)
            if (appLogs.isNotEmpty()) {
                allLogs.addAll(appLogs)
            }
            //allLogs.sortBy { it.date }
           /* val filteredList = when (filter) {
                LogsSortType.NONE -> allLogs
                LogsSortType.NEWER_OLDER -> allLogs.sortedByDescending { it.date }
                LogsSortType.OLDER_NEWER -> allLogs.sortedBy { it.date }
            }*/
            //allLogs.sortedByDescending { it.date }
```

```
            _logs.postValue(allLogs)
          }
        }
    }

    private fun extractLogDateGroupsFromCursor(rule:Rule, cursor: Cursor?): List<LogDateGroup> {
        val logEntries = mutableListOf<LogEntry>()

        cursor?.let { cursor ->
            if (cursor.moveToFirst()) {
                do {
                    val id = cursor.getLong(cursor.getColumnIndexOrThrow("ID"))
                    val address = cursor.getString(cursor.getColumnIndexOrThrow("daddr"))
                    val date = cursor.getLong(cursor.getColumnIndexOrThrow("time"))
                    val status = cursor.getInt(cursor.getColumnIndexOrThrow("block"))

                    logEntries.add(LogEntry(id, address, date, status))
                } while (cursor.moveToNext())
            }
        }

        val uniqueEntries = logEntries.distinctBy { it.address }
        val groupedEntries = uniqueEntries.groupBy { formatDateOnly(it.date) }

        return groupedEntries.map { (date, entries) ->
            LogDateGroup(rule, date, entries, false)
        }
    }

    fun getLogs () : LiveData<MutableList<LogDateGroup>> {
        return _logs
    }

    private fun formatDateOnly(date: Long): String {
        return dateFormat.format(Date(date))
    }
}
```

ДОДАТОК Г

MainActivity.kt

```
package com.safebarrier.view

@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    private var notBackFragments = listOf<Int>(
        R.id.welcomeFragment,
        R.id.boardFragment,
        R.id.paywallFragment,
        R.id.mainFragment
    )

    private lateinit var viewModel:AppViewModel
    private lateinit var subscriptionHelper:SubscriptionHelper

    private var REQUEST_CODE = 123

    companion object {
        const val NOTIFICATION_DELAY_INTERVAL = 8 * 60 * 60 * 1000L
        const val WIFI_KEY = "wifi"
        const val OTHER_KEY = "other"
        const val APPLY_KEY = "apply"
        const val SCREEN_WIFI_KEY = "screen_wifi"
        const val SCREEN_OTHER_KEY = "screen_other"
        const val ROAMING_KEY = "roaming"
        const val LOCK_DOWN_KEY = "lockdown"
        const val NOTIFY_KEY = "notify"
        const val REQUEST_VPN = 1
    }

    override fun onResume() {
        super.onResume()
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
    intent.getIntExtra("notificationId", 0).let {
        NotificationManagerCompat.from(this).cancel(null, it)
    }
    viewModel = ViewModelProvider(this).get(AppViewModel::class.java)
    setContentView(R.layout.activity_main)
    initRules()
    setupMain()
    setupNotifyChannel()
    //subscriptionListener()
    //InstallReferrerUtil.sendInstall(this@MainActivity)
}

private fun setupNotifyChannel () {
    val notificationManager: NotificationManager = getSystemService(
        NotificationManager::class.java
    )
    val channel = NotificationChannel(
        "foreground",
        "Foreground Service",
        NotificationManager.IMPORTANCE_LOW
    )
    notificationManager.createNotificationChannel(channel)
}

@Deprecated("Deprecated in Java")
override fun onBackPressed() {
    if (!notBackFragments.contains(findNavController(R.id.main_view).currentDestination?.id)){
        super.onBackPressed()
    } else {
        val backIntent = Intent(Intent.ACTION_MAIN)
        backIntent.addCategory(Intent.CATEGORY_HOME)
        backIntent.flags = Intent.FLAG_ACTIVITY_NEW_TASK
        startActivity(backIntent)
    }
}

private fun setupMain () {
    lifecycleScope.launch {
        viewModel.activeSubscription.value = viewModel.activePromo.value!!
```

```
        subscriptionHelper = SubscriptionHelper(this@MainActivity, {
            if (it) {
                InstallReferrerUtil.sendSubscription(this@MainActivity)
            }
            if (!viewModel.activePromo.value!!) {
                viewModel.activeSubscription.postValue(it)
            }
            if (it && findNavController(R.id.main_view).currentDestination?.id == R.id.paywallFragment) {
                findNavController(R.id.main_view).navigate(R.id.mainFragment)
            }
        }
        ) {
            viewModel.subscription.postValue(it)
        }
    }
}

private fun subscriptionListener () {
    viewModel.activeSubscription.observe(this, Observer {
        if (it) stopNotificationService() else startNotificationServiceWithPermissionCheck()
    })
}

private fun initRules () {
    lifecycleScope.launch(Dispatchers.IO) {
        val prefs = PreferenceManager.getDefaultSharedPreferences(this@MainActivity)
        if (prefs.getBoolean("isFirstStart", true)) {
            val rule = Rule.getRules(false, this@MainActivity)
            for (r in rule) {
                r.wifi_blocked = false
                r.other_blocked = false
                updateRule(this@MainActivity, r, rule.toMutableList())
            }
            prefs.edit().putBoolean("isFirstStart",  false).apply()
        }
    }
}

private  fun  startNotificationServiceWithPermissionCheck() {
```

```
    if (Build.VERSION.SDK_INT >= 33) {
        if (ContextCompat.checkSelfPermission(
                this, Manifest.permission.POST_NOTIFICATIONS
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            requestPermissions(
                arrayOf(Manifest.permission.POST_NOTIFICATIONS),
                REQUEST_CODE
            )
        } else {
            startNotificationService()
        }
    } else {
        startNotificationService()
    }
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == 123) {
        if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            startNotificationService()
        } else {}
    }
}
private fun startNotificationService() {
    if (!isServiceRunning(this)) try {
        val intent = Intent(this, NotificationService::class.java)
        startService(intent)
    } catch (e: Exception) {
        e.printStackTrace()
    }
}

private fun stopNotificationService() {
```

```
    try {
        val intent = Intent(this, NotificationService::class.java)
        stopService(intent)
    } catch (e: Exception) {
        e.printStackTrace()
    }
}

private fun isServiceRunning(context: Context): Boolean {
    val manager: ActivityManager =
        context.getSystemService(Context.ACTIVITY_SERVICE) as ActivityManager
    for (service in manager.getRunningServices(Int.MAX_VALUE)) {
        if (NotificationService::class.java.name == service.service.className) {
            return true
        }
    }
    return false
}

fun launchSubscriptionFlow () {
    subscriptionHelper.startBillingFlow(SUBSCRIPTION_ID, this)
}

fun updateRule (context: Context, rule: Rule, listAll: List<Rule>) {
    val root = false
    val wifi = context.getSharedPreferences(WIFI_KEY, Context.MODE_PRIVATE)
    val other = context.getSharedPreferences(OTHER_KEY, Context.MODE_PRIVATE)
    val apply = context.getSharedPreferences(APPLY_KEY, Context.MODE_PRIVATE)
    val screen_wifi = context.getSharedPreferences(SCREEN_WIFI_KEY, Context.MODE_PRIVATE)
    val screen_other = context.getSharedPreferences(SCREEN_OTHER_KEY, Context.MODE_PRIVATE)
    val roaming = context.getSharedPreferences(ROAMING_KEY, Context.MODE_PRIVATE)
    val lockdown = context.getSharedPreferences(LOCK_DOWN_KEY, Context.MODE_PRIVATE)
    val notify = context.getSharedPreferences(NOTIFY_KEY, Context.MODE_PRIVATE)
    if (rule.wifi_blocked == rule.wifi_default) wifi.edit().remove(rule.packageName)
        .apply() else wifi.edit().putBoolean(rule.packageName, rule.wifi_blocked).apply()
    if (rule.other_blocked == rule.other_default) other.edit().remove(rule.packageName)
        .apply() else other.edit().putBoolean(rule.packageName, rule.other_blocked).apply()
    if (rule.apply) apply.edit().remove(rule.packageName).apply() else apply.edit()
        .putBoolean(rule.packageName,  rule.apply).apply()
```

```
        if (rule.screen_wifi == rule.screen_wifi_default) screen_wifi.edit()
            .remove(rule.packageName).apply() else screen_wifi.edit()
            .putBoolean(rule.packageName, rule.screen_wifi).apply()
        if (rule.screen_other == rule.screen_other_default) screen_other.edit()
            .remove(rule.packageName).apply() else screen_other.edit()
            .putBoolean(rule.packageName, rule.screen_other).apply()
        if (rule.roaming == rule.roaming_default) roaming.edit().remove(rule.packageName)
            .apply() else roaming.edit().putBoolean(rule.packageName, rule.roaming).apply()
        if (rule.lockdown) lockdown.edit().putBoolean(rule.packageName, rule.lockdown)
            .apply() else lockdown.edit().remove(rule.packageName).apply()
        if (rule.notify) notify.edit().remove(rule.packageName).apply() else notify.edit()
            .putBoolean(rule.packageName, rule.notify).apply()
        rule.updateChanged(context)
        val listModified: MutableList<Rule> = ArrayList()
        for (pk in rule.related) {
            for (related in listAll) if (related.packageName == pk) {
                related.wifi_blocked = rule.wifi_blocked
                related.other_blocked = rule.other_blocked
                related.apply = rule.apply
                related.screen_wifi = rule.screen_wifi
                related.screen_other = rule.screen_other
                related.roaming = rule.roaming
                related.lockdown = rule.lockdown
                related.notify = rule.notify
                listModified.add(related)
            }
        }
        val listSearch: MutableList<Rule> = if (root) ArrayList(listAll) else listAll.toMutableList()
        listSearch.remove(rule)
        for (modified in listModified) listSearch.remove(modified)
        for (modified in listModified) updateRule(context, modified, listSearch)
    }


}
```