НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
_____/Голуб Б.Л., доц., к.т.н. /
Підпис ПІБ, вчене звання і ступінь

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення системи моніторингу та активації пристроїв ОС Windows на ОС Android»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми к.т.н., доцент_____/ Голуб Б.Л./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської
кваліфікаційної роботи :_____/ Василюк-Зайцева С.В./
підпис ПІБ

Виконав:_____/ Синяєв І.О./
Підпис ПІБ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
Підпис ПІБ, вчене звання і ступінь
"_17_" __березня____2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Синяєву Ігорю Олександровичу

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення системи моніторингу та активації пристроїв ОС Windows на ОС Android

Затверджена наказом ректора НУБіП України № 683 "С" від 24.04.2025

2. Термін подання завершеної роботи на кафедру 2025. 06. 02
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Керівник бакалаврської
кваліфікаційної роботи_____ / Василюк-Зайцева С.В. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Синяєв І.О. /
підпис ініціали та прізвище

Дата отримання завдання 2025 . 02 . 25
рік, місяць, число

Зміст

ВСТУП

У світлі глибоких трансформацій, що охоплюють цифрові системи та інформаційно-технологічну інфраструктуру на глобальному рівні, концепція віддаленого керування стає не просто бажаною функціональністю, а фундаментальним елементом сучасного IT-менеджменту. Комплексне розширення мережевих структур, поява розподілених обчислень, а також популяризація підходів до оптимізації енергоспоживання створюють передумови для переосмислення традиційних методів адміністрування комп'ютерних систем. У цьому контексті ключову роль відіграють механізми, які забезпечують **інтелектуальне управління на апаратному рівні**, зокрема ті, що дозволяють ініціювати активацію пристроїв без фізичного доступу.

Одним із ефективних інструментів, що відповідає зазначеним викликам, є технологія **Wake-on-LAN (WoL)**, яка функціонує відповідно до стандарту IEEE 802.3. В її основі лежить використання широкомовного спеціалізованого пакета (magic packet), що містить унікальний шаблон MAC-адреси цільового пристрою. У разі, якщо мережева карта комп'ютера знаходиться в режимі очікування й налаштована відповідним чином у BIOS/UEFI, отримання такого пакета призводить до пробудження системи. Цей підхід дає змогу здійснювати контроль над інфраструктурою із будь-якої точки, не обмежуючись фізичним розміщенням чи часовими рамками. [6, 9, 12, 13]

Попри те, що технологія Wake-on-LAN була розроблена ще у 1990-х роках, вона не втратила своєї актуальності. Навпаки — в умовах переходу до концепцій мобільності, безперервного доступу й управління IT-ресурсами через персональні пристрої, її значущість зростає. Смартфони і планшети, оснащені потужними комунікаційними модулями, здатні функціонувати як багатофункціональні клієнти керування інфраструктурою, дозволяючи оперативно реагувати на зміну стану мережі, запускати комп'ютери,

контролювати доступ до даних тощо. Саме тому мобільна реалізація інтерфейсів до WoL-процедур — це не просто зручність, а техніко-економічна необхідність. [2, 8]

Метою розробки, описаної в цій роботі, є створення функціонального програмного застосунку для ОС Android, що поєднує можливості сканування підмережі, виявлення активних IP-адрес, збереження даних про пристрої у внутрішній SQLite-базі та реалізації WoL-запитів через формування UDP-пакетів. Це дозволяє користувачеві не лише бачити структуру локальної мережі, а й у будь-який момент запускати потрібні ПК без використання стаціонарних консольних клієнтів чи спеціалізованих мережевих скриптів.

Розроблене рішення відображає сучасну тенденцію до консолідації функціональності в мобільному інтерфейсі, що, з одного боку, знижує поріг входу для користувачів без спеціальної технічної підготовки, а з іншого — забезпечує гнучкість, швидкість реагування та масштабованість мережевих операцій. Реалізація такого ПЗ формує підґрунтя для більш складних архітектур, де керування здійснюється не лише вручну, але й за допомогою автоматизованих політик (наприклад, планових запусків серверів або тригерів на основі подій).

З погляду архітектурної інженерії, впровадження WoL на мобільній платформі супроводжується низкою викликів. Це, зокрема, необхідність роботи з обмеженими правами доступу до мережевого рівня в Android, потреба у використанні нестандартних API для створення сирих сокетів, обхід обмежень у доступі до ARP-таблиць, а також забезпечення сумісності між версіями ОС. Водночас це відкриває перспективи для досліджень у галузі мобільного безпечного обміну даними, оптимізації роботи із системними ресурсами та адаптивного UI/UX-дизайну, орієнтованого на системних адміністраторів і технічний персонал.

Крім цього, важливим аспектом є енергоефективність. Застосування WoL дозволяє значно скоротити час роботи обладнання у стані простою.

Сервери, робочі станції чи лабораторні ПК можуть залишатися вимкненими у позаробочий час, активуючись лише за потреби — наприклад, у момент запланованого бекапу або підготовки до віддаленої лекції. Таким чином, технологія стає основою для реалізації глобальних екологічних ініціатив, пов’язаних зі скороченням споживання енергії в корпоративних і навчальних середовищах.

Загалом, розробка Android-застосунку для керування WoL-пристроями є першим кроком до створення комплексної системи віддаленого управління обчислювальними ресурсами, яка може надалі еволюціонувати у напрямку використання VPN-зв’язків, інтеграції з хмарними платформами та впровадження інструментів кібербезпеки.

Основні положення даної роботи були представлені у вигляді тез доповіді на VII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем» (НУБіП України, Київ, 2025) [14].

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

У контексті стрімкого розвитку інформаційних технологій та переходу підприємств до цифрових моделей управління, необхідність у впровадженні рішень для автоматизованого адміністрування локальних мереж стає дедалі очевиднішою. Надійне дистанційне керування пристроями, зокрема комп'ютерами, робочими станціями та мережевими вузлами, дозволяє не лише скоротити витрати на обслуговування, але й оптимізувати процеси моніторингу та реагування на інфраструктурні події. З огляду на це, у межах даного проєкту реалізується мобільний додаток для платформи Android, орієнтований на здійснення активації пристроїв у локальній мережі з використанням протоколу Wake-on-LAN (WoL). Увага приділяється не лише базовій функціональності, але й архітектурній масштабованості, енергоефективності та інтеграційним можливостям у майбутньому.

Бізнес-вимоги

Для досягнення стратегічних цілей підприємств і підвищення операційної ефективності, до програмного забезпечення висуваються такі бізнес-вимоги:

- Централізоване управління мережею з мобільного пристрою**:** додаток має стати частиною загальної IT-екосистеми підприємства, забезпечуючи швидкий доступ до функцій моніторингу й активації обладнання незалежно від фізичної присутності адміністратора у серверному приміщенні.
- Підвищення доступності технічної інфраструктури**:** функціональність Wake-on-LAN, інтегрована в мобільний інтерфейс,

зменшить середній час відгуку на інциденти або запити користувачів, пов’язані з технічною недоступністю пристроїв.

- Скорочення споживання енергоресурсів: розробка передбачає запровадження контрольованого сценарію увімкнення/вимкнення техніки, що дозволить значно скоротити витрати на електроенергію в умовах великої кількості робочих станцій.
- Оптимізація трудових ресурсів**:** зниження кількості рутинних звернень до технічного персоналу через реалізацію мобільного інтерфейсу для автоматизованої активації пристроїв.

Функціональні вимоги

Усі елементи функціональності мають бути реалізовані з урахуванням принципів модульності, відмовостійкості й інтуїтивної зрозумілості користувацького досвіду:

- Сканування мережевих вузлів**:** програмний інтерфейс повинен ініціювати автоматизований пошук пристроїв у вказаному IP-діапазоні з використанням ICMP-запитів, ARP-кешу або UDP-зондування.
- Формування та надсилання WoL-пакету: реалізація створення "magic packet" відповідно до RFC 2730 із можливістю вказівки користувацького порту для відправлення.
- Графічний інтерфейс контролю пристроїв: панель адміністратора повинна візуалізувати перелік пристроїв, MAC і IP-адреси, а також дозволяти швидке виконання операцій над ними.

- Користувацькі шаблони та групи пристроїв: функціональність додатку повинна підтримувати збереження сценаріїв масового пробудження пристроїв, наприклад, за відділами або класами обладнання.
- Інтерфейс додавання вручну: крім автоматичного сканування, користувач повинен мати змогу вручну додати пристрій, задавши всі його параметри — з подальшою валідацією формату IP і MAC.
- Логування: зберігання інформації про кожну взаємодію з пристроєм, включаючи час відправки пакету, IP-адресу джерела, UID користувача та статус відповіді (успішно / таймаут / відмова).

Системні вимоги

Мобільний застосунок має відповідати вимогам сумісності, продуктивності та адаптивності до гетерогенного середовища користувачів:

- Мінімальна підтримувана версія Android: API 26 (Android 8.0) з обов'язковим використанням сучасних permission-механізмів (runtime permissions) для доступу до мережевих інтерфейсів.
- Сумісність із пристроями WoL: пристрої, до яких здійснюється підключення, повинні мати підтримку WoL, налаштовану у BIOS/UEFI, а мережеві адаптери — перебувати в режимі очікування пакету на визначеному порту.
- Мережеві умови: функціонування у середовищі з локальними сегментами IPv4, з можливістю передавання широкомовного трафіку (255.255.255.255) або directed broadcast.

- Низький рівень споживання системних ресурсів: застосунок повинен працювати у фоновому режимі з використанням WorkManager / JobScheduler, не створюючи excessive wake locks або drain батареї.
- Альтернативні методи поширення: підтримка збірки APK або AAB для встановлення через Google Play або внутрішні канали MDM-систем із підписом власним сертифікатом.

Нефункціональні вимоги

Ці вимоги охоплюють продуктивність застосунку в умовах реальної експлуатації, захист даних користувача, адаптивність до інтерфейсних умов різних пристроїв і локалізацій, а також загальну надійність у роботі в динамічному мережевому середовищі.

- Продуктивність**:** час виявлення пристроїв у підмережі /24 не повинен перевищувати 5 секунд; час генерації та відправлення пакету WoL — менше 100 мс.
- Надійність**:** гарантія стабільної роботи при одночасному відображенні й керуванні щонайменше 250 пристроями.
- Безпечність обробки даних: шифрування локального сховища SQLite/SharedPreferences, захист від MITM-атак у випадку можливих зовнішніх мереж, обмеження доступу до критичних дій через автентифікацію (включаючи biometric prompt).
- Мовна локалізація та адаптивність UI: гнучка система string-ресурсів з підтримкою мінімум трьох мов інтерфейсу, адаптація елементів під DPI/форм-фактор, масштабування шрифтів відповідно до системних налаштувань доступності.

Таким чином, розгорнута система вимог формує цілісне технічне ядро програмного рішення, забезпечуючи підґрунтя для реалізації стійкого, адаптивного та масштабованого мобільного засобу керування локальними мережевими пристроями у корпоративному середовищі або інституційних структурах.

1.2 Моделювання предметної області

У сучасній практиці інженерії програмного забезпечення моделювання предметної області виступає не лише як інструмент концептуального аналізу, а й як формалізований підхід до організації знань про систему. Це не просто підготовчий етап, а повноцінна методологія, що дозволяє створити архітектурно узгоджену, масштабовану та адаптивну інформаційну модель. Зростаюча складність програмних систем, орієнтованих на взаємодію з мережевим оточенням, потребує точного визначення взаємозв'язків між компонентами, їх поведінки та залежностей. В цьому сенсі модель предметної області стає критично важливим фактором не лише реалізації системи, а й її експлуатаційної надійності, підтримуваності та довготривалої еволюційної гнучкості.

Проєкт створення мобільного застосунку для дистанційного керування ПК з ОС Windows через технологію Wake-on-LAN (WoL) вимагає мультипарадигмального підходу до моделювання. Система одночасно охоплює апаратну взаємодію, логіку взаємодії користувача, мережеві протоколи низького рівня та взаємозв’язки з локальною базою даних. Об’єктно-орієнтований підхід (ООП) забезпечує зручну абстракцію для класифікації основних сутностей, їх методів та атрибутів, у той час як поведінкові моделі дозволяють моделювати складні процеси керування у динаміці.

Ключові структурно-логічні компоненти предметної області

Device (Пристрій) — цифровий образ фізичного вузла в мережі, що підтримує пробудження за WoL. Пристрій має інваріантні атрибути

(наприклад, MAC-адреса), динамічні (IP, статус активності), а також повноцінну поведінкову модель: функції ініціації пробудження, опитування стану та перевірки з'єднання. У майбутньому можна реалізувати підкласи для різних типів пристроїв.

User (Користувач) — абстракція оператора системи, що взаємодіє з графічним інтерфейсом застосунку, ініціює запити, змінює конфігурації та споживає результати моніторингу. Модель користувача підтримує розширення до мультирівневої ролі (адміністратор, спостерігач, технік), що дозволяє формувати політики безпеки й контролю доступу.

Network (Мережа) — формалізоване представлення мережевого середовища. Містить логіку розподілу IP-діапазонів, структуру маршрутизації, шлюзи, маски підмереж та інші критичні параметри. Забезпечує функціонал широкомовної передачі (broadcast) і виявлення пристроїв у підмережі.

Wake-on-LAN Command — об’єкт нижчого рівня, який інкапсулює процес створення magic packet та його доставку через транспортний рівень (UDP). Ураховуються особливості реалізації мережевого стеку Android, включно з обмеженнями на raw-сокети.

Database Layer — класова абстракція локального сховища даних, реалізованого на SQLite. Вона містить таблиці пристроїв, журнал подій, конфігураційні налаштування користувача. Доступ до даних здійснюється через обгортки DAO з чітко визначеною транзакційною логікою.

Поведінкові шаблони, робочі сценарії та алгоритмічні процедури

В основі динаміки системи лежать ключові сценарії використання, що мають відповідну послідовність дій, винятки й логіку переходів між станами об’єктів:

Сканування локальної мережі — багатопотоковий процес, що здійснює ARP або ICMP-опитування кожної адреси у заданому діапазоні. Результати фільтруються за валідністю MAC-адрес і зберігаються у локальній БД. У

подальших версіях можливе використання асинхронної обробки через Coroutines або RxJava.

Пробудження пристрою — побудова WoL-пакету та його передача широкомовною адресою на порт 7/9. Застосовується перевірка доступності IP через ping до надсилання команди. Передбачено таймаути, повторні спроби, логування результатів.

1.3 Діаграма прецедентів

Діаграма прецедентів є концептуальним механізмом для формального опису функціональних можливостей програмного забезпечення та визначення взаємодії користувачів із системою. Вона слугує візуальним представленням поведінкових аспектів системи, дозволяючи детально моделювати варіанти використання, визначати зв’язки між актором і системою, встановлювати чіткі межі відповідальності компонентів, а також ідентифікувати можливі точки розширення функціональності у процесі подальшої еволюції програмного забезпечення. [15]

Основні компоненти діаграми прецедентів

Актори – зовнішні сутності, що ініціюють взаємодію з системою. В контексті розглядуваної системи основним актором є Користувач, який здійснює операції з управління та моніторингу пристроїв. Додатково можуть бути передбачені допоміжні актори, такі як адміністратори системи, які здійснюють конфігурацію параметрів роботи мережі або підтримують роботу програмного забезпечення.

Прецеденти – функціональні можливості системи, що доступні актору, визначаючи межі його можливих дій. Вони відображають можливі сценарії використання системи, включаючи основні та альтернативні потоки взаємодії.

Система – програмне забезпечення, яке обробляє запити та реалізує відповідні функції відповідно до закладеної логіки. Вона може містити окремі модулі, що відповідають за обробку команд, збереження історії операцій та забезпечення зворотного зв’язку для користувача.

Пристрій – пасивний елемент, який є об’єктом маніпуляцій, але не виконує автономних дій. У межах розглядуваної предметної області пристрій є кінцевим об’єктом впливу, що підлягає активації через Wake-on-LAN.

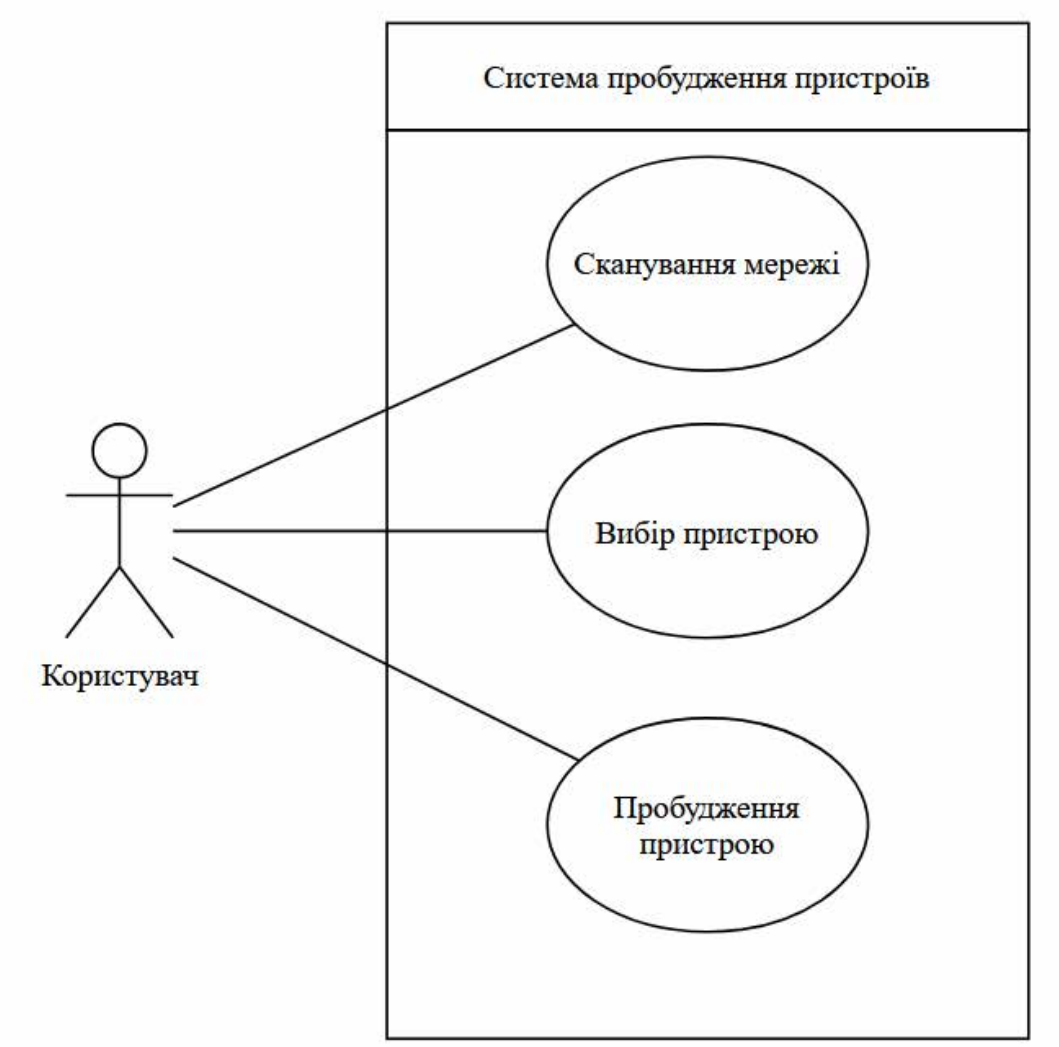

Рис. 1.1 Діаграма прецедентів

Декомпозиція функціональних сценаріїв

Табл. 1.1

Прецедент	Опис
Сканування мережі	Ініціація процесу аналізу топології локальної мережі з метою ідентифікації активних пристроїв, збору інформації про їхні параметри (IP, MAC-адреса, стан тощо) та їх подальшої обробки в межах програмного середовища. Процес сканування може здійснюватися періодично або на вимогу користувача, що дозволяє підтримувати актуальність бази даних пристроїв.
Вибір пристрою	Після виконаного сканування користувач здійснює вибір конкретного пристрою для подальших маніпуляцій, що передбачає фільтрацію, сортування та можливість внесення розширених даних. Додатково можуть бути реалізовані механізми автоматизації вибору на основі історії взаємодії користувача або параметрів пріоритетності.
Пробудження пристрою	Надсилання пакету Wake-on-LAN (WoL) для ініціації активації пристрою, що включає формування запиту, перевірку підтримки функціоналу WoL та контроль успішності виконання операції. У разі невдалого пробудження передбачено механізми повторного надсилання запиту або сповіщення користувача про можливі проблеми.

Динаміка взаємодії акторів із системою

Користувач здійснює взаємодію з мобільним додатком, ініціюючи процес сканування локальної мережі. Після отримання переліку доступних пристроїв він вибирає необхідний пристрій, після чого система надає

можливість виконати команду пробудження через WoL. Програмне забезпечення забезпечує верифікацію стану мережі, перевіряє доступність пристрою та обробляє можливі помилки комунікації.

У рамках розширеної функціональності система може містити засоби автоматичного контролю стану пристроїв та надсилання користувачеві відповідних сповіщень про зміни їхнього статусу. Крім того, можливе інтегрування алгоритмів оптимізації мережевої взаємодії, що дозволить мінімізувати затримки при виконанні команд WoL та підвищити загальну ефективність функціонування системи.

Діаграма прецедентів формує основу для розробки архітектурних рішень та деталізації функціональних вимог, дозволяючи визначити межі відповідальності між підсистемами, розробити механізми взаємодії між компонентами та забезпечити відповідність бізнес-логіці програмного забезпечення. У процесі подальшої роботи над проєктом можливе доповнення моделі новими прецедентами, що враховуватимуть розширення можливостей системи відповідно до зростаючих потреб користувачів та нових технічних викликів.

1.4 Діаграма активності

Діаграма активності є критичним інструментом формального моделювання динамічних аспектів програмного забезпечення, що забезпечує аналітичний підхід до візуалізації алгоритмів виконання операцій та їх взаємозв’язків. Вона дозволяє описати послідовність дій, обробку умовних переходів і альтернативні сценарії взаємодії системних компонентів. Такий підхід сприяє глибшому розумінню структурних залежностей у контексті складних інформаційних процесів. Крім того, діаграма активності виступає основою для подальшого аналізу ефективності роботи системи, дозволяючи оцінювати її продуктивність, потенційні вузькі місця та можливості для оптимізації. [15]

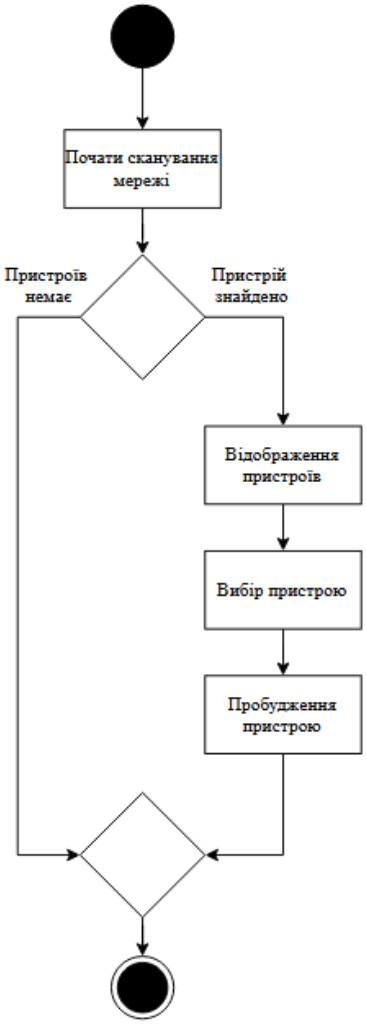

Рис. 1.2. Діаграма активності

Етапи виконання бізнес-логіки

Ініціалізація процесу сканування

Користувач активує процедуру аналізу локальної мережі для ідентифікації підключених пристроїв.

Програмне забезпечення генерує запити на рівні транспортного або мережевого протоколів, застосовуючи методи активного або пасивного виявлення вузлів.

Враховується топологія мережі, типи підключень та можливі обмеження з боку мережевих пристроїв або адміністративних політик.

Обробка результатів сканування

Отримані дані аналізуються з урахуванням таких параметрів, як мережевий сегмент, IP-адресація, MAC-ідентифікатори та відповідність протоколу Wake-on-LAN.

У разі успішного виявлення пристроїв система переходить до подальшого оброблення отриманої інформації.

Якщо пристрої не ідентифіковані, відбувається завершення виконання без подальших дій.

В окремих випадках може бути запропонована спроба повторного сканування з модифікованими параметрами.

Візуалізація результатів та ідентифікація цільового вузла

Визначений набір пристроїв відображається у графічному інтерфейсі користувача з можливістю фільтрації, сортування та редагування параметрів.

Користувач обирає пристрій, що потребує активації, з урахуванням доступних мережевих параметрів та пріоритетності управління.

Передбачено можливість перегляду історії взаємодії з пристроями для швидкого доступу до раніше пробуджених об’єктів.

Формування та передача команди Wake-on-LAN

Програма ініціює генерацію «магічного пакета», що містить повторювану MAC-адресу пристрою.

Надсилання пакета здійснюється через широкомовну або цільову адресу залежно від архітектури мережі.

Верифікація відбувається через аналіз реакції пристрою на запит або непряму оцінку його активності у мережевому середовищі.

При виявленні помилок мережевого рівня користувач отримує розширені рекомендації щодо їх усунення.

Завершення операційного циклу

Незалежно від успішності пробудження пристрою або завершення без дії, система завершує виконання поточного сценарію.

Інформація про взаємодію з мережею логіюється для подальшого аналізу та діагностики.

В разі необхідності користувач може переглянути журнал операцій та вжити додаткових заходів для підвищення ефективності пробудження пристроїв.

Детальна розробка діаграми активності сприяє формалізації логіки управління мережевими пристроями та забезпеченню відповідності між функціональними вимогами та архітектурними рішеннями. Впровадження аналітичного підходу до побудови такої діаграми підвищує предиктивність та ефективність системного моделювання. Крім того, аналіз діаграми активності дозволяє виявити потенційні вузькі місця в системі, провести порівняльний аналіз різних підходів до управління мережевими пристроями та забезпечити високий рівень адаптивності алгоритмів до змінних умов мережевого середовища.

1.5 Діаграма послідовності

Діаграма послідовності є критичним інструментом у формальному аналізі та моделюванні інформаційних систем, що дозволяє не лише визначати послідовність обміну повідомленнями між структурними компонентами, а й виявляти потенційні проблеми у синхронізації процесів. Вона відіграє ключову роль у проєктуванні складних архітектурних рішень, забезпечуючи прозорий механізм візуалізації комунікаційних потоків та алгоритмічної взаємодії між модулями. Завдяки детальному розгляду сценаріїв використання та аналізу можливих відхилень, діаграма послідовності виступає фундаментальним інструментом у забезпеченні надійності та масштабованості програмного забезпечення. [15]

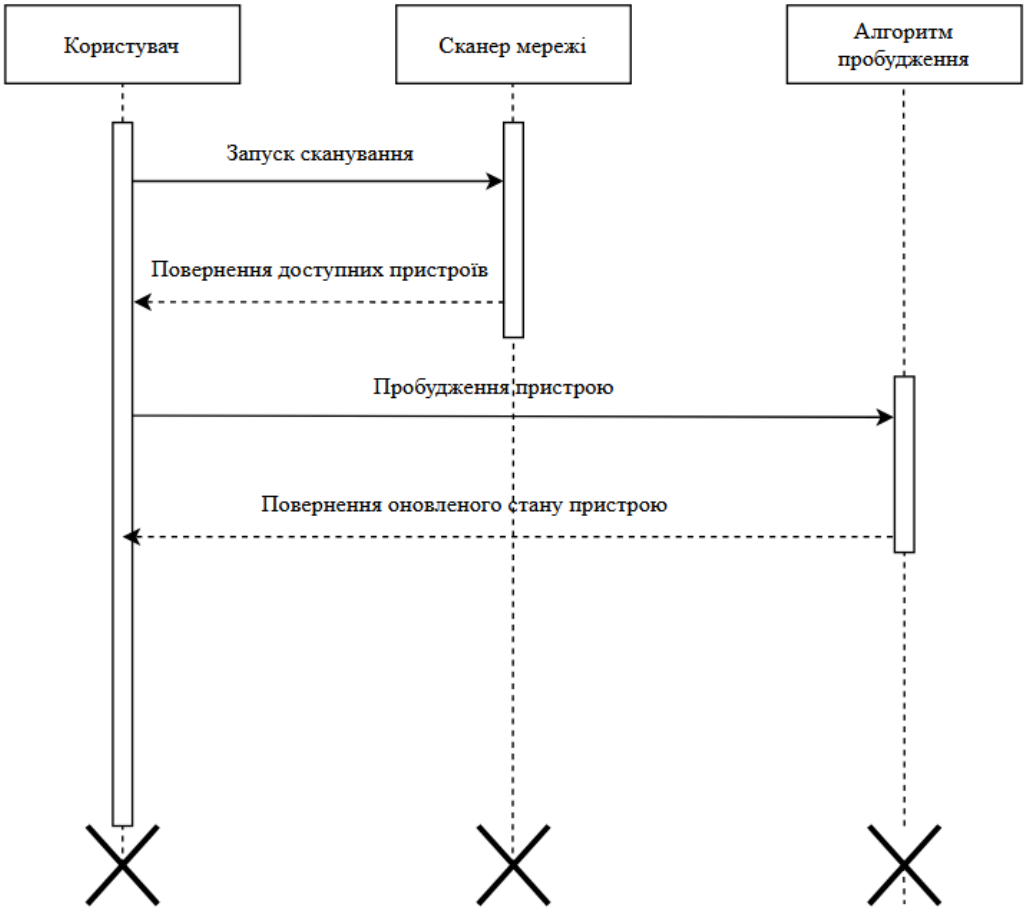

Рис. 1.3 Діаграма послідовності

Основні структурні елементи діаграми

Користувач – ініціатор взаємодії із системою, що надсилає запити на виконання операцій, отримує відповіді та аналізує результати виконання.

Сканер мережі – реалізує механізм ідентифікації доступних пристроїв у локальному сегменті мережі, аналізуючи мережевий трафік та обробляючи відповідні протокольні запити.

Алгоритм пробудження – спеціалізований модуль, що формує та передає керуючі пакети Wake-on-LAN, ініціюючи активацію цільових пристроїв на рівні мережевих інтерфейсів.

Послідовність операцій у межах інформаційного обміну

Запуск процедури сканування мережі

Користувач ініціює запит на виконання аналізу мережевого середовища через інтерфейс програми.

Сканер мережі транслює запити за допомогою ARP, ICMP, SNMP або інших методів залежно від конфігурації мережі.

Виявлені пристрої каталогізуються у структурованому форматі та повертаються користувачеві для подальшого аналізу.

Обробка отриманих результатів та прийняття рішення

Користувач переглядає список доступних пристроїв, обираючи цільовий вузол для активації.

Запит на пробудження передається у відповідний модуль управління живленням пристроїв.

Формування та передача WoL-пакета

Алгоритм пробудження генерує широкомовний Wake-on-LAN (WoL) пакет із MAC-адресою обраного пристрою.

Передача пакета здійснюється через UDP-мережу за допомогою broadcast або direct message залежно від топології підключення.

Успішне прийняття команди пристроєм фіксується через сигнали зворотного зв’язку, що дозволяє перевірити коректність виконання запиту.

Оновлений статус пристрою передається користувачеві, що підтверджує успішність пробудження.

Альтернативні сценарії взаємодії

Відсутність відповіді від пристрою – система генерує повідомлення про неможливість встановлення зв’язку, надаючи користувачеві можливість повторного сканування з коригуванням параметрів аналізу.

Відсутність підтримки WoL – алгоритм визначає обмеження на рівні обладнання або BIOS та пропонує можливі шляхи вирішення, включаючи необхідність змін у конфігурації пристрою.

Перешкоди у мережевій комунікації – передбачено механізми трасування маршруту пакета для виявлення можливих бар’єрів у доставці керуючої команди.

Динамічна зміна мережевої конфігурації – якщо під час взаємодії з пристроями відбуваються зміни у їх доступності, система адаптується шляхом оновлення інформації про топологію мережі в режимі реального часу.

Розширене використання діаграм послідовності дозволяє не лише описати стандартні сценарії взаємодії, але й забезпечити формальний аналіз системи в умовах високої навантаженості, виявлення точок можливої деградації продуктивності та їх усунення на ранніх етапах розробки. Врахування альтернативних шляхів виконання операцій є критичним для підтримки гнучкості архітектури та її адаптації до змінних умов експлуатації. Таким чином, запропонований підхід забезпечує комплексне моделювання програмної системи, підвищуючи її надійність, масштабованість та ефективність роботи у складних мережевих середовищах.

1.6 Структурно-функціональна блок-схема алгоритму

Блок-схема — це структуроване графічне подання алгоритмічних процесів, яке забезпечує формалізовану інтерпретацію логіки функціонування системи. Цей метод є особливо ефективним у контексті програмного забезпечення, де важливо демонструвати як статичні, так і динамічні залежності між логічними компонентами.

Їх застосування особливо виправдане у фазах попереднього проєктування, налагодження системної архітектури, а також під час створення технічної документації та проведення освітніх презентацій. Завдяки блок-схемам розробники можуть чітко окреслити контрольні точки, умовні переходи, розгалуження логіки та сценарії завершення дій. [15]

Ключові переваги, які формують основу ефективності цього підходу:

Прозора концептуалізація процесів — шляхом стандартизованої візуалізації стає можливою оперативна ідентифікація логічних вузлів, що спрощує розуміння як всередині команди, так і серед зовнішніх зацікавлених сторін (наприклад, технічних рецензентів чи аудиторів).

Систематизація і уніфікація — блок-схеми базуються на загальноприйнятих нотаціях (наприклад, ISO/IEC 5807:1985 або IEEE Std 610.12-1990), що забезпечує відповідність документації міжнародним стандартам.

Оптимізація командної комунікації — діаграми виступають спільним інформаційним простором для програмістів, інженерів ПЗ, тестувальників, UI/UX-дизайнерів та аналітиків.

Вона охоплює ключові дії користувача: від моменту запуску до генерації та передачі WoL-пакета, що активує цільовий пристрій у мережі. Така модель дозволяє розробнику формалізувати логіку використання й уніфікувати поведінкові патерни під час рефакторингу, тестування та масштабування системи. Розберемо структурно-функціональну блок-схему алгоритму на рис. 1.4

Початок (Start) — точка входу до системної логіки. Цей блок позначає ініціалізацію мобільного клієнта Android з запуском з ярлика або через фоновий виклик. У технічному сенсі це еквівалент методу onCreate() у головній Activity.

Ініціалізація застосунку — виконання первинної логіки. Здійснюється імпорт ресурсів, ініціалізація бази даних SQLite, перевірка доступу до мережі, формування візуального інтерфейсу відповідно до поточного стану програми.

Умова: перевірка вподобаних пристроїв — логічна перевірка таблиці "favoriteDevices". Залежно від її вмісту система переходить до відповідного гілкування:

Відсутні пристрої — застосунок показує порожнє вікно з інструкцією або натяком на необхідність сканування локальної мережі.

Пристрої наявні — відбувається динамічне рендерення карток із MAC/IP-адресами, іменами пристроїв та кнопками керування (зокрема, "DELETE" та кнопка активації).

Активація сканування — запуск функціоналу сканування підмережі за допомогою локального broadcast або послідовного ARP-опитування, оброблюваного через API Android. Використовується асинхронна обробка результатів з урахуванням обмежень багатопоточності.

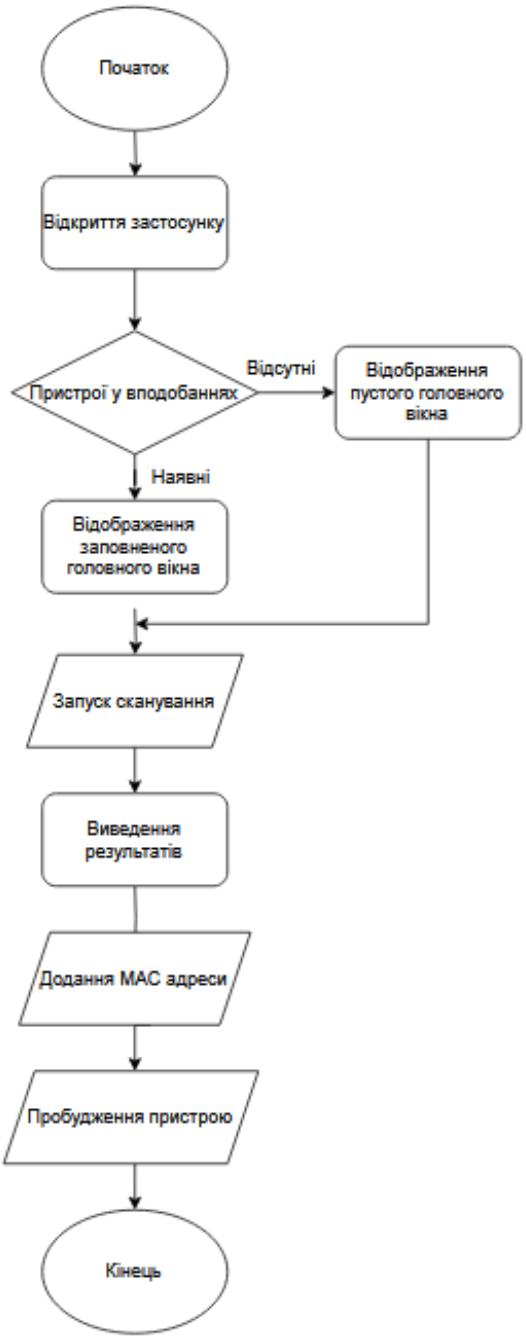

Рис. 1.4 Структурно-функціональна блок-схема алгоритму

Інтерфейсне відображення результатів — користувач бачить список знайдених пристроїв у вигляді структурованих карток. У разі відсутності

MAC-адрес (через обмеження Android API), відображається позначка "Unknown", що дозволяє користувачу ідентифікувати пристрої вручну.

Додавання MAC-адрес вручну — реалізується за допомогою модального діалогового вікна, яке дозволяє зберегти інформацію про пристрій у локальну БД. Це забезпечує подальше використання пристрою без повторного сканування.

Передача WoL-пакету — на завершальному етапі, при натисканні кнопки живлення, застосунок генерує Magic Packet згідно з протоколом Wake-on-LAN (стандарт IEEE 802.3). Пакет надсилається широкомовною адресою (broadcast), орієнтуючись на вказаний MAC. Для цього застосовуються UDP-сокети, зазвичай на порт 9 або 7, залежно від реалізації прошивки пристрою.

Завершення сценарію — програмна логіка переходить у неактивний стан або знову очікує на подальші дії користувача. Жодних системних змін не ініціюється до наступного тригеру події.

2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Загальні відомості про ER-діаграму

ER-діаграма (Entity-Relationship Diagram, ERD) є основоположним інструментом концептуального моделювання баз даних, що забезпечує формалізоване представлення логічної структури інформаційної системи через систему взаємопов'язаних сутностей, їх атрибутів та типів зв’язків. Ця методологія є невід’ємною частиною процесу розробки інформаційних систем, оскільки дозволяє створювати ефективні, структуровані та гнучкі моделі, що враховують специфіку функціонування складних розподілених середовищ. Завдяки ER-моделюванню розробники можуть формалізувати концептуальну модель системи, що, своєю чергою, значно полегшує процес її подальшої реалізації, оптимізації та підтримки.

Застосування ER-моделювання у контексті складних інформаційних систем, зокрема тих, що функціонують у динамічних мережевих середовищах, дозволяє визначити не лише статичну організацію даних, а й забезпечити коректне відображення міжсистемних взаємодій. Це особливо важливо у випадках, коли система потребує реалізації механізмів централізованого керування та моніторингу пристроїв у межах локальної мережі, що вимагає побудови ієрархічних та рекурсивних відносин між її структурними компонентами. Додатково, такий підхід сприяє покращенню механізмів контролю за доступом, забезпеченню узгодженості даних та інтеграції з іншими інформаційними системами.

Основні конструктивні елементи ER-діаграми

Сутності (Entities) – семантичні об'єкти, що відіграють ключову роль у доменній моделі системи. Вони можуть репрезентувати фізичні (пристрої, сервери, користувачі) або логічні (сесії зв’язку, статуси доступності) об’єкти. Формалізація сутностей дозволяє уникнути неоднозначностей при побудові

бази даних та створює передумови для коректного визначення логіки взаємодії між компонентами системи.

Атрибути (Attributes) – дескриптивні характеристики сутностей, що визначають їх унікальні властивості та дозволяють ідентифікувати об’єкти у системі. Серед них можуть бути як прості (IP-адреса, MAC-адреса), так і складні (структуровані дані, що містять кілька взаємопов’язаних полів). Визначення ключових атрибутів сутностей має вирішальне значення для забезпечення ефективності пошуку та маніпулювання інформацією в базі даних.

Зв’язки (Relationships) – формалізовані асоціації між сутностями, що визначають логіку взаємодії між ними. Вони можуть бути бінарними (один до одного, один до багатьох) або мультиплікативними (багато до багатьох), що визначає рівень складності моделі. У системах управління розподіленими обчислювальними середовищами зв’язки можуть мати також атрибути, що описують їхні параметри, такі як часові обмеження чи правила доступу.

Значення ER-моделі у проєктуванні програмного забезпечення

У контексті побудови системи управління моніторингом та активацією пристроїв у локальній мережі ER-моделювання забезпечує:

Формалізацію доменної області – чітке визначення основних об’єктів системи, їх характеристик та взаємозв’язків, що дозволяє усунути неоднозначності у структурі даних.

Оптимізацію логічної та фізичної структури бази даних – попередню ідентифікацію функціональних залежностей між атрибутами, що мінімізує надмірність даних і зменшує ризики їх аномальної дуплікації. Це, у свою чергу, сприяє підвищенню продуктивності системи та зменшенню вимог до ресурсів.

Моделювання динаміки змін у структурі системи – врахування варіативності доступу до ресурсів мережі, що забезпечує адаптивність системи до змін у конфігурації пристроїв та їх статусів у реальному часі. Гнучкість

ER-моделі дозволяє реалізовувати механізми, які підтримують автоматичне оновлення та синхронізацію даних між різними вузлами мережі.

Забезпечення узгодженості бізнес-логіки та її реалізації – побудову структури даних, що корелює з реальними бізнес-процесами, що є критичним аспектом при реалізації складних розподілених архітектур. Це дозволяє мінімізувати втрати продуктивності при інтеграції системи з іншими програмними модулями, що функціонують у межах корпоративної інфраструктури.

2.2 Побудова ER-діаграми

У процесі концептуального моделювання інформаційної системи, орієнтованої на моніторинг та активацію пристроїв у локальному мережевому середовищі, побудова ER-діаграми (діаграми «сутність-зв’язок») виступає як один із найбільш критичних і методологічно вагомих етапів. ER-діаграма забезпечує високорівневе абстраговане подання структури даних, що дозволяє не лише верифікувати повноту логічної моделі, але й синхронізувати уявлення розробників і замовників про архітектуру майбутньої системи. Таке уніфіковане представлення є фундаментальним для коректної трансформації вимог предметної області у внутрішні структури СУБД, підтримки цілісності взаємозв’язків між об’єктами та формалізації процесів взаємодії між ними.

У контексті розробки подібних моделей широке визнання здобув інструмент ERwin Data Modeler, який репрезентує корпоративний рівень підходу до візуалізації й управління моделями даних. Завдяки розширеним можливостям, зокрема підтримці forward- і reverse-engineering, вбудованим механізмам перевірки на відповідність стандартам, автоматичному генеруванню фізичних моделей та SQL-інструкцій, ERwin виступає незамінним інструментом у великих програмно-інженерних екосистемах. Його підтримка складних схем успадкування, багаторівневого представлення метаданих, а також можливість інтеграції з системами контролю версій та управління змінами забезпечує гнучке й надійне управління життєвим циклом

моделей даних у середовищах з високими вимогами до аудиту та відповідності нормативам.

Проте при розробці прототипів, експериментальних застосунків або у випадках обмежених ресурсів, доцільним є звернення до легковагових альтернатив, що поєднують простоту інтерфейсу з достатнім рівнем функціональності. У рамках цього дослідження побудова ER-діаграми була здійснена з використанням вебплатформи **draw.io** (нині відомої як diagrams.net), яка демонструє високу придатність для академічних, навчальних і малих проєктів. Draw.io забезпечує користувачам підтримку шаблонів для ER-моделей, гнучке редагування та організацію об’єктів, збереження у форматах PNG, JPEG, SVG, PDF, XML та синхронізацію з популярними сервісами зберігання даних. Крім того, платформа дозволяє організовувати спільну роботу над моделлю, що є важливим у командному середовищі.

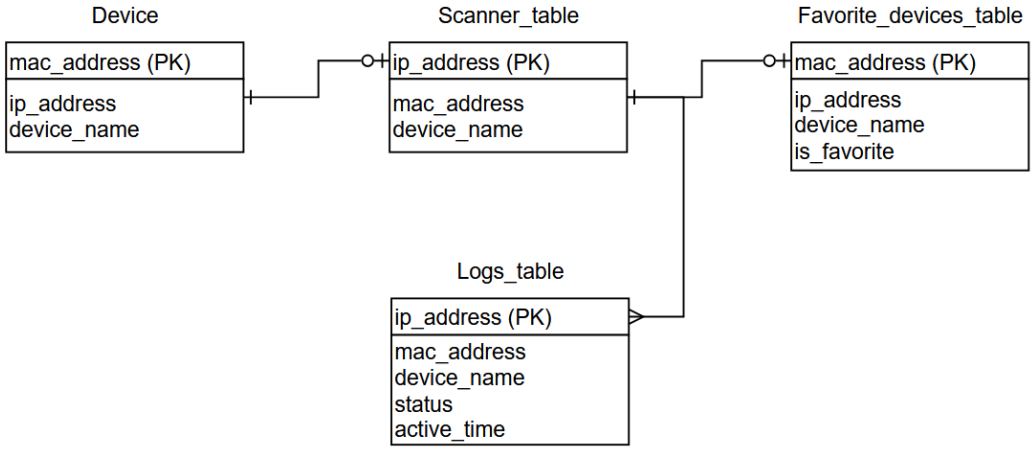

Рис 2.1 ER-діаграма

Розроблена в межах цього етапу ER-діаграма фіксує логічні залежності між сутностями, що відображають об’єкти предметної області: "Device", "Scanner_table", "Logs_table" і "Favorite_devices_table". Кожна сутність містить атрибути, що мають критичне значення для уніфікованої ідентифікації пристроїв — передусім, це MAC-адреса як первинний ключ, а також IP-адреса, ім’я пристрою та інші значущі параметри. Конфігурація зв’язків — від типу «один до одного» до «один до багатьох» — репрезентує семантику взаємодії між таблицями, визначаючи логіку обробки даних у таких підсистемах, як

історія активності, обробка результатів сканування мережі та управління списками обраних пристроїв.

Таким чином, побудова концептуальної моделі даних за допомогою draw.io дозволила ефективно поєднати гнучкість графічного представлення з технічною адекватністю структури моделі до функціональних вимог. Отримана ER-діаграма виступає не лише документаційною одиницею, але й аналітичним артефактом, що забезпечує наступність між етапами проєктування, реалізації та валідації програмного рішення. Її формування є обов’язковим для гарантування когерентності логіки обробки даних, масштабованості та адаптивності СУБД у контексті розвитку мобільної платформи.

2.3 Вибір та обгрунтування СУБД

У контексті створення мобільних застосунків, що функціонують як автономні або частково автономні інструменти керування інфраструктурними об'єктами в локальній мережі, правильний вибір системи управління базами даних (СУБД) визначає не лише технічну надійність, а й адаптивність у майбутньому. Це не просто питання ефективного зберігання інформації — це основа, на якій будуватиметься вся логіка програмної взаємодії, користувацький досвід, стабільність під навантаженням і можливість гнучкої інтеграції з іншими компонентами екосистеми.

Зважаючи на специфіку мобільного середовища, доцільно розглядати СУБД не як ізольований функціональний модуль, а як стратегічний елемент, що повинен враховувати такі параметри, як обмеженість апаратних ресурсів, циклічність підключення до мережі, потреба в захищеності, а також збереження повної функціональності в умовах відсутності зв’язку. У даному розділі проаналізовано найрелевантніші СУБД для реалізації архітектури, яка оперує локальними та мережевими даними, зокрема SQLite, Firebase Realtime Database, Firebase Firestore, Couchbase Lite та ObjectBox.

Ключові вимоги до СУБД у середовищі Android-платформи

При оцінці придатності тієї чи іншої системи управління даними були визначені такі обов’язкові параметри:

Самодостатність роботи в офлайн-режимі без втрати даних;

Продуктивність операцій читання/запису у середовищі з обмеженим CPU та I/O;

Підтримка транзакцій як механізм гарантії цілісності даних у разі збоїв;

Масштабованість у разі розширення системи до мультикористувацької або клієнт-серверної архітектури;

Простота вбудовування в Android-застосунок з урахуванням Android SDK та обмежень версій API;

Можливість майбутньої інтеграції з хмарними сервісами**.**

SQLite

SQLite вже понад два десятиліття зберігає статус базової вбудованої реляційної СУБД для мобільних пристроїв. Її ключова особливість — це самодостатня архітектура, яка не вимагає окремого серверного процесу. Усе управління здійснюється за допомогою бібліотеки, яка безпосередньо виконує SQL-запити до єдиного файл-репозиторію, що розташований у файловій системі пристрою. [5, 10]

Переваги**:**

Ультракомпактність: не займає багато пам’яті, не створює окремих фонових процесів;

Повноцінна підтримка SQL-92**:** транзакції, ключі, індекси, JOIN-запити;

ACID-сумісність: гарантія консистентності навіть у разі аварійного завершення процесу;

Масштабованість до десятків тисяч записів без деградації продуктивності**;**

Велике ком’юніті: величезна кількість бібліотек, wrapper-ів, документації.

Обмеження**:**

Обмеження паралелізму: через файлове блокування можливе зниження продуктивності при одночасному доступі;

Відсутність нативної синхронізації: потрібна реалізація REST або WebSocket-адаптерів для зв’язку з хмарою.

Аналіз альтернативних платформ управління даними

Firebase Realtime Database

Ця хмарна СУБД реалізована як NoSQL-сховище, що дозволяє автоматичну синхронізацію між усіма клієнтами. Вона ідеально підходить для застосунків із необхідністю реального часу, але вимагає постійного підключення до інтернету. [3]

Переваги**:**

Синхронізація у реальному часі між усіма під’єднаними користувачами;

Високий рівень інтеграції з екосистемою Firebase (автентифікація, хостинг, логування).

Недоліки**:**

Відсутність реляційної моделі й підтримки складних фільтрів;

Обмежені можливості для офлайн-роботи та складних транзакцій.

Firebase Firestore

Розвинена альтернатива Realtime Database з кращим підходом до структурування даних і підтримкою транзакцій на рівні документів. [3]

Переваги**:**

Ієрархічна модель даних: документи та колекції;

Покращена підтримка офлайн-роботи через локальне кешування;

Потужні механізми безпеки через правила доступу.

Недоліки**:**

Відсутність складної агрегаційної логіки;

Покладається на мережу для оновлення кешованих даних.

Couchbase Lite

Повністю локальна NoSQL СУБД, орієнтована на корпоративні потреби. Має можливість синхронізації через Couchbase Sync Gateway. [2]

Переваги**:**

Робота в офлайн-режимі з можливістю пізнішої синхронізації;

Підтримка реплікацій і конфлікт-резолвінгу.

Недоліки**:**

Відносно висока складність конфігурації;

Значне споживання ресурсів пристрою.

ObjectBox

Сучасна об’єктно-орієнтована СУБД, що використовує концепцію entity-based моделі замість SQL. [7]

Переваги:

Висока швидкість CRUD-операцій;

Нативна підтримка Kotlin/Java без SQL-запитів;

Генерація типобезпечного коду під час компіляції.

Недоліки:

Менш зріла екосистема;

Обмеження в реалізації складної логіки фільтрації.

Висновок і стратегічна оцінка вибору

На основі комплексного порівняльного аналізу з урахуванням факторів стабільності, продуктивності, архітектурної простоти та ресурсоощадності, вибір на користь SQLite є не лише виправданим, але й стратегічно логічним. Застосунок, орієнтований на локальну обробку даних, не потребує складних механізмів синхронізації у першій ітерації свого життєвого циклу, а отже, базові характеристики SQLite покривають поточні потреби на 100%. У майбутньому розширення до гібридної моделі може бути реалізоване через розробку проміжного API-рівня або синхронізацію через Firebase, що не суперечить архітектурній концепції.

2.4 Створення БД

У рамках створення мобільного застосунку, що виконує завдання з дистанційного керування активацією пристроїв через технологію Wake-on-LAN, внутрішня база даних виступає не просто елементом для зберігання тимчасової інформації. Вона є критично важливим компонентом архітектурного проєкту, що забезпечує логічну цілісність, незалежність від зовнішніх джерел даних і безперебійну роботу застосунку навіть у середовищах із обмеженим або відсутнім доступом до мережі.

У цьому контексті застосування вбудованої реляційної СУБД SQLite виявилося найбільш доцільним: ця система дозволяє реалізувати повноцінну логіку збереження, індексації, фільтрації та модифікації даних без потреби у зовнішньому серверному забезпеченні. Оскільки SQLite функціонує як вбудований компонент у вигляді динамічної бібліотеки, вона має мінімальне навантаження на ресурси пристрою, демонструє високі показники продуктивності та зручна для інтеграції в Android-середовище через нативні інтерфейси Android SDK. [4, 8, 9]

Взаємодія із СУБД реалізована за допомогою спеціалізованого класу DatabaseManager, побудованого на основі SQLiteOpenHelper. Цей клас не лише забезпечує стандартний цикл створення, відкриття й оновлення БД, але й надає механізми для управління версіями схеми, виявлення та обробки змін у структурі таблиць, а також реалізує підготовку SQL-запитів для ініціалізації, модифікації та очищення таблиць. [5, 10]

Структура бази даних та її функціональні компоненти

Таблиця **scanned_devices**

Цей елемент виконує роль оперативного реєстру пристроїв, що були виявлені в процесі сканування локальної мережі. Дані зберігаються в табличній формі з жорстко визначеними типами полів, що дозволяє забезпечити консистентність і мінімізувати помилки при взаємодії з інтерфейсом користувача.

Склад таблиці:

ip_address — унікальний ідентифікатор, який функціонує як первинний ключ;

mac_address — апаратний ідентифікатор мережевого інтерфейсу пристрою;

name — символьне позначення пристрою, що може бути присвоєне вручну або зчитане автоматично.

Після кожного нового сканування вміст таблиці оновлюється, а попередні записи — перезаписуються або очищуються, в залежності від логіки реалізації. Вона виступає основним джерелом інформації для побудови головного інтерфейсу застосунку.

Таблиця **favoriteDevices**

Ця таблиця використовується для зберігання інформації про пристрої, які користувач додав до списку обраних. Її роль — забезпечити швидкий доступ до найбільш часто використовуваних функцій взаємодії без потреби повторного сканування мережі.

Склад таблиці:

mac_address — первинний ключ, який гарантує унікальність пристрою;

ip_address — остання відома IP-адреса, зафіксована під час останньої взаємодії;

device_name — ідентифікаційна мітка, зручна для візуального представлення в UI.

Завдяки використанню цієї таблиці можна зменшити навантаження на підсистему сканування та значно покращити загальний користувацький досвід.

Таблиця **logs**

Попри те, що на етапі поточної реалізації таблиця не є інтегрованою в основну бізнес-логіку застосунку, вона має потенціал стати основою для реалізації аналітики, аудиту або журналювання системних подій.

Склад таблиці:

id_logged — автоматично інкрементований ключ;

ip_address, mac_address, device_name — базові параметри пристрою, на який здійснювалася дія;

status — текстовий опис результату операції;

check_time — точний часовий штамп події.

Така таблиця може бути використана як джерело для побудови графіків активності, формування звітів або реалізації механізмів оповіщення при невдалих спробах взаємодії.

Логіка створення та супроводу бази даних

Створення бази даних виконується в автоматичному режимі при першому зверненні до неї. Метод onCreate() викликається при ініціалізації DatabaseManager і послідовно виконує SQL-команди створення всіх таблиць відповідно до заздалегідь заданих шаблонів. Якщо під час оновлення програми змінюється структура бази або додаються нові поля — метод onUpgrade() дозволяє обробити ці зміни без втрати цілісності всієї системи. У типових випадках реалізується механізм DROP+CREATE для оновлення схем.

З метою полегшення налагодження та тестування було реалізовано функцію logDatabaseContent(), яка зчитує вміст усіх ключових таблиць та виводить їх через Logcat Android Studio. Це дозволяє ефективно виявляти логічні помилки в реалізації збереження або відновлення даних, особливо на етапі раннього тестування.

3. ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕПЕЧЕННЯ

3.1 Вибір інструментарію для розробки програмного забезпечення

У межах реалізації мобільного застосунку, орієнтованого на дистанційне керування пристроями в локальній мережі з використанням протоколу Wake-on-LAN, особлива увага приділяється вибору технологічного інструментарію. Це рішення має не лише задовольнити поточні технічні вимоги, але й забезпечити довготривалу підтримку, масштабованість, а також відкритість до розширення в контексті майбутніх функціональних модифікацій. Сучасний ринок мобільних платформ вимагає від інструментів розробки високої адаптивності до змінного середовища, підтримки стандартів безпеки, ефективності у використанні ресурсів та взаємодії з численними сервісами — як локальними, так і хмарними.

З огляду на вищезазначене, середовище розробки Android Studio було обрано як основна платформа для реалізації системи. Це IDE, офіційно підтримуване Google, є не просто редактором коду, а повноцінним середовищем, яке включає Android SDK, Gradle-механізм складання, Jetpack-компоненти, Android Emulator, систему аналізу продуктивності (Android Profiler), менеджер ресурсів та підтримку CI/CD-процесів. Крім базової розробки, Android Studio надає засоби для A/B-тестування, Firebase-інтеграції, та автоматичної генерації build-конфігурацій, що дає можливість інтегрувати систему у більші проєкти на базі мікросервісної архітектури або у рамках корпоративної інфраструктури. [1, 4]

Однією з головних переваг Android Studio є глибока підтримка візуального редагування інтерфейсів користувача через Layout Editor**,** що в поєднанні з ConstraintLayout дозволяє будувати інтерфейси, які не тільки адаптивні до будь-яких розмірів екранів, але й підтримують оптимізацію UI відповідно до сучасних принципів UX-дизайну. Live Preview, темізація,

відображення станів для accessibility, а також симуляція жестів — усе це робить процес побудови взаємодії з користувачем більш передбачуваним та зручним.

Мовою програмування обрано Java, зважаючи на її стабільну підтримку з боку Android-платформи, наявність широкої екосистеми бібліотек, а також глибоку інтеграцію з Android API на рівні нативних сервісів. Крім цього, Java дає змогу впровадити чітку ієрархічну модель класів, реалізувати шаблони проектування (наприклад, Singleton, Factory, Observer), а також повноцінно застосовувати парадигму об’єктно-орієнтованого програмування в контексті організації системної логіки застосунку.

На рівні локального зберігання даних було застосовано SQLite — ефективну вбудовану СУБД, яка працює у межах файлової системи самого застосунку, забезпечуючи при цьому транзакційність, підтримку зв’язків між таблицями, індексацію та використання агрегатів. Завдяки обгортці SQLiteOpenHelper, розробник отримує засоби контролю версій, міграції схеми та можливість реалізовувати кросверсійні оновлення без втрати цілісності даних. У рамках поточної реалізації за допомогою SQLite організовано кілька структур — таблиці пристроїв, списки обраного, лог дій користувача — що забезпечує оптимізацію як швидкодії, так і структури збереженої інформації.

Для мережевої взаємодії було використано java.net API, зокрема класи DatagramPacket та DatagramSocket, що надали можливість реалізувати механізм надсилання широкомовного "magic packet" для пробудження пристроїв, сумісних із Wake-on-LAN. Такий підхід дозволив уникнути сторонніх залежностей і забезпечити контроль над усіма етапами формування пакету, починаючи від трансляції MAC-адреси до low-level бінарного складання пакету.

Збереження конфігурацій користувача (наприклад, список останніх IP, мітки пристроїв, статуси сканування) реалізовано через SharedPreferences, що забезпечує легковагову модель збереження налаштувань у форматі ключ-

значення без потреби в повноцінній БД. У поєднанні з LiveData або ViewModel ці налаштування можуть бути динамічно підвантажені в інтерфейс у режимі реального часу.

На рівні тестування та відлагодження застосовано Android Debug Bridge (ADB). Цей інструмент надає гнучкі можливості прямого встановлення, логування, знімання дампів пам’яті, керування емуляцією мережі, а також створення скриптів масової інсталяції на реальні пристрої. Завдяки ADB також було можливим профілювання енергоспоживання, перевірка роботи мережевого інтерфейсу в режимі Wake-on-LAN та зчитування зворотного стану пристроїв.

У результаті, вибір Android Studio, Java, SQLite, стандартних Android API та допоміжних інструментів дозволив реалізувати системне, модульне та гнучке рішення, що демонструє високу продуктивність та стабільність у середовищі з обмеженими ресурсами. Усі компоненти технологічного стеку гармонійно взаємодіють, забезпечуючи повноцінну підтримку життєвого циклу розробки, можливість CI/CD-інтеграції, масштабування та уніфікації бізнес-логіки в перспективі розгортання на корпоративному рівні або адаптації під сценарії IoT-інфраструктури.

3.2 Принципи роботи у середовищі візуальної розробки програм

Сучасні середовища візуального проєктування програм, такі як Android Studio, вже давно переросли рамки класичних IDE. Сьогодні вони виступають як комплексні інструментальні платформи, що дозволяють інженерам-програмістам реалізовувати повний життєвий цикл застосунку — від концептуального прототипування до профілювання продуктивності в постпродакшн-середовищі. У випадку створення системи для дистанційного керування пристроями з використанням технології Wake-on-LAN, яка функціонує в середовищі Android, роль такого інструменту як Android Studio важко переоцінити. Він не лише значно скорочує час розробки, а й сприяє

забезпеченню відповідності коду найвищим вимогам якості, сумісності та надійності. [1]

Одним із ключових аспектів Android Studio є двостороння синхронізація між декларативним представленням інтерфейсу (XML) та візуальним редактором Layout Editor. Цей підхід дозволяє одночасно редагувати вигляд екранів у графічному вигляді та негайно бачити результат змін у структурі коду.

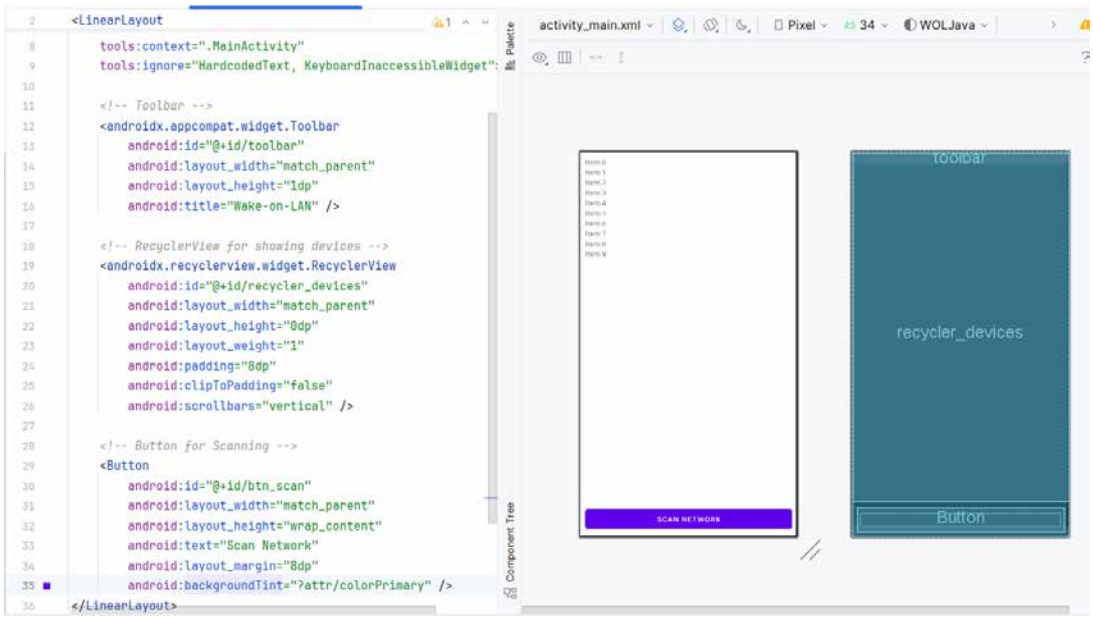

Рис. 3.1 Android Studio – Layout Editor

Реалізований принцип WYSIWYG (What You See Is What You Get) гарантує, що графічна частина відображає реальний стан проекту. У процесі прототипування, коли проєктна концепція проходить численні зміни, це дає змогу розробнику швидко адаптувати застосунок до потреб замовника або кінцевого користувача без повторної компіляції чи перезапуску середовища.

Live Preview, як частина редактора, дозволяє миттєво верифікувати вигляд UI-компонентів, підтримує режим симуляції різних станів (наприклад, активні помилки у полях введення, нічний режим, масштабування шрифтів), і є незамінним при проектуванні адаптивних макетів, орієнтованих на велику кількість пристроїв із різними параметрами екрану та DPI.

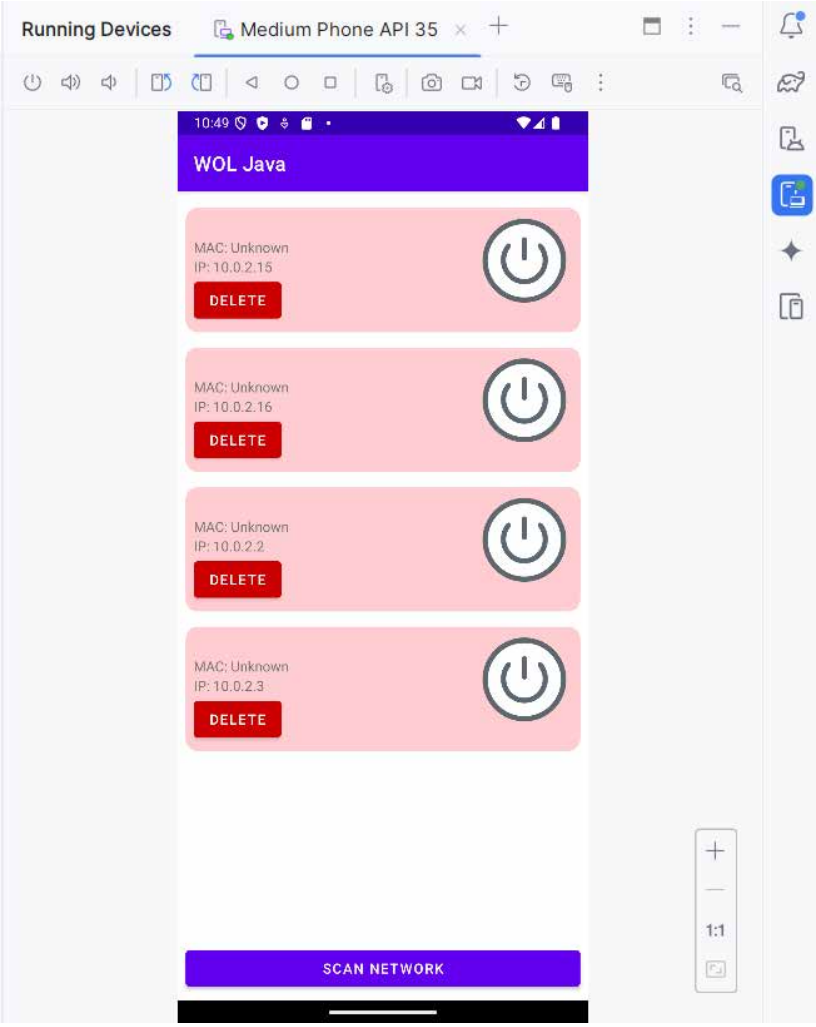

Рис. 3.2 Android Studio – Running Device

Для емулювання роботи застосунку на різних пристроях Android Studio має потужну підсистему — Android Virtual Device Manager (AVD).

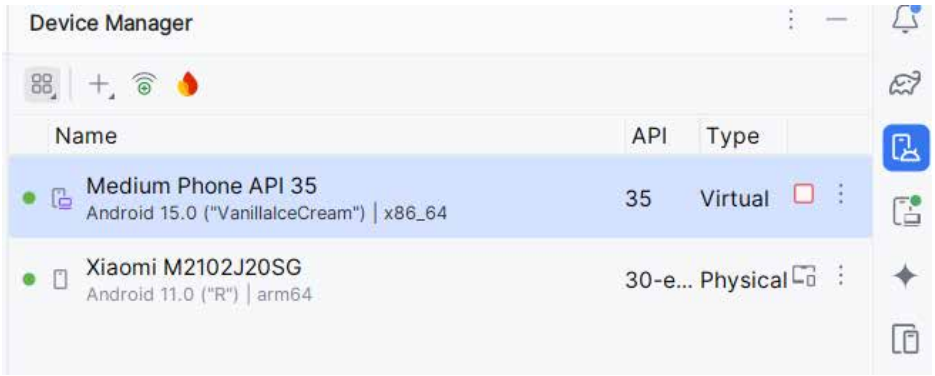

Рис. 3.3 Android Studio – Device Manager

Віртуальні пристрої створюються з параметрами, що включають модель пристрою, версію Android, тип процесора (x86/ARM), обсяг оперативної пам’яті, роздільність дисплея, апаратні можливості (датчики, мережа, камери

тощо). Це дозволяє протестувати застосунок на широкому спектрі конфігурацій, зокрема в умовах слабкого зв’язку, обмеженого живлення або багатозадачності. Для проєктів із мережевим компонентом, таких як Wake-on-LAN, AVD дає змогу змоделювати сценарії з нестабільним підключенням, перевірити тайм-аути, роботу в офлайн-режимі або перемикання між мережами.

Ще одним визначальним аспектом є підтримка архітектурної модульності. Android Studio сприяє реалізації таких шаблонів, як MVVM, MVP, MVI або навіть Clean Architecture, дозволяючи розділити код на незалежні шари: представлення, логіку та дані. Це не тільки покращує підтримуваність проєкту, а й спрощує впровадження юніт-тестів, мок-інтерфейсів, CI/CD-конвеєрів. Особливо актуальною є інтеграція з Jetpack Navigation Component, який надає візуальний редактор маршрутизації, дозволяючи будувати логіку переходів між фрагментами, активностями або діалогами з урахуванням вкладених станів і аргументів.

Усередині Android Studio реалізовано цілу низку механізмів інтелектуального контролю якості коду. Це не лише автозаповнення та підказки, а й Lint-аналізатор, що виявляє стилістичні невідповідності, потенційні помилки, дублювання коду або застарілі API. У складних проєктах він виступає автоматизованим консультантом, який допомагає підтримувати єдиний стиль програмування в команді, не допускаючи деградації структури проекту з часом.

Робота з ресурсами є ще однією сильною стороною Android Studio. Підтримуються адаптивні ресурси: окремі версії для темної/світлої теми, локалізації на десятки мов, варіанти зображень для різних типів щільності пікселів (ldpi, mdpi, hdpi, xhdpi, xxhdpi), стилі й шаблони. Це дає змогу реалізувати інтерфейси, максимально наближені до стандартів UX-дизайну, які добре виглядають на будь-якому пристрої, не потребуючи ручної адаптації під кожен екран.

У підсумку, Android Studio — це не просто зручне IDE, а цілісна інженерна платформа, що підтримує весь процес створення мобільного застосунку: від технічного проектування до профілювання продуктивності. Вона забезпечує відповідність між логікою й інтерфейсом, дозволяє ефективно тестувати, оптимізувати й вдосконалювати додаток, що має особливе значення для систем, орієнтованих на роботу в умовах нестабільного мережевого середовища, як-от реалізації Wake-on-LAN. Завдяки цьому Android Studio не просто спрощує життя розробнику — вона формує стандарти сучасної мобільної інженерії.

4. ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Тестування системи

Теоретичні основи тестування програмного забезпечення

Тестування програмного забезпечення виступає не просто як етап розробки, а як окрема дисципліна в межах інженерії програмного забезпечення, що тісно пов’язана з такими напрямами, як керування якістю, оцінювання ризиків, моделювання поведінки системи та формальна верифікація. Сучасний підхід до тестування визначається не лише потребою в контролі функціональної коректності, але й комплексною перевіркою відповідності системи цілісній моделі якості. У межах життєвого циклу програмного забезпечення тестування виконує роль системної оцінки зрілості, надійності та готовності продукту до експлуатації, особливо у критичних або чутливих до помилок галузях, таких як інформаційна безпека, телекомунікації чи керування інфраструктурою.

Стандартизована основа тестування

ISO/IEC/IEEE 29119: Стандарти програмного тестування

Цей міжнародний стандарт визначає формалізовану модель тестування, яка передбачає декомпозицію тестового процесу на окремі фазові складові. Основними положеннями цієї серії є:

ISO/IEC/IEEE 29119-1 — базові поняття, терміни та визначення, що забезпечують уніфікацію термінологічного поля галузі.

29119-2 — опис процесів: від стратегічного планування й оцінки ризиків до тактичного виконання, збирання метрик та формування підсумкової тестової звітності.

29119-3 — специфікація структури документації: включає тест-плани, тест-кейси, чеклісти, таблиці трасування вимог та дефектів.

29119-4 — техніки: класифікує методи на основі моделей, досвіду, статистики або структурної декомпозиції програмного коду (контроль потоків, покриття умов тощо).

ISO/IEC 25010: Модель якості програмного забезпечення

Цей стандарт пропонує багатовимірну модель якості програмного забезпечення, яка дозволяє не лише формулювати вимоги до продукту, а й структурувати цілі тестування. Вісім основних характеристик охоплюють:

Функціональна придатність — точність, повнота та відповідність функцій очікуванням користувача;

Надійність — стійкість до відмов, здатність до відновлення, поведінка в умовах стрес-навантажень;

Продуктивність — відповідь системи під різними профілями навантаження, швидкість виконання транзакцій, оптимальність використання ресурсів;

Зручність використання — доступність, легкість навігації, зрозумілість взаємодії з інтерфейсом;

Безпека — захист конфіденційності, цілісності, аутентифікація, контроль доступу;

Сумісність — коректна взаємодія з іншими системами, підтримка стандартів і API;

Підтримуваність — зрозумілість коду, можливість внесення змін, адаптація до нових вимог;

Переносимість — здатність функціонувати в нових середовищах без модифікацій або з мінімальними змінами.

Ця модель застосовується як для визначення критеріїв якості, так і для побудови відповідних тестових стратегій.

Категоризація видів тестування

Різноманіття підходів до проектування програмного забезпечення зумовлює потребу в багаторівневому тестуванні, яке враховує як ізоляцію окремих модулів, так і складну взаємодію між компонентами:

Unit-тестування (модульне) — верифікація поведінки одиничного компонента у відриві від інших, як правило, автоматизоване з використанням фреймворків (JUnit, TestNG тощо).

Інтеграційне тестування — дослідження коректності передачі даних між компонентами, типово реалізується з використанням мок-об'єктів, емуляторів і stub-компонентів.

Системне тестування — розгортання повної версії ПЗ у цільовому середовищі з метою симуляції реального використання.

Приймальне тестування (acceptance testing) — оцінка відповідності очікуванням замовника, часто з використанням формальних сценаріїв (user stories, BDD-тести).

Методологічні підходи до тестування

У сучасній практиці застосовуються комбіновані методи тестування, що дозволяють досягти балансу між глибиною аналізу та охопленням поведінкових сценаріїв:

Black-box testing — тестувальник не має доступу до внутрішньої логіки; застосовуються методи еквівалентного розбиття, граничних значень, аналізу таблиць прийняття рішень.

White-box testing — орієнтовано на структуру коду: тестування всіх можливих шляхів виконання, перевірка циклів, умов, логічних операторів.

Grey-box testing — використовує знання архітектури або вхідних/вихідних даних для ефективнішої побудови тестів, часто використовується при тестуванні API або службових інтерфейсів.

Кожен із цих підходів має власні переваги і недоліки, тому у реальних проєктах часто застосовується стратегія комбінування, що дозволяє досягнути вищої достовірності результатів.

Методологічні принципи тестування

У фундаменті ефективної тестової діяльності лежать принципи, які не лише визначають етапи, але й оптимізують підхід до розробки тестових сценаріїв:

Неможливість повного тестування — вимагає застосування евристик та принципів ризик-орієнтованого тестування.

Тестування на ранніх етапах — дозволяє виявити дефекти в специфікаціях, архітектурних рішеннях і дизайні до переходу у фазу кодування.

Контекстна адаптивність — вибір стратегії залежить від сфери застосування ПЗ (напр., фінанси, охорона здоров’я, вбудовані системи).

Принцип «пестицидного ефекту» — запобігає втраті ефективності через повторне використання незмінних тестів, вимагає постійного оновлення та розширення тестового покриття.

Прозорість і трасованість — забезпечення зв’язку між вимогами, тестами і дефектами через механізми трасування (traceability matrix).

Таким чином, сучасне тестування програмного забезпечення — це системна, стандартизована діяльність, що базується на науково обґрунтованих принципах і слугує важелем для гарантування високої якості та стійкості цифрових систем. Впровадження практик, що відповідають міжнародним стандартам, дозволяє створювати масштабовані, підтримувані та безпечні програмні рішення, готові до функціонування в умовах складних інформаційних екосистем.

Практична реалізація тестування програмного забезпечення

Практична реалізація тестування програмного забезпечення в рамках даного дослідження охоплювала повний цикл перевірки мобільного

застосунку для дистанційної активації пристроїв з ОС Windows через Android-платформу. Особливістю проведення тестування стало використання реального експлуатаційного середовища з максимальною ідентичністю умов подальшого використання кінцевими користувачами. Тестування здійснювалось на фізичному мобільному пристрої розробника, що був безпосередньо інтегрований у локальну комп'ютерну мережу, з використанням функціоналу Wake-on-LAN (WoL) для віддаленого пробудження сумісних пристроїв.

З огляду на відповідність цілям функціонального тестування, а також необхідність відповідати критеріям якості, окресленим у стандартах ISO/IEC/IEEE 29119 та ISO/IEC 25010, було визначено цілісну методику перевірки, яка поєднувала як класичні тестові стратегії, так і адаптовані емпіричні процедури, притаманні мобільному середовищу.

Тестове середовище та попередня конфігурація

Застосунок було зібрано у середовищі Android Studio за допомогою системи збірки Gradle. У якості цільового пристрою використовувався персональний смартфон розробника, на який застосунок було завантажено через функцію "Run on Device". Такий підхід забезпечив прямий доступ до нативного середовища виконання з урахуванням апаратних та програмних особливостей пристрою, включно з актуальним рівнем API Android, налаштуванням дозволів і доступом до системних мережевих сервісів.

Для симуляції реального сценарію використання було забезпечено повне мережеве з’єднання: пристрій підключено до Wi-Fi мережі з відкритим широкомовним доступом для UDP-пакетів, маршрутизатор налаштовано для прийому WoL-пакетів, а кінцевий ПК із підтримкою Wake-on-LAN був сконфігурований на рівні BIOS і драйверів мережевого адаптера.

Деталізація тестових етапів:

1. Сканування підмережі локального середовища

В рамках початкового функціонального тесту було перевірено реалізацію алгоритму виявлення активних вузлів у локальній мережі. Застосунок здійснював перебір IP-адрес у діапазоні підмережі, ініціював ARP-запити та збирав відповідні MAC-адреси для ідентифікованих пристроїв. Оцінювалися такі параметри: точність виявлення, швидкість обробки пакету відповідей, стійкість до втрати відповідей, а також інтеграція з базою даних для збереження знайденої інформації.

2. Тестування надсилання "magic packet"

Було ретельно перевірено модуль, відповідальний за генерацію та надсилання WoL-пакетів, які базуються на протоколі UDP і містять в структурі 6 байтів 0xFF та 16 повторів MAC-адреси цільового пристрою. Усі надіслані пакети реєструвались у логах маршрутизатора, а стан віддаленого пристрою відслідковувався з боку системи. Було підтверджено негайне фізичне пробудження пристрою після отримання пакету, що засвідчує правильність формування та маршрутизації трафіку.

3. Операції з локальною базою даних (SQLite)

Особливу увагу приділено стабільності обробки запитів до вбудованої СУБД. Перевірено логіку створення, оновлення та видалення записів пристроїв у таблицях "scanned_devices" та "favoriteDevices". Було підтверджено консистентність даних між циклами запуску додатку, підтримку індексації для пришвидшення вибірок, а також правильну реалізацію обробки повторних записів (unique constraints).

4. Тестування на стійкість до помилок

Було змодельовано декілька ситуацій з відмовами: вимкнене Wi-Fi-з'єднання, MAC-адреса поза підмережею, закриті порти, відсутність відповіді від пристрою. Застосунок у кожному випадку відображав користувачеві інформативне повідомлення, без зависань чи збоїв. Це свідчить про ефективне опрацювання винятків та відповідність принципам fail-safe.

5. Перевірка відповідності UX/UI

На рівні користувацького інтерфейсу було протестовано сценарії навігації, адаптацію до темної теми, зміну мови системи та відображення динамічного контенту. Крім того, було оцінено ергономічність використання, розташування інтерактивних елементів, час відгуку та читабельність на різних розмірах екранів.

4.2 Апаратні та технічні засоби.

У межах реалізації мобільного застосунку, що виконує функцію дистанційної активації пристроїв у локальній комп'ютерній мережі за допомогою технології Wake-on-LAN, надзвичайно важливою є відповідність програмно-апаратного забезпечення як мінімальним, так і рекомендованим експлуатаційним вимогам. Ці параметри охоплюють апаратну конфігурацію кінцевого Android-пристрою, характеристику програмного середовища, умови функціонування локальної мережі, а також специфіку технічної реалізації серверної частини або клієнтського обладнання, яке має реагувати на WoL-пакети.

Аналіз та визначення цих характеристик є невід'ємною частиною інженерного підходу до створення стабільної, надійної і масштабованої системи.

Апаратні вимоги до мобільного пристрою (Android-смартфона)

Табл. 4.1

Ресурс	Мінімальні вимоги	Рекомендовані вимоги
Процесор	1.4 ГГц (4 ядра)	2.0 ГГц і вище (8 ядер)
Оперативна пам'ять	2 ГБ	4 ГБ і вище
Вбудована пам'ять (ROM)	8 ГБ	32 ГБ і вище
Мережевий модуль Wi-Fi	802.11n	802.11ac або вище

Табл. 2 «продовження»

Ресурс	Мінімальні вимоги	Рекомендовані вимоги
Bluetooth	версія 4.0	версія 5.0 і вище
USB-порт	Micro USB / USB-C	USB-C
Живлення	Акумулятор 2500 мА·г	Акумулятор 4000 мА·г і вище
Екран	4.5" (854x480)	6" (1920x1080) або вище
Операційна система	Android 7.0 Nougat	Android 10 і вище
Сенсори	Wi-Fi, GPS, акселерометр	Підтримка енергоефективного Wi-Fi Scan

Програмні вимоги до мобільного пристрою

Табл. 4.2

Ресурс	Мінімальні вимоги	Рекомендовані вимоги
Версія Android SDK	API level 24 (Android 7.0)	API level 29 (Android 10) або вище
Права доступу	Локальні мережі, Wi-Fi, інтернет	Повний доступ до мережевих функцій, фонових процесів
Доступ до Play Services	Не обов’язково	Рекомендовано
Підтримка SQLite	Вбудована	Повна сумісність
Мова програмування	Java	Java (8+)
Доступ до хмарних API	Не обов’язково	За потребою інтеграції
Доступ до служб локації	Вимкнено	Необхідно для сканування Wi-Fi у фоновому режимі

Додаткові апаратно-мережеві вимоги

Мережеве середовище

Для надійного передавання широкомовних WoL-пакетів необхідна наявність стабільного каналу зв’язку, з підтримкою протоколів рівня мережевого доступу.

Мережеве середовище

Табл. 4.3

Параметр мережі	**Мінімальні вимоги**	**Рекомендовані вимоги**
Тип з'єднання	Wi-Fi 802.11n	Wi-Fi 802.11ac або вище
Частотний діапазон	2.4 ГГц	2.4 ГГц + 5 ГГц
Пропускна здатність	5 Мбіт/с	20 Мбіт/с і вище
Середня затримка (latency)	до 100 мс	< 30 мс
Втрати пакетів	< 2%	Близькі до нульових значень
Підтримка широкомовних пакетів	Часткова	Повна підтримка UDP Broadcast/Multicast
DHCP/статична адресація	Будь-яка	Статична IP-адреса для стабільності

Апаратні вимоги до цільового ПК з підтримкою Wake-on-LAN

Табл. 4.4

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Материнська плата	BIOS із базовою підтримкою Wake-on-LAN	BIOS із опціями Wake on PCI/PME, ACPI S3/S4
Мережевий адаптер	Ethernet 10/100 Мбіт/с з підтримкою WoL	Гігабітний адаптер з підтримкою Magic Packet
Операційна система	Windows 7 / 8 / 10	Windows 10 / 11 або Linux зі сховищем wol-tools
Драйвери	Базові драйвери виробника	Оновлені драйвери з підтримкою енергозбереження
Джерело живлення	Постійне підключення до мережі	+5VSB живлення на порту LAN для пасивного прослуховування
Енергозберігаючий режим	Режим сну (Sleep) або очікування (Standby)	Глибокий режим S3 або S4 з активною підтримкою WoL
Додаткові налаштування	WoL активований вручну в BIOS і ОС	Автоматичне пробудження з журналюванням подій

Загальний аналітичний висновок

Комплексна відповідність апаратним, мережевим і програмним вимогам є передумовою ефективної реалізації системи моніторингу та дистанційного керування пристроями. Недотримання навіть одного з ключових параметрів — наприклад, відсутність підтримки широкомовного трафіку або відключене

живлення мережевого порту в режимі очікування — здатне повністю нівелювати працездатність рішення.

Запропоновані мінімальні та рекомендовані конфігурації дозволяють масштабувати рішення відповідно до конкретних умов експлуатації, забезпечують безперервність сервісу та адаптивність до змін у топології мережі. Ретельно визначені параметри дозволяють досягти оптимального балансу між технічною складністю впровадження, витратами на апаратне забезпечення та експлуатаційною надійністю системи.

4.3 Підготовка та поширення інсталяційного пакету застосунку

З метою забезпечення можливості повноцінної експлуатації розробленого мобільного застосунку поза межами середовища розробки, обов’язковим етапом стало формування інсталяційного пакета у форматі APK (Android Package Kit). Цей пакет є стандартом для доставки Android-застосунків на кінцеві пристрої, містить усі необхідні артефакти (ресурси, байткод, AndroidManifest.xml), та дозволяє незалежне встановлення на фізичний пристрій без залучення Android Studio чи Google Play.

APK-файл було підготовлено в підписаному вигляді із застосуванням ключа розробника, що забезпечує валідацію цілісності під час інсталяції. Поширення пакета можливе за допомогою кількох каналів:

- пряме встановлення через USB-кабель або локальну мережу;
- розгортання у внутрішньому середовищі підприємства через систему MDM (Mobile Device Management);
- доступ через захищений сервер або внутрішній портал із посиланням на APK-файл;
- розміщення у Google Play (після проходження процесу перевірки та відповідності політикам публікації).

Для успішного встановлення за межами Google Play необхідно активувати відповідний дозвіл на встановлення програм із зовнішніх джерел у налаштуваннях пристрою. Пакет не потребує зовнішніх залежностей, API чи з'єднання з хмарними сервісами, що забезпечує його повну автономність та застосовність у середовищах із обмеженим або ізольованим мережевим доступом.

Формування такого інсталяційного пакету є ключовим етапом у трансформації прототипу застосунку в завершений продукт, готовий до впровадження в реальні експлуатаційні умови, включно з навчальними закладами, IT-відділами та підприємствами із розгалуженою локальною інфраструктурою.

4.4 Опис роботи програми

Розроблений мобільний застосунок "WOL Java" реалізує функціонал сканування локальної мережі, додавання пристроїв до списку для пробудження через Wake-on-LAN, а також зберігання вибраних записів. Основний інтерфейс побудовано з урахуванням принципів мінімалізму та зручності користування.

Головний екран

Після запуску програми користувач потрапляє на головний екран, який одразу відображає список збережених пристроїв. Кожен пристрій подано у вигляді картки із зазначенням:

імені пристрою;

MAC-адреси;

IP-адреси;

кнопки запуску (подачі WoL-пакету);

кнопки видалення ("DELETE").

Іконка увімкнення виконує функцію генерації та надсилання WoL-пакета за вказаними параметрами (MAC-адресою). Пристрій, що належно відреагував, зазвичай вмикається через декілька секунд після натискання.

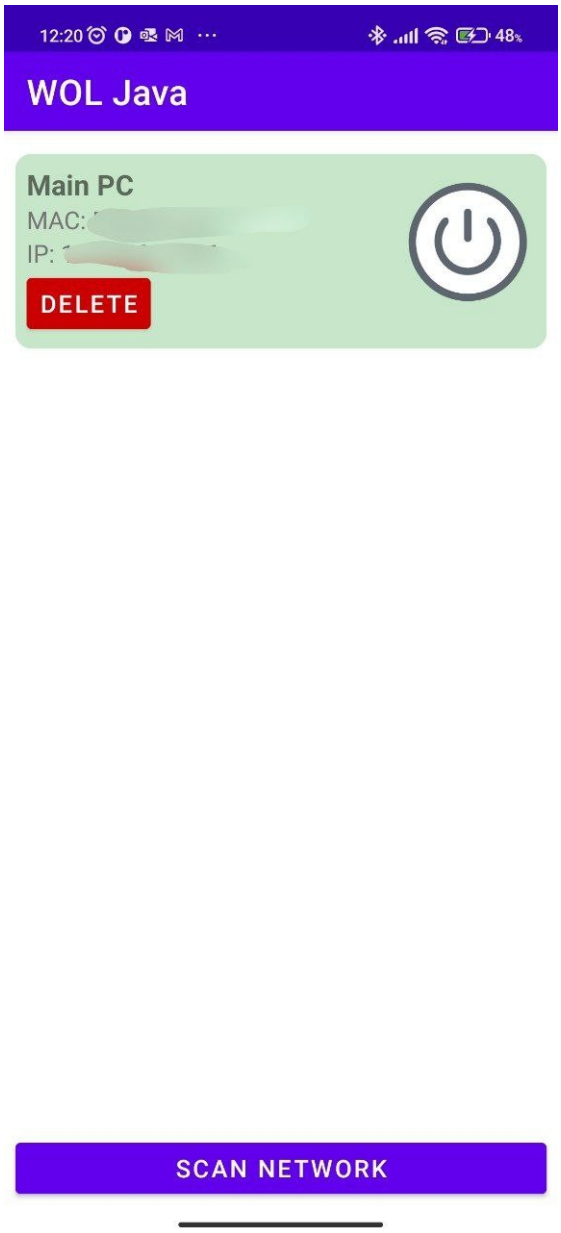

Рис. 4.1 Головний екран застосунку

Результати сканування

Після натискання кнопки "SCAN NETWORK" здійснюється обхід підмережі та відображення усіх знайдених активних IP-адрес. Якщо пристрій не був раніше збережений, і його MAC-адреса не може бути визначена (через обмеження Android), у полі MAC вказується "Unknown". Такі записи виділено червоним кольором для зручності ідентифікації нових пристроїв.

Користувач має можливість додати знайдений пристрій до бази даних вручну, вказавши ім’я та MAC-адресу — це особливо актуально, якщо WoL

підтримується, але автоматичне визначення MAC-адреси неможливе через обмеження операційної системи.

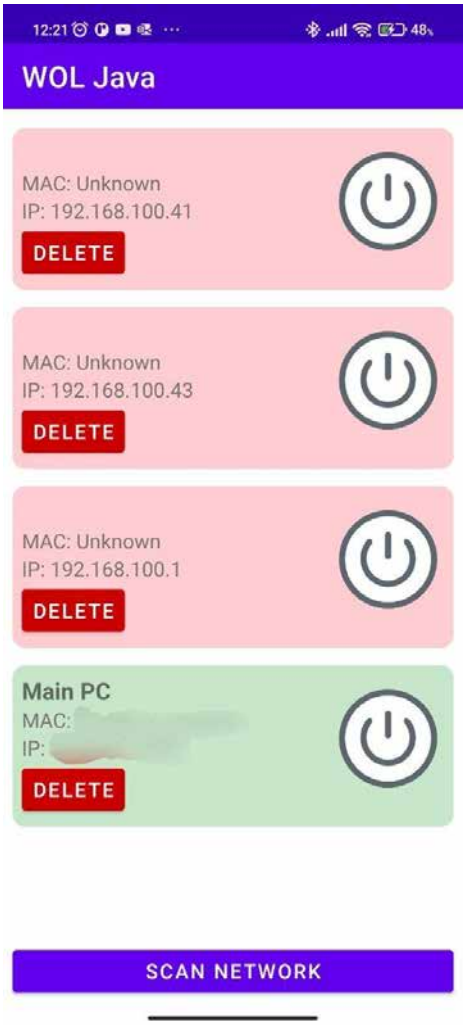

Рис. 4.2 Список знайдених пристроїв

Додавання пристрою

При натисканні на незаповнений пристрій відкривається діалогове вікно з двома полями:

Device Name (ім’я пристрою);

MAC-адреса (в форматі XX:XX:XX:XX:XX:XX).

Кнопка "ADD" дозволяє зберегти пристрій до локальної бази даних та зробити його доступним для пробудження в подальшому. Також передбачена кнопка "HELP?", що відкриває довідку з інструкцією.

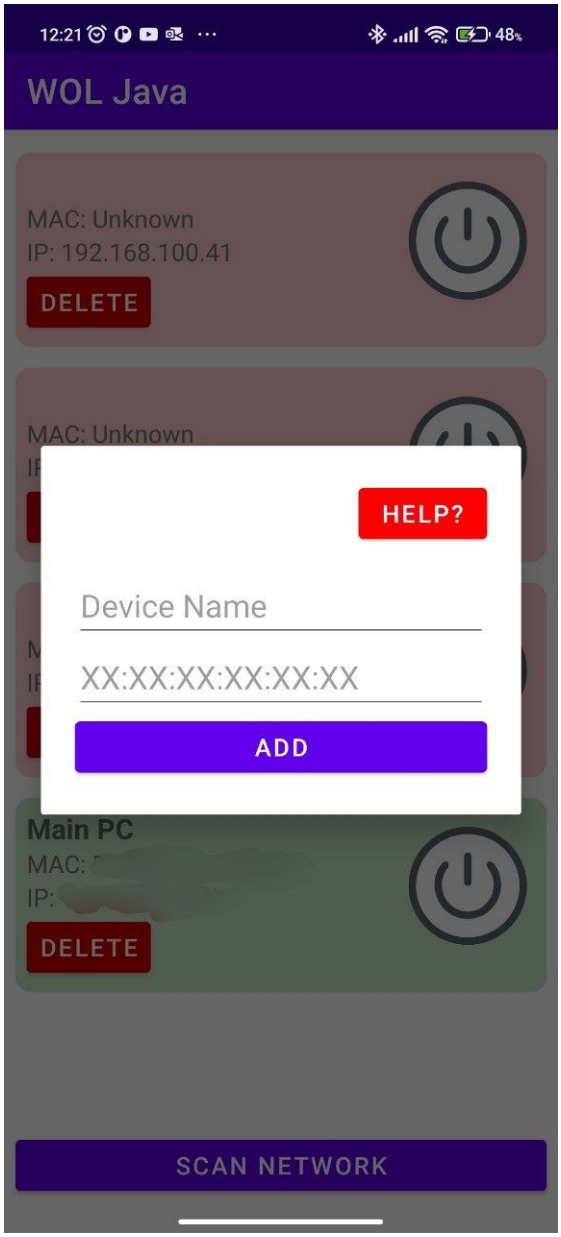

Рис. 4.3 Діалогове вікно додавання пристрою

Інструкція активації Wake-on-LAN

Вбудована підказка містить інструкцію з активації WoL на комп’ютері:

У BIOS/UEFI: необхідно активувати параметри "Wake on LAN", "Power On By PCI-E" та вимкнути ERP, щоб уникнути втрати живлення на мережевій карті.

У Windows: у диспетчері пристроїв слід відкрити властивості мережевого адаптера та дозволити пробудження пристрою за допомогою мережі (галочка "Allow this device to wake the computer").

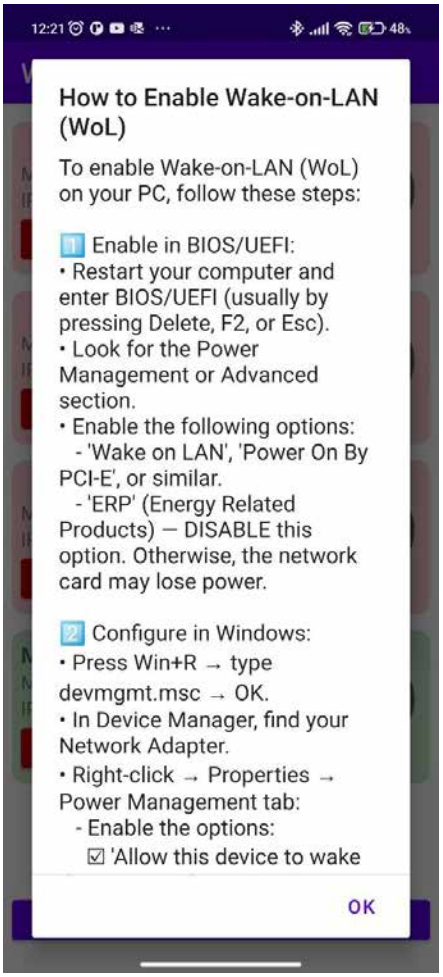

Рис. 4.4 Інструкція налаштування

Це забезпечує повну функціональність віддаленого пробудження через мобільний застосунок.

Виведення повідомлень

Після надсилання пакета користувач бачить коротке повідомлення типу "WOL packet sent" — це підтвердження успішної генерації й надсилання WoL-команди. У випадку помилки, інтерфейс повідомляє про неможливість виконання дії або вимагає коректного введення MAC-адреси.

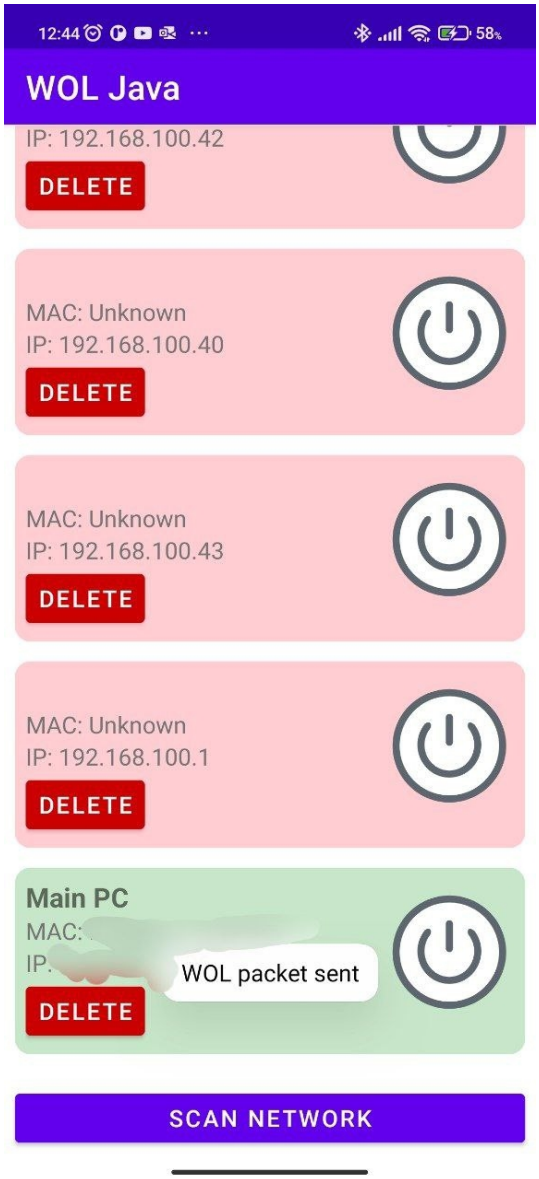

Рис. 4.5 Діалогові вікна

Ці повідомлення не лише підтверджують статус дій, але й виступають у ролі мікрофідбеку, забезпечуючи покращене UX-забезпечення для кінцевого користувача.

ВИСНОВОК

У межах реалізованого дослідження було створено повнофункціональну мобільну програмну систему, що забезпечує можливість дистанційного управління живленням комп’ютерних пристроїв у межах локальної мережі з використанням технології Wake-on-LAN. Розроблений Android-застосунок ґрунтується на механізмах широкомовного UDP-пакетного обміну та дозволяє ініціювати увімкнення пристроїв, перебуваючи з ними у спільному мережевому сегменті. Усі технічні рішення, реалізовані в рамках цього проєкту, були підібрані відповідно до сучасних вимог до безпеки, ефективності та автономності мобільних клієнтів, які функціонують в умовах гетерогенного мережевого середовища.

Проєктна мета, що полягала у створенні універсального мобільного інструменту для активації пристроїв під управлінням ОС Windows із клієнта на Android без серверної або хмарної компоненти, була реалізована в повному обсязі. Функціональність системи охоплює декілька ключових підсистем: сканування IP-діапазону підмережі /24 з виявленням активних хостів, збереження параметрів пристроїв у вбудованій базі даних SQLite, генерація та широкомовна передача WoL-пакетів (Magic Packet) відповідно до стандарту IEEE 802.3, а також створення локалізованого інтерфейсу користувача з підтримкою адаптації під різні розміри екранів та DPI.

Архітектурно застосунок побудовано на основі патерну MVVM, що забезпечує чітке розділення відповідальностей, зручність масштабування та супроводу проєкту. Застосування Android WorkManager та JobScheduler дає змогу делегувати фонові процеси без порушення принципів енергозбереження, характерних для сучасних мобільних платформ. Увесь мережевий функціонал реалізований із використанням java.net API, із забезпеченням обробки винятків та стабільної роботи.

У рамках тестування було підтверджено відповідність усім заявленим нефункціональним характеристикам: система демонструє здатність до сканування повного діапазону адрес менш ніж за 5 секунд, підтримує стабільну взаємодію з більш ніж 250 пристроями в активному списку. Автономність реалізації дозволяє функціонувати повністю в офлайн-сценаріях, що є критично важливим для застосування в ізольованих мережах або умовах з обмеженим доступом до інтернету.

Особливу увагу приділено забезпеченню практичного використання. Під час фінального етапу розробки було сформовано підписаний APK-файл, який дозволяє встановлення безпосередньо на пристрої кінцевих користувачів, в тому числі в середовищі без доступу до Play Store. Такий формат розповсюдження дає змогу інтегрувати застосунок у внутрішні IT-системи підприємств, розгорнути через MDM-рішення або надати доступ користувачам за допомогою внутрішніх порталів. Наявність готового інсталяційного пакету робить розробку практично придатною до масового впровадження.

Таким чином, результати роботи свідчать про повну відповідність поставленим цілям, високий рівень технічної реалізації, ефективність архітектурних і алгоритмічних рішень, а також значний потенціал до масштабування й подальшого розвитку. Зокрема, можливим є розширення функціональності за рахунок підтримки централізованого журналювання дій, авторизації користувачів за допомогою токенів або біометричних методів, інтеграції з хмарними платформами, додавання VPN-маршрутизації та впровадження аналітичних панелей для моніторингу історії взаємодій.

У цілому, дане рішення можна розглядати не лише як локальний інструмент для IT-адміністраторів, а й як технологічну основу для формування цілісних мобільних платформ управління енергоактивами в межах корпоративних мереж, що відкриває перспективи для прикладних досліджень та впроваджень у сфері автоматизації інфраструктурних процесів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android Studio – Wikipedia, The Free Encyclopedia [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Android_Studio
2. Couchbase Lite Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.couchbase.com/couchbase-lite
3. Firebase Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://firebase.google.com/docs
4. Guide to App Architecture – Android Developers Official Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.android.com/topic/architecture
5. Hipp, D. R. SQLite Technical Documentation [Електронний ресурс]. – SQLite.org. – Режим доступу до ресурсу: https://sqlite.org/docs.html
6. Magic Packet Technology – AMD White Paper №20213, Rev. A, листопад 1998. – 6 с.
7. ObjectBox Database [Електронний ресурс]. – Режим доступу до ресурсу: https://objectbox.io
8. Privacy changes in Android 10 (restricted access to /proc/net) – Android Developers Documentation, 2019 [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.android.com/about/versions/10/privacy/changes#proc-net-filesystem
9. Sommerville, I. Software Engineering (10th ed.) – Pearson Education, 2016. – ISBN: 978-1-292-09304-3
10. SQLite in Android: A Complete Guide – W3Resource [Електронний ресурс – Режим доступу до ресурсу: https://www.w3resource.com/sqlite/snippets/sqlite-android-guide.php

11. Taylor A. Wake On LAN: How It Works and Why It’s Useful [Електронний ресурс] – Cleverence Tech Blog, 29 січня 2023 р. – Режим доступу до ресурсу: https://www.cleverence.com/articles/tech-blog/wake-on-lan-how-it-works-and-why-it-s-useful/

12. Wake-on-LAN – Wikipedia, The Free Encyclopedia [Електронний ресурс]. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Wake-on-LAN

13. WakeOnLAN – Wireshark Wiki [Електронний ресурс]. – Режим доступу до ресурсу: https://wiki.wireshark.org/WakeOnLAN

14. Синяєв І. О. Програмне забезпечення системи моніторингу та активації пристроїв ос windows на ос android. Збірник наукових праць за матеріалами VII Всеукраїнської науково-практичної конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем 2025», м. Київ, 24 квітня. 2025 р. НУБІП України. Київ, 2025.

15. Як будувати UML-діаграми. [Електронний ресурс]. – Режим доступу: https://dou.ua/forums/topic/40575

ДОДАТОК А

База даних

Сторінок 2

Київ-2025


```
// Scanned Devices Table Create
private static final String CREATE_TABLE_SCANNED_DEVICES =
        "CREATE TABLE " + TABLE_SCANNED_DEVICES + " (" +
                "ip_address TEXT PRIMARY KEY, " +
                "mac_address TEXT, " +
                "name TEXT);";

// Logs Table Create
private static final String CREATE_TABLE_LOGS =
        "CREATE TABLE " + TABLE_LOGS + " (" +
                "id_logged INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "ip_address TEXT, " +
                "mac_address TEXT, " +
                "device_name TEXT, " +
                "status TEXT, " +
                "check_time TEXT);";

// Favorite Devices Table Create
private static final String CREATE_TABLE_FAVORITE_DEVICES =
        "CREATE TABLE " + TABLE_FAVORITES + " (" +
                "mac_address TEXT PRIMARY KEY, " +
                "ip_address TEXT, " +
                "device_name TEXT);";


public void logDatabaseContent() {
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor;

    // Scanned Devices
    cursor = db.rawQuery("SELECT * FROM " + TABLE_SCANNED_DEVICES, null);
    Log.d("DatabaseContent", "Scanned Devices Table:");
    while (cursor.moveToNext()) {
        String ip =
cursor.getString(cursor.getColumnIndexOrThrow("ip_address"));
        String mac =
cursor.getString(cursor.getColumnIndexOrThrow("mac_address"));
        String name = cursor.getString(cursor.getColumnIndexOrThrow("name"));
        Log.d("DatabaseContent", "IP: " + ip + ", MAC: " + mac + ", Name: " +
name);
    }
    cursor.close();

    // Logs
    cursor = db.rawQuery("SELECT * FROM " + TABLE_LOGS, null);
    Log.d("DatabaseContent", "Logs Table:");
    while (cursor.moveToNext()) {
        int id = cursor.getInt(cursor.getColumnIndexOrThrow("id_logged"));
        String ip =
cursor.getString(cursor.getColumnIndexOrThrow("ip_address"));
        String mac =
cursor.getString(cursor.getColumnIndexOrThrow("mac_address"));
        String name =
cursor.getString(cursor.getColumnIndexOrThrow("device_name"));
        String status =
cursor.getString(cursor.getColumnIndexOrThrow("status"));
```

```
        String time =
cursor.getString(cursor.getColumnIndexOrThrow("check_time"));
        Log.d("DatabaseContent", "ID: " + id + ", IP: " + ip + ", MAC: " +
```



```
mac +
                ", Name: " + name + ", Status: " + status + ", Time: " +
time);
    }
    cursor.close();

    // Favorites
    cursor = db.rawQuery("SELECT * FROM " + TABLE_FAVORITES, null);
    Log.d("DatabaseContent", "Favorite Devices Table:");
    while (cursor.moveToNext()) {
        String mac =
cursor.getString(cursor.getColumnIndexOrThrow("mac_address"));
        String ip =
cursor.getString(cursor.getColumnIndexOrThrow("ip_address"));
        String name =
cursor.getString(cursor.getColumnIndexOrThrow("device_name"));
        Log.d("DatabaseContent", "MAC: " + mac + ", IP: " + ip + ", Name: " +
name);
    }
    cursor.close();

    db.close();
```

- **ДОДАТОК Б**

Код програми

Сторінок 4

Київ-2025

```
public void addFavorite(Device device) {
    SQLiteDatabase db = dbManager.getWritableDatabase();
    ContentValues values = new ContentValues();
    Log.d("Device", "MAC: " + device.getMacAddress() + ", Name: " +
device.getDeviceName());
    values.put("mac_address", device.getMacAddress());
    values.put("ip_address", device.getIpAddress());
    values.put("device_name", device.getDeviceName());
    Log.d("Device", "MAC: " + device.getMacAddress() + ", Name: " +
device.getDeviceName());


    db.insertWithOnConflict(TABLE_FAVORITES, null, values,
SQLiteDatabase.CONFLICT_REPLACE);
    db.close();
}


btnConfirm.setOnClickListener(v -> {
    String enteredMac = macInput.getText().toString().trim();
    String enteredName = nameInput.getText().toString().trim();

    // Checking MAC format
    if (!enteredMac.matches("^([0-9A-Fa-f]{2}:){5}[0-9A-Fa-f]{2}$")) {
        Toast.makeText(this, "Invalid MAC address format",
Toast.LENGTH_SHORT).show();
        return;
    }

    // Setting MAC & Name
    device.setMacAddress(enteredMac);
    device.setDeviceName(enteredName.isEmpty() ? "Unknown" : enteredName); //
If name empty - set "Unknown"

    // Add device into Favorites
    favoritesRepository.addFavorite(device);

    // Showing message
    Toast.makeText(this, "Added to favorites", Toast.LENGTH_SHORT).show();

    // Close dialog
    dialog.dismiss();

    // Update favorite list
    loadFavoriteDevices();
});


private void startNetworkScan() {
    Log.d("NetworkScan", "Starting network scan...");
    Toast.makeText(this, "Scanning network...", Toast.LENGTH_SHORT).show();

    NetworkScanner.getLocalSubnet(this, new NetworkScanner.ScanListener() {
        @Override
        public void onScanCompleted(List<Device> devices) {
            Log.d("NetworkScan", "Scan completed. Found devices: " +
devices.size());
            runOnUiThread(() -> {
```

```
                updateDeviceListWithScanResults(devices);
                Toast.makeText(MainActivity.this, "Scan completed!",
Toast.LENGTH_SHORT).show();

                //Sending result into DB
                DatabaseManager databaseManager = new
DatabaseManager(MainActivity.this);
                databaseManager.logDatabaseContent();
            });
        }
    });
}


holder.btnWake.setOnClickListener(v -> {
    Executors.newSingleThreadExecutor().execute(() -> {
        try {
            new WOLManager().sendWOLPacket(device.getMacAddress());
            WOLManager.showToast(context, "WOL packet sent");
        } catch (Exception e) {
            Log.e("WOL", "Failed to send WOL packet", e);
            WOLManager.showToast(context, "WOL failed: " + e.getMessage());
        }
    });
});

holder.btnDelete.setOnClickListener(v -> {
    String mac = device.getMacAddress();
    if (mac != null && !mac.isEmpty()) {
        favoritesRepository.removeFavorite(mac);
        deviceList.remove(position);
        notifyItemRemoved(position);
        notifyItemRangeChanged(position, deviceList.size());
        Toast.makeText(context, "Device deleted", Toast.LENGTH_SHORT).show();
    }
});


public void sendWOLPacket(String macAddress) throws Exception {
    byte[] macBytes = getMacBytes(macAddress);
    byte[] packetBytes = new byte[6 + 16 * macBytes.length];

    // Header: 6 bytes 0xFF
    for (int i = 0; i < 6; i++) {
        packetBytes[i] = (byte) 0xFF;
    }

    // Repeat MAC 16 times
    for (int i = 6; i < packetBytes.length; i += macBytes.length) {
        System.arraycopy(macBytes, 0, packetBytes, i, macBytes.length);
    }

    InetAddress broadcastAddress = getBroadcastAddress();
    if (broadcastAddress == null) {
        throw new Exception("No broadcast address found");
    }

    DatagramPacket packet = new DatagramPacket(packetBytes,
packetBytes.length, broadcastAddress, 9);
```

```
    DatagramSocket socket = new DatagramSocket();
    socket.setBroadcast(true);

    socket.send(packet);

    socket.close();

    Log.d(TAG, "WOL packet sent to " + macAddress + " via " +
broadcastAddress.getHostAddress());
}


public static void scanNetwork(Context context, ScanListener listener, String
networkPrefix) {
    new Thread(() -> {
        List<Device> devices = new ArrayList<>();
        DatabaseManager db = new DatabaseManager(context);

        ExecutorService executor = Executors.newFixedThreadPool(127);
        CountDownLatch latch = new CountDownLatch(254);

        for (int i = 1; i <= 254; i++) {
            String ip = networkPrefix + "." + i;

            executor.execute(() -> {
                try {
                    if (pingDevice(ip)) {
                        synchronized (devices) {
                            devices.add(new Device(null, "Unknown", ip, null,
false));
                        }

                        synchronized (db) {
                            db.getWritableDatabase().execSQL(
                                    "INSERT OR REPLACE INTO scanned_devices
(ip_address, mac_address, name) VALUES (?, NULL, NULL);",
                                    new Object[]{ip}
                            );

                            db.getWritableDatabase().execSQL(
                                    "INSERT INTO logs (ip_address,
mac_address, device_name, status, check_time) VALUES (?, NULL, NULL, ?,
datetime('now'));",
                                    new Object[]{ip, "reachable"}
                            );
                        }
                    }
                } finally {
                    latch.countDown();
                }
            });
        }

        try {
            latch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
        executor.shutdown();
        db.close();

        Log.d("NetworkScan", "Scan completed. Found " + devices.size() + "
devices.");
        listener.onScanCompleted(devices);
    }).start();
}


private static boolean pingDevice(String ip) {
    try {
        InetAddress inetAddress = InetAddress.getByName(ip);
        boolean reachable = inetAddress.isReachable(200);
        Log.d("NetworkScan", "Ping to " + ip + " reachable: " + reachable);
        return reachable;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}
```