КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ-КОМПАНЬЙОНА ДЛЯ СИНХРОНІЗАЦІЇ З ПК

Виконав студент 4-го курсу
Тарас МИРОНЮК

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат наук,
Євгеній ІВАНОВ

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«25» травня 2022 р.,
протокол № 10
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 43 сторінок, 8 ілюстрацій, 1 таблиця, 12 джерел посилань.

ANDROID РОЗРОБКА, QT РОЗРОБКА, МОБІЛЬНИЙ ЗАСТОСУНОК, МОДЕЛЬ МЕРЕЖЕВОЇ ВЗАЄМОДІЇ, СИНХРОНІЗАЦІЯ ПОВІДОМЛЕНЬ, СИНХРОНІЗАЦІЯ ПРИСТРОЇВ У ЛОКАЛЬНІЙ МЕРЕЖІ, СИНХРОНІЗАЦІЯ СПОВІЩЕНЬ, ШИФРУВАННЯ З ВІДКРИТИМ КЛЮЧЕМ.

Метою роботи є створення мобільного застосунку для синхронізації з персональним комп’ютером в локальній мережі.

Об’єктом роботи є процес безпечного обміну інформацією між пристроями в межах локальної мережі. Предметом роботи є створення мобільного застосунку для синхронізації з персональним комп’ютером за допомогою локальної мережі.

Інструменти розроблення: мова програмування C++ та фреймворк Qt з відповідним пакетом інструментів, включно з Qt Creator і Qt Design Studio, а також інтегроване середовище розробки Android Studio та мова програмування Java.

Результати роботи: Проаналізовано найпоширеніші існуючі аналоги KDE Link та Windows Phone Link, визначено їх переваги та недоліки. Показано, що вони мають багато надлишкового функціоналу, який витрачає ресурси системи та навантажує інтерфейс користувача. Розроблено застосунок для синхронізації мобільного пристрою з персональним комп’ютером по локальній мережі, основними функціями якого є синхронізація сповіщень та повідомлень.

Розроблений застосунок може застосовуватися для синхронізації робочого процесу, підвищення впорядкованості та покращення організації роботи при одночасному зменшенні психологічного навантаження на працівників в умовах хаотичного надходження інформації, спричиненого переходом на дистанційні форми роботи в багатьох галузях у зв’язку з епідемією COVID-19.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AOT – Ahead-of-time compilation, компіляція перед виконанням;

API – Application Programming Interface, прикладний програмний інтерфейс;

IDE – Integrated Design Environment, інтегроване середовище розробки;

IPC – Inter Process Communication, міжпроцесний зв’язок;

JIT – Just-in-time compilation, компіляція при виконанні, “саме вчасно”;

QML – Qt Modeling Language;

RSA – Rivest–Shamir–Adleman, криптографічний алгоритм з відкритим ключем;

SMS – Short Message Service, Служба коротких повідомлен;

SSL – Secure Sockets Layer, Рівень захищених сокетів;

TCP – Transmission Control Protocol, Протоко́л керува́ння передаванням;

TLS – Transport Layer Security, Протокол захисту транспортного рівня;

UDP – User Datagram Protocol, Протокол датаграм користувача.

ОС – Операційна система;

ПК – Персональний комп’ютер;

ВСТУП

Оцінка сучасного стану об’єкта розробки. У зв’язку з подіями останніх років, робочій процес значною мірою перейшов у дистанційний режим. При цьому його організація почасти дозволяє бажати кращого. Не рідкісною є ситуація, коли координація робочого процесу відбувається за допомогою планувальника, декількох месенджерів та додатку для конференцій. Далеко не завжди можливо встановити всі ці застосунки на робочий комп’ютер.

Застосунки для синхронізації мобільних пристроїв з ПК існують щонайменше з 2014 року [1]. Наразі на більшість популярних платформ доступні додатки з подібним функціоналом. Більшість з них розроблялись для бездротової передачі файлів – переважно фото та відео – між пристроями в межах локальної мережі. Деякі за час розробки отримали додаткові функції, такі як дистанційне керування пристроєм, трансляція екрану чи синхронізація сповіщень. Саме остання функція набуває актуальності сьогодні.

Сповіщення – це повідомлення, яке операційна система відображає за межами інтерфейсу додатка, щоб нагадати користувачеві про певну подію, показати повідомлення від інших людей або іншу важливу інформацію з програми. На різних операційних системах, користувачі можуть скористатися сповіщенням, щоб відкрити застосунок, або виконати дію безпосередньо зі сповіщення.

Так склалося, що більшість сервісів мають у тій чи іншій формі мобільний додаток. З огляду на вищенаведені фактори, пропонується рішення поставленої проблеми шляхом розробки застосунку для синхронізації сповіщень між мобільним пристроєм та персональним комп’ютером по локальній мережі.

Саме цим шляхом рухаються популярні застосунки для синхронізації. Так, разом з Windows 11 компанія Microsoft презентувала оновлення Phone Link, що зосередилось саме на взаємодії зі сповіщеннями [2]. А проект KDE у 2021 році відновив роботу над додатком для IOS через сильний запит користувачів [1].

Однак, як буде показано в цій роботі, обидва рішення мають недоліки та обмеження, пов’язані з їх попереднім напрямком розробки.

Актуальність роботи та підстави для її виконання. У зв’язку з епідемією COVID-19 в багатьох галузях відбувається перехід на дистанційні форми роботи. Через відсутність широковживаного програмного забезпечення для синхронізації робочого процесу, робота часто вимагає одночасного використання декількох різних сервісів для отримання текстової інформації, подекуди на різних пристроях.

Оскільки більшість сервісів мають в тій чи іншій формі мобільний додаток, створення застосунку для синхронізації смартфону з персональним комп’ютером є дієвим та перевіреним рішенням цієї актуальної проблеми.

Мета й завдання роботи. Метою кваліфікаційної роботи є створення мобільного застосунку для синхронізації з персональним комп’ютером в локальній мережі. Для досягнення цієї мети поставлено такі завдання:

- Провести дослідження аналогів, визначити їх переваги та недоліки.
- Розробити технічне завдання для застосунку.
- Визначити оптимальний набір технологій та інструментів розробки.
- Спроектувати модель взаємодії між пристроями в мережі.
- Розробка та тестування прототипу системи.
- Застосувати можливості обраних інструментів для оптимізації застосунку.

Об'єкт, методи й засоби розроблення. Об’єктом розроблення мобільного застосунку для синхронізації з ПК є процес безпечного обміну інформацією між пристроями в межах локальної мережі.

Розробці програмного засобу передувало дослідження відомих аналогів застосунку, аналіз їх переваг та недоліків та розробка технічного завдання на основі зібраних даних.

В якості інструменту для розробки мобільної частини було обрано Android Stuido – інтегроване середовище розробки (IDE) для платформи Android від

компанії Google. Мовою розробки обрано Java, яка надає великий перелік низькорівневих інтерфейсів, необхідних для реалізації функціоналу програми.

Для розробки настільної частини було використано фреймворк Qt, який є популярним рішенням для розробки графічного інтерфейсу. Модуль Qt Quick дозволяє інтегрувати в застосунок, написаний мовою QML, код на C++, що значно покращує продуктивність програми, зберігаючи легкість розробки [3]. Також можливості компіляторів Qt 6.3 дозволяють ще краще оптимізувати застосунок, завдяки автоматичній компіляції класів QML в C++.

Можливі сфери застосування. Розроблений застосунок призначений для підвищення продуктивності праці при роботі в дистанційному режимі, агрегуючи всю необхідну працівникові інформацію в одному застосунку.

1 РОЗРОБКА ТЕХНІЧНОГО ЗАВДАННЯ ЗАСТОСУНКУ

1.1 Огляд існуючих рішень

1.1.1 KDE connect

KDE Connect – це проект, метою якого є дозволити пристроям користувача спілкуватися один з одним [1]. Він є частиною більшої екосистеми додатків для ОС Linux (K Desktop Environment). Ось досить стислий перелік функцій, що надає KDE Connect:

- Можливість отримувати сповіщення зі свого телефону на ПК і відповідати на повідомлення;
- Керувати музикою, що відтворюється на ПК, зі свого телефону;
- Використовувати свій телефон як пульт дистанційного керування для ПК;
- Виконувати попередньо визначені команди на своєму ПК з підключених пристроїв;
- Перевірити рівень заряду акумулятора телефону на робочому столі;
- Подзвонити на свій телефон, щоб знайти його;
- Обмін файлами та посиланнями між пристроями;
- Транслювати екран телефону на JIT;
- Керувати гучністю ПК за допомогою телефону;
- Надсилати SMS зі свого робочого столу.

Застосунок KDE Connect доступний чи портований на більшість платформ. Він доступний у вигляді пакету для всіх популярних дистрибутивів Linux, а також портований на Windows, і доступний для завантаження з Microsoft Store.

Додаток для Android можна знайти як у магазині Google Play, так і в безкоштовному відкритому магазині F-Droid. Розробка клієнтської програми KDE Connect для iOS була запущена в 2014 році, проте не була завершена, поки

її знову не запустили у 2018–2019 роках з парою виправлень і підтримкою TLS [1]. У 2021 році було розпочато масштабний проект по оновленню більшої частини старого коду на базі нових фреймворків для можливостей подальшого розширення. KDE Connect також доступний для телефонів на базі Linux (Plasma Mobile, PostmarketOS, тощо).

KDE Connect складається з двох частин. Програма для ПК і додаток для телефону. Щоб працювати, обидва пристрої повинні бути в одній локальній мережі. Спершу пристрої необхідно спарувати. Програма самостійно просканує мережу, і запропонує користувачу обрати свій пристрій зі списку (рис. 1).

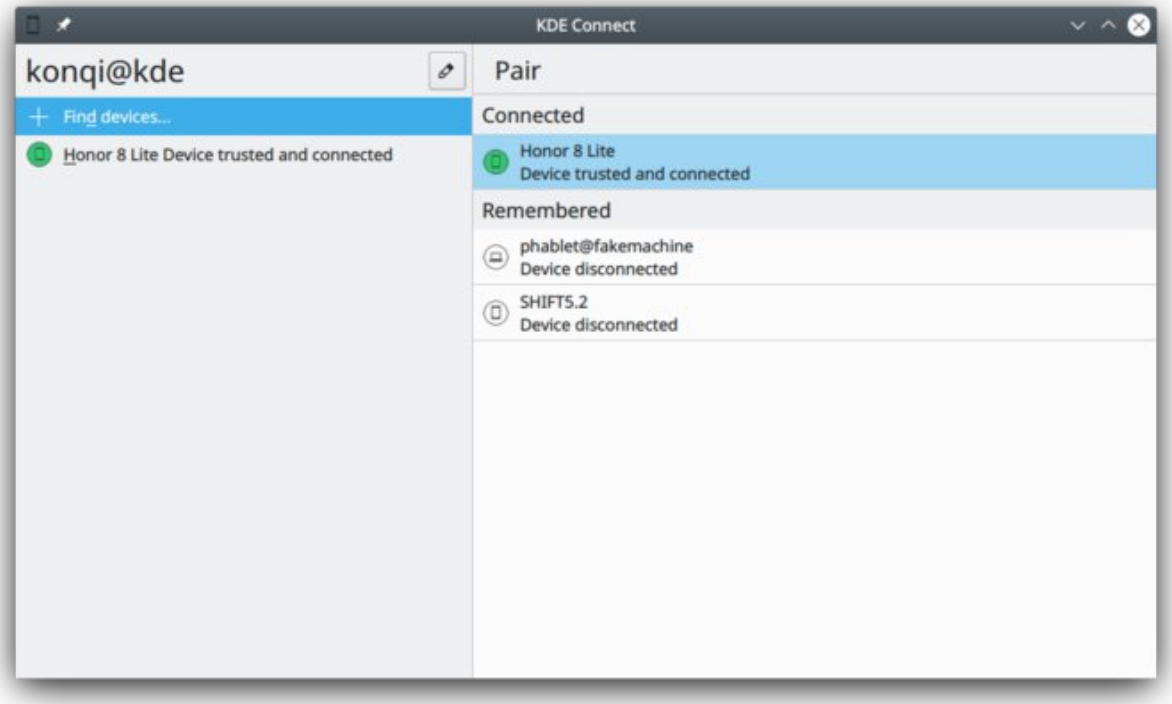

Рисунок 1 – Інтерфейс підключення пристроїв KDE Connect

Хоча не має обмеження на кількість спарованих пристроїв, одночасно можна під’єднати тільки один. Цікавою особливістю цього застосунку є те, що роль сервера в системі виконує ПК, хоча в інших аналогах ця роль відводиться смартфону. Це пов’язано з великою кількістю додаткових функцій застосунку, як от відтворення відео зі смартфона на ПК, чи дистанційне керування мультимедіа з телефона. Побічним ефектом є можливість з’єднати за допомогою цієї програми

два настільні комп’ютери. І хоча в такій конфігурації більшість функціоналу не працює, у деяких ситуаціях така можливість може стати у нагоді.

Інтерфейс застосунку, зображений на рис. 2, виконаний у простому стилі, проте він ховає під собою безліч різних функцій, налаштувань, гарячих клавіш для них та можливостей для персоналізації досвіду використання програми. Після тривалого користування виникає розуміння, що інтерфейс сильно перевантажений, і потребує повного переосмислення.

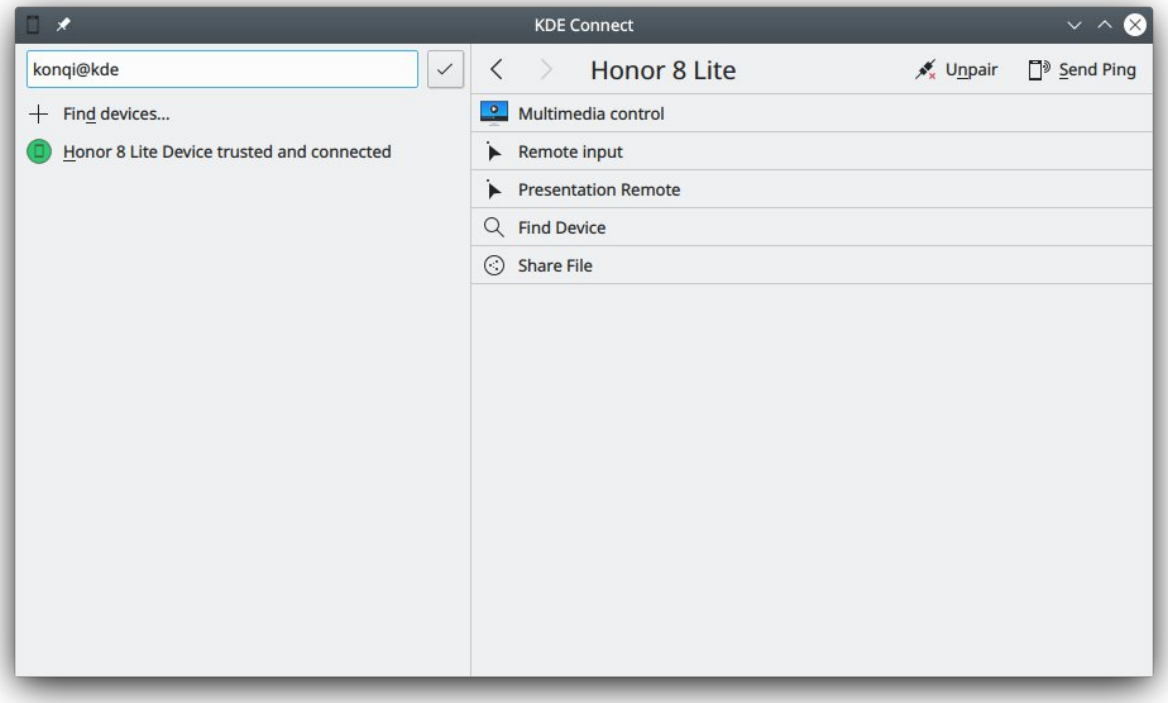

Рисунок 2 – Головна сторінка KDE Connect

Значну частину інтерфейсу, як видно на рис. 3, займає менеджер сховища, який дозволяє передавати файли між пристроями, але також повноцінно виконує всі функції файлового менеджеру. Окремо варто зазначити можливість інтеграції хмарного сховища та функцію кошику, з можливістю пам’яті.

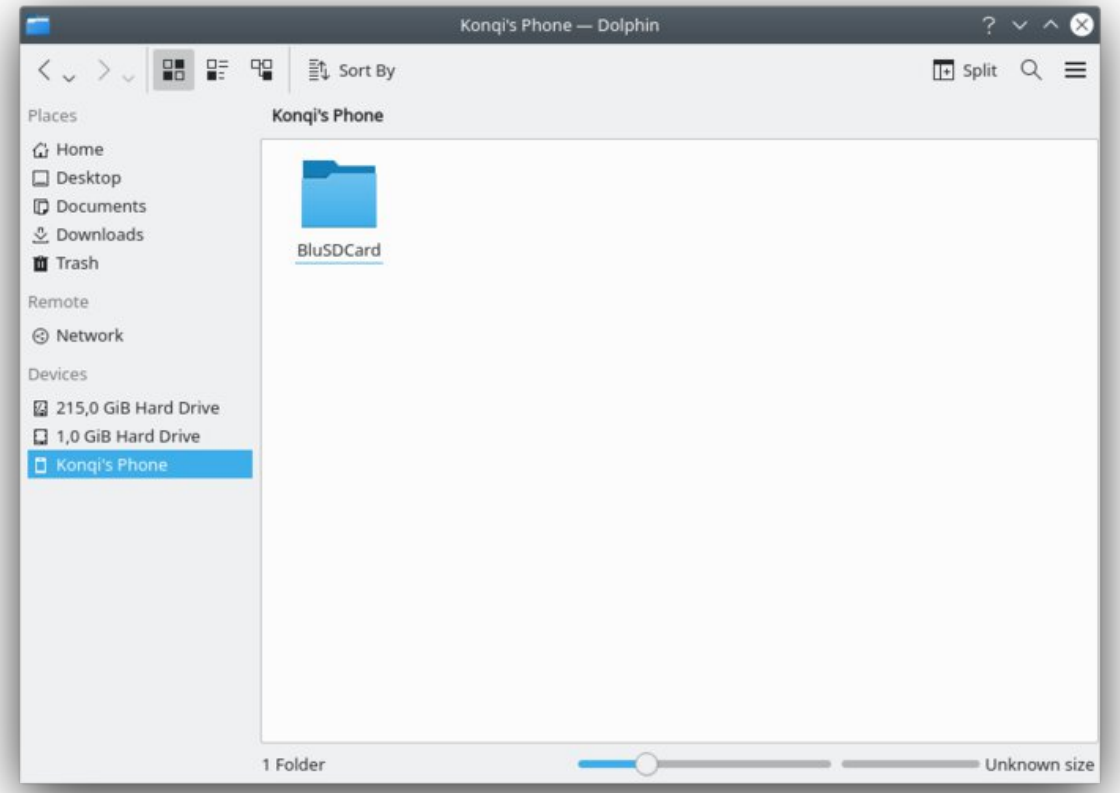

Рисунок 3 – Менеджер пам’яті KDE Connect

І все ж, незважаючи на такий значний функціонал застосунку, більшість інструментів які він надає, є настільки специфічними, що навряд чи будуть використані більше одного разу, і тільки для рішення дуже неочевидних задач. Під час використання є враження, що деякі функції програми були додані виключно тому, що була можливість їх реалізувати, без розуміння навіщо вони потрібні користувачу. Ціною такого підходу став громіздкий і неочевидний інтерфейс.

1.1.2 Windows Phone Link

Windows Phone Link – це застосунок для синхронізації робочого процесу між смартфоном і комп’ютером, розроблений компанією Microsoft. Він

доступний на комп’ютерах з ОС Windows 10, версії від травня 2020 року, або новішої, та мають принаймні 8 ГБ оперативної пам’яті, разом зі смартфоном з ОС Android 11 [2]. Застосунок постачається разом з останніми версіями операційної системи Windows, та широко застосовує останні функції Universal Windows Platform, що є одним з чинників його популярності.

Завданням застосунку розробники визначають підвищення продуктивності для людей, які проводять більшу частину дня за робочим комп’ютером, але все ще повинні стежити за своїми телефонами [2, 4]. Після його належного налаштування, користувач може переглядати текстові повідомлення, сповіщення та фотографії зі свого Android смартфону, відповідати на повідомлення, сповіщення та виклики, не торкаючись телефону – безпосередньо на робочому столі. Варто зазначити, що, в оригінальному задумі розробників, основним функціоналом застосунку мав стати швидкісний доступ до мультимедійних файлів на телефоні. Причиною було рішення Google про припинення підтримки USB Picture Transfer Protocol після Android 11. Проте, з відгуків користувачів, стало зрозуміло, що запит був саме на подальший розвиток функцій для віддаленої взаємодії зі смартфоном [4].

Разом з виходом ОС Winodws 11, застосунок отримав оновлення, що додало такі функції, як сортування та фільтри для сповіщень [2]. З цим оновленням також з’являється новий інтерфейс, закруглені кути, нова область сповіщень, а також деякі нові опції в налаштуваннях. З’явилася опція під’єднання до нового комп’ютера, просканувавши QR-код.

Інтерфейс програми (рис. 4) розділений на 3 секції, з яких ліва містить всю інформацію про телефон та сповіщення. Виводиться заряд акумулятора та показники мережі пристрою. Також присутні 4 кнопки для керування телефоном – перемикач режиму “не турбувати”, перемикач Bluetooth, регулятор гучності та медіаплеєр.

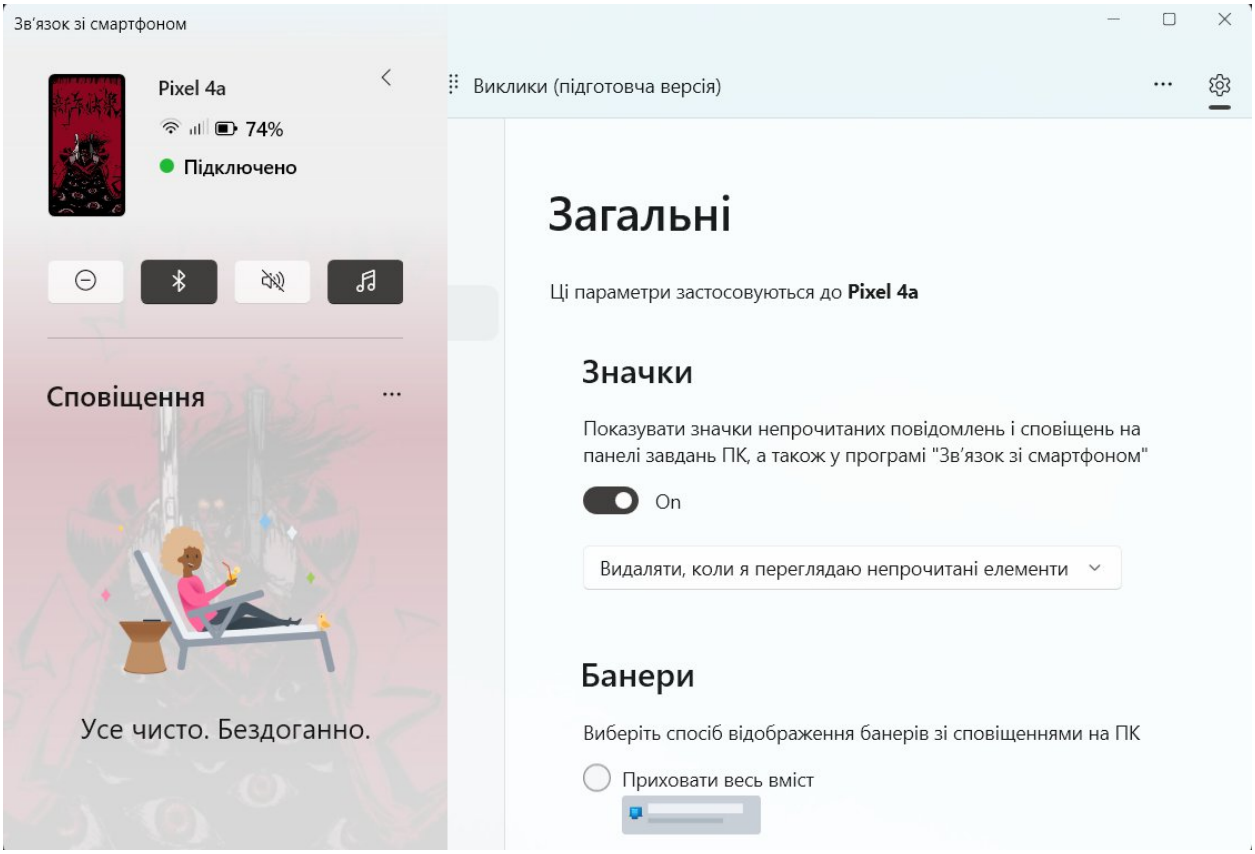

Рисунок 4 – Інтерфейс Widnows Phone Link

Сповіщення. Певне, основна функція застосунку Phone Link, – це синхронізація сповіщень. Після підключення вашого пристрою Android сповіщення з нього з'являться на вашому ПК з Windows. У деяких випадках можна виконати дію зі сповіщенням, наприклад відповісти на повідомлення, просто з робочого столу. Коли ви відхиляєте сповіщення в Phone Link, воно також відхиляється на телефоні.

Повідомлення. Другою важливою особливістю є можливість отримувати текстові SMS повідомлення та відповідати на них на комп'ютері з Windows. Це працює практично так само, як і синхронізація сповіщень. Основна відмінність полягає в тому, що повідомлення відображаються на власній вкладці у додатку Phone Link.

Фотографії. Наступна вкладка в додатку Phone Link призначена для фотографій. Тут можна переглянути фотографії зі свого пристрою Android. Що

ще важливіше, їх можна легко зберегти їх на ПК і навіть просто перетягнути їх в інші програми Windows.

Дзвінки. Оскільки програма Phone Link може підключатися до вашого телефону через Bluetooth, ваш ПК може по суті виконувати роль гарнітури Bluetooth. Це дає змогу здійснювати та отримувати дзвінки з Windows, коли пристрій Android підключено.

Загалом інтерфейс застосунку є зручним та продуманим, а доступний функціонал повністю достатнім. Проте з ним також виникають проблеми. Зокрема через бажання розробників застосувати максимум можливостей ще не завершених інструментів Universal Windows Platform, сповіщення додатку є незручними та непередбачуваними.

Переваги та недоліки розглянутих застосунків підсумовано в табл. 1.

Таблиця 1 – Переваги та недоліки застосунків KDE Connect та Phone Link

Назва функції	KDE Connect	Phone Link	Важливість функції
Реєстрація декількох пристроїв	+	+	необхідна
Сповіщення	+	+	необхідна
Повідомлення	+	+	необхідна
Дзвінки	+	+	помірно
Фотографії	+	+	надлишкова
Файли	+	-	надлишкова
Дистанційне керування телефоном	+	+	помірно
Дистанційне керування ПК	+	-	надлишкова
Трансляція екрану	+	-	надлишкова
Інформація про стан телефону	+	+	помірно

Таким чином, основним функціоналом застосунку визначено функції синхронізації сповіщень та повідомлень.

1.2 Опис технічного завдання проекту

На основі проведеного аналізу, технічне завдання до майбутнього застосунку включає:

Платформа

- Застосунок складається з двох частин: мобільної (Android) та настільної (Windows).
- Бажаною є реалізація частини для ПК з використанням кросплатформенних інструментів.
- Мобільний пристрій виступає сервером, ПК – клієнтом.
- Всі комунікації відбуваються по захищеному каналу.

Функціонал

- Можливість реєстрації (парування) декількох пристроїв.
- Сповіщення з мобільного пристрою відображаються на робочому столі. Передбачена можливість з ними взаємодіяти.
- Повідомлення з мобільного пристрою відображаються на робочому столі. Передбачена можливість дати відповідь на них.
- Передбачена можливість побачити інформацію про стан телефону (заряд батареї, мережа).
- Функції для дистанційного керування телефоном (Керування медіа, режим “Не турбувати”).

2 ПРОЕКТУВАННЯ ЗАСТОСУНКУ

2.1 Вибір технологій для реалізації функціоналу застосунку

Фреймворки грають на користь ефективної розробки додатків через готові компоненти. Їхня мета – допомогти стандартизувати процес розробки програмного забезпечення, зробивши його швидшим і простішим. Qt Framework є популярним рішенням для розробки графічного інтерфейсу [5].

Використання програми на Qt можливо з будь-якою популярною операційною системою, включаючи як настільні, так і мобільні платформи: Windows, Linux, macOS, iOS, Android та інші. Qt містить специфічні для платформи додаткові функції, які дозволяють створити графічний інтерфейс, максимально наближений до рідного графічного інтерфейсу.

Qt має свій власний конструктор, спроектувати гарний графічний інтерфейс з ним досить просто. Бібліотека Qt Quick полегшує проектування інтерфейсу користувача, порівняно з традиційним підходом (C або C++) [3][6]. Нові бібліотеки Qt Controls і Qt Layouts ще більше скорочують час розробки. Ці модулі забезпечують елементи керування інтерфейсом користувача та макети.

При цій простоті, Qt має ряд беззаперечних переваг перед більшістю альтернатив:

- QML використовує GPU для прискорених обчислень.
- Зрештою, Qt написаний на C++ і працює на підтримуваних платформах.
- Компоненти QML написані на C++.
- Можливість реалізувати логіку програми за межами інтерфейсу інтерфейсу на C++ для підвищення ефективності.
- QML використовує JIT і AOT компілятори.

Останній пункт особливо цікавий, оскільки з останньою версією Qt 6.3, компілятор може самостійно обирати оптимальний підхід для кожного модуля QML [7].

Слід також додати, що оглянутий раніше KDE Connect створено за допомогою фреймворку Qt.

Що стосується мобільної частини застосунку, використання Qt є очевидно надлишковим, оскільки для нього не передбачається об’ємного графічного інтерфейсу. Навпаки, застосунок відіграє роль серверного процесу заднього плану. Також слід зазначити що такі фреймворки як Qt не надають низькорівневих інструментів, необхідних для наприклад відслідковування сповіщень. Таким чином, послідовним вибором технології для мобільної частини застосунку є звичайний Java Android застосунок.

2.2 **Проектування клієнт-серверної взаємодії**

У запропонованій системі, мобільний застосунок виступає сервером. До нього під’єднуються клієнти – ПК. Для простоти розглянемо систему, в якій до сервера в один момент під’єднано одного клієнта. Тобто, при під’єднанні нового клієнта, сервер розриває підключення з попереднім, якщо такий був.

Система має хоча б три канали комунікації: канал пошуку, канал ініціалізації з’єднання і робочі канали. Також система має два стани: нормальний стан і стан очікування.

2.2.1 Канал пошуку

Реалізований як зарезервований UDP порт, який використовується клієнтами для пошуку активних серверів у мережі. Канал активний завжди. Щоб отримати список активних серверів, клієнт надсилає широкомовний запит, і отримує від кожного активного сервера відповідь. Перебуваючи в стані очікування, клієнт з певною періодичністю здійснює пошук активних серверів. Сервер відповідає на ці запити тільки якщо він також перебуває в стані очікування. Визначені наступні види повідомлень:

- Запит активних серверів (широкомовний запит);
- Відповідь сервера.

2.2.2 Канал ініціалізації з’єднання

Захищений канал, по якому передається системна інформація під час встановлення з’єднання. Канал активний тільки коли пристрої перебувають у режимі підключення. У режимі очікування сервер очікує запити по цьому каналу. Визначені наступні види повідомлень:

- Запит на реєстрацію на сервері;
- Запит на з’єднання з сервером;
- Повідомлення про успіх реєстрації;
- Повідомлення про успіх автентифікації;
- Повідомлення про помилку реєстрації;
- Повідомлення про помилку автентифікації.

2.2.3 Робочі канали

Для кожного підключеного клієнта, сервер має окремий захищений робочій канал. Ці з’єднання активні як в стані очікування, так і в нормальному стані. В нормальному стані, передача даних між сервером та клієнтами відбувається саме через робочі канали. Визначені види повідомлень:

- Отримане нове сповіщення
- Внесені зміни до сповіщення
- Сповіщення скасовано
- Дія зі сповіщенням
- Мультимедійна дія

2.3 Алгоритми підключення пристроїв

Пристрій входить в стан очікування якщо він не має жодного активного робочого каналу. Очевидно, очікування є початковим станом додатку.

Щоб переконатися в безпеці передачі даних, необхідно вирішити дві проблеми:

Проблема передачі даних. Необхідно шифрувати дані, що передаються по робочому каналу. Для цього достатньо гарантувати захищений канал між пристроями. Для більшості імплементацій рекомендовано використовувати протокол SSL.

Проблема довіри до пристрою. Необхідно підтвердити, що пристрій до якого ми під'єднуємось є тим, за кого себе видає. Пропонується наступне рішення. При створенні пари, сервер генерує псевдовипадкове число, і користувач вручну вводить його в інтерфейс користувача. Таким чином користувач самостійно підтверджує валідність сервера на першому етапі, а сервер має спосіб підтвердити клієнта. Тут же клієнт генерує пару ключів RSA і відправляє публічний ключ на сервер для подальшої автентифікації. Використовувати технологію цифрового підпису на цьому етапі вважаємо надлишковим, через низьку криптостійкість ключа такої довжини, що може ввести користувач.

При подальших автоматичних з'єднаннях, клієнт подаватиме запит на з'єднання, разом з цифровим підписом RSA, за допомогою якого сервер автентифікує клієнта. Рекомендовано видаляти зареєстрованих клієнтів після періоду неактивності.

Підключення нових пристроїв відбувається згідно наступних алгоритмів:

Реєстрація нового пристрою

1. Сервер генерує випадкове шестизначне число.
2. Користувач вибирає сервер зі списку та вводить це число в інтерфейсі.
3. Клієнт генерує відкритий та закритий ключ RSA.

4. Клієнт відправляє запит на сервер, що містить введене число та відкритий ключ.
5. Якщо отримане шестизначне число відмінне від згенерованого, сервер відповідає повідомленням про помилку реєстрації, генерує нове число.
6. Інакше, сервер вносить клієнта до списку зареєстрованих пристроїв.
7. Сервер відправляє повідомлення про успіх реєстрації.
8. Клієнт реєструє сервер та записує свій приватний ключ.
9. Клієнт ініціює підключення до сервера.

Підключення відомого пристрою

1. Клієнт визначає, що сервер під’єднаний до мережі та ініціює з’єднання.
2. Клієнт генерує запит, підписаний раніше записаним приватним ключем RSA для цього підключення.
3. Якщо сервер не впізнає клієнта, він відповідає повідомленням про помилку автентифікації.
4. Інакше, сервер перевіряє валідність підпису за допомогою збереженого відкритого ключа.
5. Якщо підпис дійсний, сервер встановлює з’єднання з клієнтом.
6. Інакше, відповідає повідомленням про помилку автентифікації.

3 ОПИС РОЗРОБКИ СЕРВЕРНОГО ANDROID ЗАСТОСУНКУ

3.1 Огляд сповіщень в ОС Android та Notification API

Сповіщення відображаються користувачеві в різних місцях і в різних форматах, наприклад, піктограма в рядку стану, детальніший запис у панелі сповіщень, як значок на значку програми (рис. 5) та на прив’язаних пристроях для носіння автоматично [8].

Рисунок 5 – Піктограма сповіщення в рядку стану

Сповіщення в найпростішій і компактній формі (також відомій як згорнута форма) відображає піктограму, заголовок і невелику кількість тексту вмісту, що видно на рис. 6.

Починаючи з Android 5.0, сповіщення можуть на короткий час з’являтися у плаваючому вікні, як на рис. 7, це називається Heads-up сповіщенням. Зазвичай така поведінка стосується важливих повідомлень, про які користувач повинен знати негайно, і з’являється, лише якщо пристрій розблоковано.

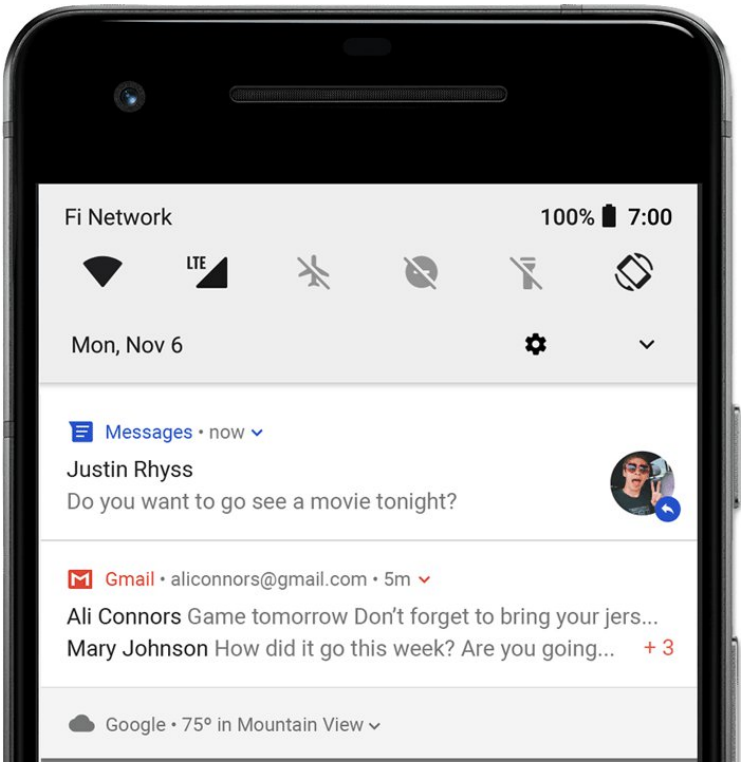

Рисунок 6 – Згорнуте сповіщення

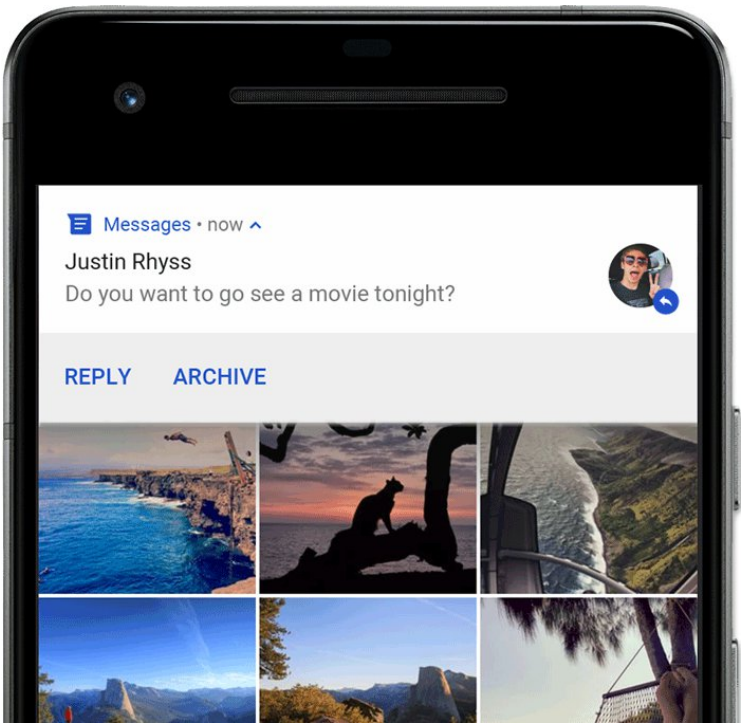

Рисунок 7 – Heads-up сповіщення

Щоб уникнути бомбардування користувачів багатьма зайвими сповіщеннями, передбачена можливість оновити наявне сповіщення, а не публікувати нове, або розглянути можливість використання сповіщень у стилі «Повідомлення», щоб відображати оновлення розмов. Однак, якщо необхідно доставити кілька сповіщень, слід розглянути можливість групування цих окремих сповіщень в групу (доступно для Android 7.0 і новіших версій). Група сповіщень дозволяє згорнути кілька сповіщень в єдину публікацію на панелі сповіщень із списком вкладень. Потім користувач може розгорнути сповіщення, щоб розкрити деталі кожного окремого сповіщення. Застосунок може поступово розширювати групу сповіщень і редагувати кожне сповіщення в ній, для надання користувачеві додаткової інформації.

Починаючи з Android 8.0, кожне сповіщення має бути призначене для певного каналу застосунку, інакше воно не відображатиметься. Розділивши сповіщення на канали, користувачі можуть вимкнути певні канали сповіщень для програми (замість вимкнення всіх сповіщень), а також мають змогу керувати візуальними та звуковими параметрами для кожного каналу. На пристроях з ОС Android 7.1 і нижчих версії користувачі можуть керувати сповіщеннями лише для кожної програми (фактично кожна програма має лише один канал на Android 7.1 і старіших версіях).

Один додаток може мати кілька каналів сповіщень – окремий канал для кожного типу сповіщень, які створює програма. Додаток також може створювати канали сповіщень у відповідь на вибір, зроблений користувачами. Наприклад, розробник може налаштувати окремі канали сповіщень для кожної групи розмов, створеної користувачем у програмі для обміну повідомленнями. Також для кожного каналу розробник чи користувач може вказати ступінь важливості сповіщень. Таким чином всі сповіщення, розміщені в одному каналі сповіщень, мають однакову поведінку.

Android використовує важливість сповіщення, щоб визначити, наскільки сповіщення має привертати увагу користувача (візуально та звуком). Чим більша важливість сповіщення, тим більш помітним буде сповіщення.

На Android 8.0 і новіших версій важливість сповіщення визначається важливістю каналу, на якому було опубліковано сповіщення. Користувачі можуть змінити важливість каналу сповіщень у налаштуваннях системи. В Android 7.1 і нижче важливість кожного сповіщення визначається окремим полем – пріоритетом сповіщення. Визначені наступні рівні важливості:

- Терміновий: має звук і відображається як heads-up сповіщення.
- Високий: має звук.
- Середній: немає звуку.
- Низький: немає звуку і не відображається в рядку стану.

Усі сповіщення, незалежно від їхньої важливості, відображаються внеперервних елементах інтерфейсу системи, наприклад на панелі сповіщень, чи у вигляді значка панелі запуску.

Починаючи з Android 5.0, користувачі можуть увімкнути режим «Не турбувати», який вимкне звук та вібрацію для всіх сповіщень. Наш застосунок повинен правильно його обробляти. Якщо даний режим увімкнено, сповіщення, як і раніше, відображаються в системному інтерфейсі, якщо користувач не вимкне їх окремо. У режиму «Не турбувати» доступні три різні рівні:

- Повна тиша: блокує всі звуки та вібрацію, включно з будильниками, музикою, відео та іграми.
- Тільки будильники: блокує всі звуки та вібрацію, окрім будильників.
- Лише пріоритет: користувачі можуть налаштувати, які загальносистемні категорії можуть переривати їх (наприклад, лише будильники, нагадування, події, дзвінки чи повідомлення). Для повідомлень і дзвінків користувачі також можуть вибрати фільтрацію на основі того, хто є відправником або абонентом

На ОС Android 8.0 і новіших версій користувачі можуть додатково дозволити надсилання сповіщень для певних категорій програми (також відомих як канали), перевизначивши режим «Не турбувати» для кожного каналу. Наприклад, платіжний додаток може мати канали для сповіщень, пов’язаних із зняттям коштів та депозитами. У пріоритетному режимі користувач може дозволити сповіщення про зняття коштів, сповіщення про депозит або обидва. На пристроях з ОС Android 7.1 і нижче користувачі можуть дозволити надсилання сповіщень для певної програми, а не на основі каналу.

Починаючи з Android 8.1, програми не можуть видавати звук сповіщення частіше одного разу на секунду. Якщо ваш додаток публікує кілька сповіщень за одну секунду, усі вони відображаються, як очікувалося, але тільки перше сповіщення має звук. Однак Android також застосовує обмеження швидкості під час оновлення сповіщення. Якщо ви надто часто публікуєте оновлення для одного сповіщення (кілька за менш ніж одну секунду), система може проігнорувати деякі з цих оновлень.

3.2 Структура сповіщень в Notification API

Основним класом, який надає Android Notification API є сам клас Notification. На рис. 8 зображені основні поля цього класу. Окрім цього, присутні ще системне поле Notification id. Ідентифікаційний номер сповіщення необхідний щоб отримати посилання на нього поза додатком, якому належить відповідний канал сповіщень. Також нам знадобиться , щоб відслідковувати зміну стану сповіщень, Activity list (список доступних дій та параметри) і Notification priority, якщо ми збираємось імплементувати це в поведінку застосунка (Поле priority в нових версіях Android API належить каналу, якому належить сповіщення).

Цих полів достатньо щоб правильно відобразити сповіщення, та виконати будь-яку дію з ним, тобто це все що необхідна застосунку для обробки сповіщення.

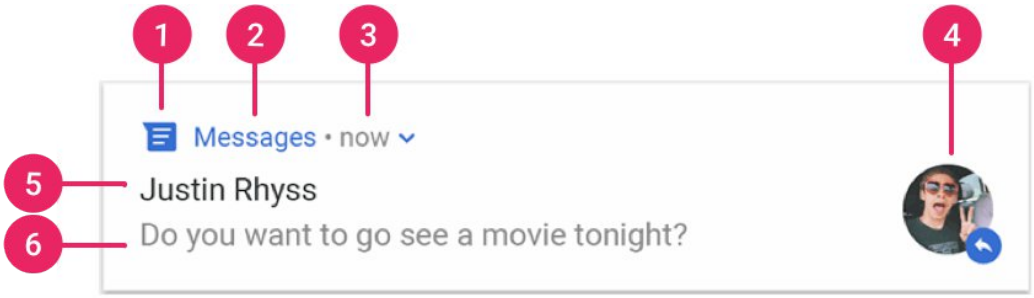

Рисунок 8 – Назви полів в структурі сповіщення: 1. Small icon, 2. App name, 3. Time stamp, 4. Large icon, 5. Title, 6. Text

Тепер, коли ми можемо обробляти сповіщення, нам необхідно знайти простий інтерфейс для доступу до сповіщень. Для цього ми скористаємось класом NotificationListenerService. Як видно з назви, цей клас реалізує службу Android. Вона, яка отримує виклик від системи, коли відбувається публікація, видалення, або змінюється рейтинг сповіщення. В ці моменти ця служба має можливість виконувати код, і має посилання на об’єкт класу Notification відповідного сповіщення.

3.3 Служби в ОС Android

Служба – це компонент програми, який може виконувати тривалі операції у фоновому режимі. Він не надає інтерфейсу користувача. Після запуску служба може продовжувати працювати деякий час, навіть якщо користувач переходить на іншу програму. Крім того, компонент може прив’язуватися до служби, щоб взаємодіяти з нею і навіть виконувати міжпроцесний зв’язок (IPC). Наприклад, служба може обробляти мережеві транзакції, відтворювати музику, виконувати ввод-вивод файлів або взаємодіяти з постачальником контенту, і все це у фоновому режимі. При цьому, служба працює в основному потоці процесу-власника; служба зазвичай не створює власний потік і не запускається в окремому процесі. Існує три типи служб:

Служба переднього плану виконує певну операцію, яка помітна для користувача. Наприклад, аудіододаток використовуватиме службу переднього плану для відтворення аудіо. Служби переднього плану повинні відображати сповіщення. Служби переднього плану продовжують працювати, навіть якщо користувач не взаємодіє з програмою.

Коли застосунок використовує службу переднього плану, він повинні відобразити сповіщення, щоб користувачі активно знали, що служба запущена. Це сповіщення не можна відхилити, доки службу не буде зупинено або вилучено з переднього плану.

Фонова служба виконує операцію, яку користувач безпосередньо не помічає. Наприклад, якщо програма використовує службу для ущільнення свого сховища, це зазвичай буде фоновою службою. Система накладає обмеження на запуск фонових служб, коли сама програма не знаходиться на передньому плані. У більшості ситуацій, наприклад, додатки не повинні отримувати доступ до інформації про місцезнаходження у фоновому режимі. Щоб виконувати роботу коли програма не знаходиться на передньому плані, слід використовувати функціонал WorkManager API.

Прив'язана служба виникає, коли компонент програми прив'язується до неї за допомогою виклику bindService(). Прив'язана служба пропонує інтерфейс клієнт-сервер, який дозволяє компонентам взаємодіяти зі службою, надсилати запити, отримувати результати і навіть робити це між процесами з міжпроцесним зв'язком (IPC). Прив'язана служба працює лише доти, доки до неї прив'язаний інший компонент програми. Декілька компонентів можуть прив'язуватися до служби одночасно, але коли всі вони від'єднуються, служба знищується.

Служба – це компонент, який може працювати у фоновому режимі, навіть коли користувач не взаємодіє з додатком. Згідно з цим описом, основний функціонал додатку слід реалізувати у вигляді служби переднього плану.

Документація зазначає, що якщо додаток використовує службу, вона все ще працює в основному потоці програми за замовчуванням, тому слід створити

новий потік у службі, якщо вона виконує інтенсивні або блокуючі операції. Оскільки більшість операцій з мережевими сокетами в мові Java є блокуючими, слід розглядати багатопоточну архітектуру додатку.

3.4 Життєвий цикл служби додатку

Створити власну службу можна шляхом розширення класу Service або використовувати один із існуючих підкласів. У власній реалізації необхідно перевизначити методи зворотного виклику, які обробляють ключові аспекти життєвого циклу служби та забезпечити механізм, який дозволяє компонентам прив’язуватися до служби, якщо це необхідно. Найважливіші методи зворотного виклику, за допомогою перевизначення яких реалізується функціонал служби:

onStartCommand(). Система викликає цей метод під час виконання startService(), коли інший компонент просить запустити службу. Коли цей метод виконується, служба запускається і може працювати у фоновому режимі необмежено. Реалізація несе відповідальність за зупинку служби, коли її робота буде завершена, викликавши stopSelf() або stopService().

onBind(). Система викликає цей метод під час виконання bindService(), коли інший компонент хоче зв’язатися зі службою. У реалізації цього методу створюється інтерфейс, який клієнти використовують для зв’язку зі службою, повертаючи IBinder. Цей метод завжди має бути реалізовано, однак, якщо є мета не дозволяти прив'язування компонентів до даної служби, метод повинен повернути null.

onCreate(). Система викликає цей метод для виконання одноразових процедур налаштування під час початкового створення служби (перед тим, як вона викликає onStartCommand() або onBind()). Якщо служба вже запущена, цей метод не викликається.

onDestroy(). Система викликає цей метод, коли служба більше не використовується і знищується. Метод відповідальний за вивільнення ресурсів,

що використовувалися службою. Це останній метод, який виконується в службі. Якщо компонент запускає службу, викликаючи startService() (що призводить до виклику onStartCommand()), служба продовжує працювати, доки не зупиниться за допомогою stopSelf() або інший компонент не зупинить її, викликавши stopService(). Якщо компонент викликає bindService() для створення служби, а onStartCommand() не викликається, служба працює лише доти, доки компонент прив'язаний до неї. Після того, як сервіс від'єднається від усіх своїх клієнтів, система знищить його.

Система Android зупиняє службу лише тоді, коли їй бракує пам'яті, і вона повинна відновити системні ресурси для діяльності, яка зосереджена на користувача. Якщо служба прив'язана до діяльності, яка зосереджена на користувачеві, ймовірність її знищення менша; якщо оголошується, що служба працює на передньому плані, вона рідко знищується в такий спосіб. Якщо служба запущена і працює довго, система з часом знижує свою позицію в списку фонових завдань, і служба стає приорітетною для знищення – якщо ваша служба запущена, ви повинні спроектувати її так, щоб коректно обробляти перезапуски з боку системи. Якщо система знищує вашу службу, вона перезапускає її, як тільки ресурси стають доступними, але це також залежить від значення, яке повертає onStartCommand().

3.5 Реалізація основного функціоналу як служби переднього плану

Необхідна нам служба NotificationListenerService є фоновою службою, або ж, якщо її використовувати разом з компонентом інтерфейсу, прив'язаною. Це означає, що вона не зможу протягом довгого часу підтримувати свою роботу. Тим більше, застосунок повинен в цей час комунікувати з клієнтом, приймати запити на з'єднання, тощо (див. Розділ 2). Тому нам слід створити власну службу переднього плану.

Служби переднього плану виконують операції, які помітні для користувача. Вони відображають сповіщення в рядку стану, щоб користувачі усвідомлювали, що програма виконує завдання на передньому плані та споживає системні ресурси. Сповіщення не може бути скасоване, якщо служба не буде зупинена або вилучена з переднього плану.

Ми вже розглянули створення та життєвий цикл служби, тож залишилось імплементувати поведінку, описану в розділі 2. Також слід зауважити, що починаючи з Android 12 є обмеження на запуск застосунками служб переднього плану, проте їх можна обійти, надавши йому необхідні дозволи.

Застосунок матиме тільки один Activity, це стан очікування застосунку. У ньому застосунок генерує випадкове число та очікує на вхідні підключення за допомогою двох фонових служб (2 канали стану очікування). Після під’єднання клієнта, застосунок запскає службу переднього плану, що виконує обмін даними з клієнтом, та забезпечує виконання нашого розширення NotificationListenerService.

4 ОПИС РОЗРОБКИ КЛІЄНТСЬКОГО WINDOWS ЗАСТОСУНКУ

4.1 Розробка мережевого застосунку використовуючи Qt Network

Модуль Qt Network надає класи, які дозволяють створювати клієнти та сервери TCP/IP. Він працює як з класами нижчого рівня, такими як QTcpSocket, QTcpServer і QUdpSocket, що представляють мережеві концепції низького рівня, так і класами високого рівня, такі як QNetworkRequest, QNetworkReply і QNetworkAccessManager для виконання мережевих операцій за допомогою загальних протоколів. Він також надає об’єктно орієнтоване API у вигляді класів QNetworkConfiguration, QNetworkConfigurationManager і QNetworkSession, які реалізують керування передачею даних в мережі.

Оскільки сервер не використовує Qt, нам доведеться використовувати низькорівневі класи, і вже розроблений в розділі 2 протокол передачі даних. В цьому розділі буде детально описано реалізацію роботи додатку в нормальному стані. Код реалізації можна знайти в додатку А.

Функціонал стану реалізовано як розширення класу QTcpServer (серверна імплементація використана тому, що з’єднання ініціює мобільний пристрій після успішної автентифікації) – NotificationWorker. Протокол SSL забезпечує достатній рівень надійності, що вимагається технічним завданням. Сервер працює в окремому потоці, оскільки використовує блокуючі операції.

В нормальному режимі сервер очікує на повідомлення від сервера, або користувача. Сервер може надіслати 3 види повідомлень:

- Отримане нове сповіщення
- Сповіщення скасовано
- Сповіщення оновлено

Для інтерпретації повідомлення, застосунок використовує парсер повідомлень, реалізований в класі ServerMessageParser. Після інтерпретації

повідомлення, NotificationWorker виділяє відповідний сигнал, щоб повідомити про це застосунок.

Для реагування на дії користувача використовуються відповідні слоти. Від користувача застосунок очікує 2 види дій:

- Скасування сповіщення
- Вибір однієї з дій сповіщення

Кожен з них буде оброблено та передано на сервер. Якщо виконання дії призведе до скасування сповіщення, сервер про це повідомить окремим повідомленням.

4.2 Створення графічного інтерфейсу додатку на мові QML

QML – це декларативна мова, яка дозволяє описувати інтерфейси користувача з точки зору їхніх візуальних компонентів і того, як вони взаємодіють і взаємодіють один з одним. Це добре читабельна мова, яка була розроблена для того, щоб компоненти можна було динамічно з’єднувати між собою, а також дозволяє легко повторно використовувати й налаштовувати компоненти в інтерфейсі користувача. Використовуючи модуль QtQuick, дизайнери та розробники можуть легко створювати плавні анімовані користувацькі інтерфейси в QML і мають можливість підключити ці інтерфейси користувача до будь-яких внутрішніх бібліотек C++ [12].

Модуль Qt Quick є стандартною бібліотекою для написання QML-додатків. Він надає всі основні типи, необхідні для створення інтерфейсів користувача з QML; забезпечує можливості, що включають типи для створення та анімації візуальних компонентів, отримання введених даних користувача, створення моделей і представлень даних, а також відкладеного створення екземплярів класів. Таким чином, QML це водночас специфікація інтерфейсу користувача та мова програмування. Це дозволяє як розробникам, так і дизайнерам створювати високопродуктивні, плавно анімовані та візуально привабливі програми. QML

пропонує добре читабельний, декларативний, схожий на JSON синтаксис з підтримкою імперативних виразів JavaScript у поєднанні з динамічними прив’язками властивостей [6].

QML було розроблено для легкого розширення за допомогою коду C++. Класи в модулі Qt QML дозволяють завантажувати QML-об'єкти та маніпулювати ними з C++, а характер інтеграції механізму QML із системою мета-об'єктів Qt дозволяє функціональність C++ викликати безпосередньо з QML. Це дозволяє розробляти гібридні програми, які реалізуються за допомогою поєднання коду QML, JavaScript і C++.

Інтеграція QML і C++ надає різноманітні можливості, включаючи можливість:

- Відокремити код інтерфейсу користувача від логічного коду програми, реалізувавши перший за допомогою QML і JavaScript в QML-документах, а другий – за допомогою C++.
- Використовувати та викликати деякі функції C++ з QML (наприклад, логіку програми можна реалізувати, використовуючи модель даних, реалізовану на C++, або викликати функції зі сторонньої бібліотеки на мові C++).
- Доступ до функцій Qt QML або Qt Quick C++ API (наприклад, для динамічного створення зображень за допомогою QQuickImageProvider).
- Реалізувати власні типи об’єктів QML із C++ для використання у власному конкретному застосунку.

Щоб дані або функціональні можливості C++ компоненту QML, вони повинні бути доступні з класу, похідного від QObject. Завдяки інтеграції механізму QML із системою мета-об’єктів бібліотеки QT, властивості, методи та сигнали будь-якого похідного від QObject класу доступні з QML. Після того, як такий клас забезпечує необхідну функціональність, він може бути завантажений QML різними способами:

- Клас можна зареєструвати як екземплярний тип QML, щоб його можна було створювати та використовувати як будь-який звичайний тип об'єкта QML із коду QML.
- Клас можна зареєструвати як Singleton Type, щоб один екземпляр класу міг бути імпортований в код QML, що дозволить отримати доступ до властивостей, методів і сигналів екземпляра з QML.
- Екземпляр класу може бути вбудований в код QML як властивість контексту або об'єкт контексту, що дозволяє отримати доступ до

Крім того, окрім можливості доступу до функціональних можливостей C++ з QML, модуль Qt QML також надає способи робити зворотне та маніпулювати QML-об’єктами з коду C++.

Таким чином, QML можна легко розширити з C++ завдяки інтеграції механізму QML із системою мета-об’єктів Qt. Ця інтеграція дозволяє з QML отримати доступ до властивостей, методів і сигналів будь-якого похідного від QObject класу: властивості можна читати та змінювати, методи можна викликати з виразів JavaScript, а обробники сигналів автоматично створюються для сигналів, якщо необхідно. Крім того, значення перерахування похідного від QObject класу доступні з QML. Типи QML можна визначити в C++, а потім зареєструвати в системі типів QML. Це дозволяє створити екземпляр класу C++ як об’єкт в QML, що дозволить реалізувати власні типи об’єктів у C++ та інтегрувати їх у існуючий код QML. Клас C++ також може бути зареєстрований для інших цілей: наприклад, його можна зареєструвати як Singleton Type, щоб дозволити імпортувати один екземпляр класу за допомогою коду QML, або його можна зареєструвати, щоб увімкнути перерахувальні значення неекземпляру.

Оскільки основна частина нашого застосунку уже загорнута в клас, що розширює QObject, щоб реалізувати графічний інтерфейс, достатньо створити два QML компоненти – сповіщення та екран налаштувань. Реалізацію інтерфейсу наведено в додатку Б.

ВИСНОВКИ

Розроблено застосунок для синхронізації мобільного пристрою з персональним комп’ютером по локальній мережі. Проаналізовано найпоширеніші існуючі аналоги KDE Link та Windows Phone Link, визначено їх переваги та недоліки. Показано, що вони мають багато надлишкового функціоналу, який витрачає ресурси системи та навантажує інтерфейс користувача. В цей час, найбільш корисною було визначено можливість відображення інформації з мобільного пристрою безпосередньо на робочому столі персонального комп’ютера. На основі цього, основними функціями розроблюваного застосунку стали синхронізація сповіщень та повідомлень.

При виконанні поставленого завдання, під час розробки додатку для персональних комп’ютерів використовувався кросплатформенний фреймворк Qt для мови програмування C++ з відповідним пакетом інструментів, а у додатку під мобільну платформу як інструмент застосовано інтегроване середовище розробки Android Studio та обрано мову програмування Java.

Розроблений застосунок відповідає поставленим технічним вимогам по продуктивності та безпеці, і є кращим за розглянуті аналоги по цим критеріям. Програма працює без використання сторонніх серверів та використовує протоколи шифрування SSL та цифрового підпису RSA. Висока продуктивність досягнута за рахунок зменшення надлишкового функціоналу, не потрібного для переважної більшості користувачів.

Застосунок пропонує трендовий підхід до підвищення продуктивності праці, та буде корисним в умовах дистанційної роботи та навчання.

У зв’язку з епідемією COVID-19 в багатьох галузях відбувається перехід на дистанційні форми роботи. У зв’язку з відсутністю широковживаного спеціалізованого програмного забезпечення для синхронізації робочого процесу, подібні застосунки – це один з небагатьох способів підвищення впорядкованості,

покращення організації роботи при одночасному зменшенні психологічно навантаження на працівників в умовах хаотичного надходження інформації.

Подальший розвиток застосунку може включати розширення функціоналу за рахунок реалізації фільтрації та сортування сповіщень, а також одночасної підтримки більш ніж двох пристроїв, хоча останнє вимагатиме переробки нинішньої моделі мережевої взаємодії.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. KDE UserBase.Tutorials. [Електронний ресурс]: KDEConnect – Режим доступу до ресурсу: https://userbase.kde.org/KDEConnect#Ring_My_Phone
2. Okemwa K. How to use the new Phone Link app in Windows 10 or Windows 11. *OnMSFT.com* [Електронний ресурс]. 2022. – Режим доступу до ресурсу: https://www.onmsft.com/how-to/how-to-use-the-new-phone-link-app
3. Lazar G. Mastering Qt 5: Create stunning cross-platform applications using C++ with Qt Widgets and QML with Qt Quick. Second Edition / G. Lazar, R. Penea. Birmingham-Mumbai: Packt Publising, 2018. – 534 p.
4. Bilka G., Laanstra J. App Spotlight: Phone Link. *WinUI Community Call* [Електронний ресурс]. 2022. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=bNHGU6xmUzE
5. Shapel M. Qt Framework and QML. Sam-solutions.com [Електронний ресурс] – Режим доступу до ресурсу: https://www.sam-solutions.com/blog/qt-framework/
6. Qt quick introduction guide [Електронний ресурс] – Режим доступу до ресурсу: https://doc.qt.io/qt-6/qtquick-index.html
7. Knoll L. Qt 6.3 released [Електронний ресурс]: Qt Quick compilers. – Режим доступу до ресурсу: https://www.qt.io/blog/qt-6.3-released
8. Android Developers. Guides [Електронний ресурс] : Notifications Overview. – Режим доступу до ресурсу: https://developer.android.com/guide/topics/ui/notifiers/notifications
9. Developer Guides. Create a notification [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/training/notify-user/build-notification
10. NotificationListenerService [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/reference/android/service/notification/NotificationListenerService#onNotificationPosted(android.service.notification.StatusBarNotification,%20android.service.notification.NotificationListenerService.RankingMap)
11. QML Applications [Електронний ресурс] – Режим доступу до ресурсу: https://doc.qt.io/qt-5/qmlapplications.html
12. Benaffane Y. Why we should use Qt framework and QML. *Medium.com* [Електронний ресурс]. 2019. – Режим доступу до ресурсу: https://medium.com/@yacine.benaffane/why-we-should-use-the-qt-framework-and-the-qml-f01c4edb13cd

ДОДАТОК А

Реалізація механізму створення робочого каналу клієнтом

notificationserver.h:

```
#include <QTcpServer>
#include <QSslSocket>
#include <QFile>

class NotificationServer : QTcpServer
{
public:
   NotificationServer(QObject* parent =0);

   void incomingConnection(int handle);
};
```

notificationserver.cpp:

```
#include "notificationserver.h"
#include <iostream>
#include <QByteArray>
#include <QSslCertificate>
#include <QSslKey>

using namespace std;

NotificationServer::NotificationServer(QObject* parent) : QTcpServer(parent){}
```

```
void NotificationServer::incomingConnection(int socketDescriptor)
{
    auto socket = new QSslSocket;
    connect(socket, SIGNAL(encrypted()),
            this, SLOT(startRead()));
    listen(QHostAddress::Any, 8889);
    if (socket->setSocketDescriptor(socketDescriptor))
    {
        QByteArray key;
        QByteArray cert;

        QFile file_key("/path_to_key/rsakey");

        if(file_key.open(QIODevice::ReadOnly))
        {
            key = file_key.readAll();
            file_key.close();
        }
        else
        {
            qDebug() << file_key.errorString();
        }

        QFile file_cert("/path_to_certificate/mycert.pem");
        if(file_cert.open(QIODevice::ReadOnly))
        {
            cert = file_cert.readAll();
            file_cert.close();
        }
```

```
        else
        {
            qDebug() << file_cert.errorString();
        }

        QSslKey ssl_key(key, QSsl::TlsV1_2);
        QSslCertificate ssl_cert(cert);

        socket->setPrivateKey(ssl_key);
        socket->setLocalCertificate(ssl_cert);

        QSslConfiguration cfg = socket->sslConfiguration();
        cfg.caCertificates();

        socket->startServerEncryption();

        new NotificationWorker(this, socket);
    }
}
```

ДОДАТОК Б

Реалізація компоненти інтерфейсу для відображення сповіщень

```
import QtQuick 2.3
import QtQuick.Controls.Universal 2.3

Window {
    width: 400; height: 170
    x: Screen.width + 2; y: Screen.height / 12
    color: "transparent"

    flags: Qt.Popup | Qt.NoDropShadowWindowHint | Qt.WindowStaysOnTopHint

    visible: x >= Screen.width ? false : true

    Image {
        id: smallIcon
        height: 16; width: 16
        anchors.left: parent.left
        anchors.top: patent.top
    }

    Text {
        id: header
        font: headerFont
        height: 16
        anchors.left: smallIcon.right
        anchors.top: patent.top
        anchors.right: parent.right
```

```
    verticalAlignment: Text.AlignVCenter
}

Text {
    id: text
    anchors.left: parent.left
    anchors.right: parent.right
    anchors.top: smallIcon.bottom
    anchors.bottom: actions.top
    font: textFont
}

ListView {
    id: actions
    anchors.left: parent.left
    anchors.right: parent.right
    height: 14;
    orientation: ListView.Horizontal
    interactive: false
    contentX: listController.contentX
}

Behavior on x {
    SpringAnimation { spring: 8; damping: 10 }
}

function hideWindow() {
    x = Screen.width + 2
}
```

```
    function showWindow() {
        peerList.clear()
        x = Screen.width - width
        peerList.refresh()
    }
}
```