Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота
за освітнім ступенем “бакалавр”
на тему:

ANDROID-ЗАСТОСУНОК ДЛЯ ЗАХОПЛЕННЯ ТА ЦИФРОВОЇ ОБРОБКИ ЗОБРАЖЕНЬ

Виконала:
здобувач IV курсу
групи КН-41
спеціальності 122 “Комп'ютерні науки”
Ігор Миколайович Приходько

Науковий керівник:
кандидат фізико-математичних наук,
ст. викладач Тарас Григорович Ляшук

Рівне – 2026

ЗМІСТ

СПИСОК ТЕРМІНІВ ТА УМОВНИХ ПОЗНАЧЕНЬ

Android Jetpack	Набір сучасних інструментів та архітектурних бібліотек Android
Camera API	Застаріле перше покоління програмного інтерфейсу камери
Camera2 API	Низькорівневий інтерфейс камери з глибоким ручним керуванням
CameraX	Високорівнева Jetpack-бібліотека для спрощеної роботи з камерою
HAL	Апаратний шар абстракції камери в ОС Android
HIDL	Архітектурна мова опису інтерфейсів взаємодії з HAL
ImageAnalysis	Модуль потокового аналізу та обробки кадрів у реальному часі
ImageCapture	Модуль фіксації, постобробки та асинхронного збереження фото
ISP	Спеціалізований апаратний мікропроцесор обробки сигналів зображення
Lifecycle-aware	Архітектурний підхід із прив'язкою до життєвого циклу ОС
LifecycleOwner	Системний інтерфейс контролю станів компонентів Android
MediaStore API	Програмний інтерфейс безпечного доступу до загальної галереї
Pipeline	Послідовний апаратний конвеєр обробки відеопотоку камери
Preview	Модуль попереднього перегляду та рендерингу живого відео
Runtime Permissions	Динамічні дозволи користувача під час виконання

	програми
Scoped Storage	Концепція ізольованого безпечного сховища в Android
Shared Storage	Публічна область спільного медіасховища пристрою
Surface	Графічна поверхня та буфер для виведення зображення
Use Cases	Архітектурні сценарії та логіка використання камери
UI	Інтенрфейс користувача
UX	Загальний користувацький досвід та зручність інтерфейсу

ВСТУП

Сучасний етап розвитку мобільних технологій характеризується стрімким зростанням вимог до якості та швидкості цифрової обробки медіаконтенту безпосередньо на портативних пристроях. Особливу актуальність це питання набуває в контексті розробки програмного забезпечення для роботи з вхідними відеопотоками в режимі реального часу, де ефективне керування апаратними ресурсами та оптимізація растрових даних стають ключовими чинниками стабільності інтерфейсу. Традиційні низькорівневі інструменти взаємодії з медіасенсорами часто виявляються надто складними у супроводі та чутливими до високої фрагментації ринку Android-пристроїв.

Актуальність роботи зумовлена необхідністю створення гнучкого, високопродуктивного мобільного застосунку для фотозйомки, який мінімізує навантаження на обчислювальні модулі пристрою та гарантує кросплатформену стійкість конвеєра обробки даних. Існуючі базові рішення часто мають недостатній рівень оптимізації оперативної пам'яті під час рендерингу важких графічних компонентів або не враховують сучасні безпекові вимоги ОС Android щодо ізоляції файлових систем.

Мета дослідження полягає в розробці та впровадженні мобільного застосунку для фотозйомки та цифрової обробки медіаданих, який забезпечує стабільну плавність інтерфейсу, асинхронну роботу з ресурсами камери та відмовостійкість сесії користувача на пристроях із різними апаратними конфігураціями.

Основні **завдання дослідження**:

- проаналізувати існуючі технологічні рішення для роботи з медіапотоками в Android та обґрунтувати доцільність розроблення системи;
- сформувати архітектурні та функціональні вимоги до мобільного застосунку;
- спроєктувати модель прецедентів, структуру користувацького інтерфейсу та конвеєр обробки зображень;

- реалізувати програмну логіку фотозйомки, динамічного перемикання сенсорів та адаптивну обробку системних відступів;
- інтегрувати механізми асинхронного даунсемплінгу, растрової трансформації та оптимізувати систему runtime-дозволів;
- виконати комплексне тестування життєвого циклу й інструментальне профілювання параметрів продуктивності застосунку.

Об'єкт дослідження – процес фіксації та цифрової обробки статичних і динамічних медіаданих на базі Android.

Предмет дослідження – методи, архітектурні компоненти та програмні засоби розробки відмовоїстійких мобільних застосунків для захоплення й оптимізації зображень з камери в реальному часі.

У кваліфікаційній роботі використано комплекс загальнонаукових та спеціальних **методів дослідження**, зокрема: аналіз і узагальнення науково-технічних джерел та існуючих API взаємодії з апаратними модулями; методи системного аналізу та об'єктно-орієнтованого проєктування; методи моделювання конвеєрів обробки даних; методи програмної інженерії (асинхронне та подійно-орієнтоване програмування); методи інструментального профілювання та тестування програмного забезпечення (кросплатформене, функціональне, тестування життєвого циклу).

Практичне значення дослідження. Розроблений мобільний застосунок та спроєктований конвеєр цифрової обробки зображень на базі Jetpack CameraX можуть бути використані як високоефективний програмний каркас для створення систем комп'ютерного зору, утиліт мобільної фотофіксації або інтегровані в освітній процес для вивчення технологій розробки сучасного кросплатформеного ПЗ з високим рівнем сумісності апаратного забезпечення.

Апробація і впровадження результатів дослідження. Основні теоретичні та практичні результати дослідження доповідалися та обговорювалися на: звітній науковій конференції викладачів, співробітників і здобувачів вищої освіти Рівненського державного гуманітарного університету за 2025 рік (м. Рівне, 14 травня 2026 р.); І Міжнародній науково-практичній

конференції «Сучасні інформаційні технології: від теорії до практики» (СІНТТЕХ 2026) (м. Луцьк, 22-23 травня 2026 р.).

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів (в яких містяться 17 рисунків та 2 таблиці), висновків, списку використаних джерел (13 найменувань) та чотирьох додатків. Загальний обсяг роботи, разом з додатками, становить 64 сторінки.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Архітектура та механізми керування камерою

Архітектурна побудова підсистеми камери в операційній системі (ОС) Android [1, 2] є однією з найскладніших частин графічного стека, оскільки вона повинна забезпечувати високу продуктивність при обробці великих масивів візуальних даних у реальному часі [3]. Сучасна реалізація базується на ієрархічній структурі, де кожен рівень виконує специфічні функції ізоляції апаратного забезпечення від прикладного коду.

Основним концептуальним елементом такої архітектури є Camera Hardware Abstraction Layer (HAL). У версіях Android 9 і вище використовується стандарт HIDL (HAL Interface Definition Language), який дозволяє оновлювати системні компоненти незалежно від драйверів виробника. HAL визначає набір інтерфейсів, які виробники модулів камери (напр., Sony, Samsung, OmniVision, тощо) повинні реалізувати для коректної роботи з ОС. Це гарантує, що незалежно від фізичної конструкції лінз чи типу матриці, програмний каркас Android отримує стандартизований вихідний потік даних. Таким чином, досягається апаратна кросплатформність, оскільки розробнику не потрібно впроваджувати окремі алгоритми для кожного конкретного модуля камери; натомість використовується єдиний високорівневий інтерфейс взаємодії, що нівелює проблему фрагментації апаратного забезпечення та дозволяє реалізувати єдиний протокол взаємодії з мультимедійною підсистемою пристрою.

Взаємодія компонентів системи відбувається через наступні ключові рівні:

- Application Framework – верхній рівень, з яким взаємодіє розробник. Він включає високорівневі класи (CameraManager та CameraDevice). На цьому рівні відбувається реєстрація слухачів подій та ініціалізація запитів на захоплення. Саме тут реалізується логіка вибору конкретного об'єктива

(напр., ширококутного, телеоб'єктива, тощо) на пристроях з декількома камерами;

- Camera Service – забезпечує централізоване керування та диспетчеризацію запитів. Він керує правами доступу, запобігаючи одночасному використанню апаратного модуля декількома процесами, що могло б призвести до помилок пам'яті. Служба камери передає конфігураційні параметри до HAL та отримує від нього метадані про стан зйомки;
- Camera HAL – низькорівневий шар, в якому відбувається безпосередня трансляція команд в інструкції для драйверів. HAL відповідає за налаштування параметрів 3A та розподіл буферів пам'яті для вихідних кадрів.

Логічним продовженням роботи цих рівнів є функціонування конвеєра обробки (pipeline). На відміну від застарілих систем, де камера розглядалася як пристрій зі станами, сучасна архітектура працює за моделлю потокових запитів. Кожен запит на отримання кадру проходить через спеціалізований мікропроцесор Image Signal Processor (ISP), який виконує операції дебайєризації, шумозаглушення, корекції геометричних спотворень лінзи та налаштування різкості.

Результатом роботи конвеєра є формування кількох незалежних потоків виводу. Система дозволяє одночасно направляти дані на різні поверхні (Surfaces): один потік низької роздільної здатності для попереднього перегляду (Preview) на екрані смартфона; інший – з максимальною якістю для кодування у файл JPEG; третій – для аналітичних алгоритмів або нейромереж. Такий підхід забезпечує багатозадачність обробки зображень без втрати кадрової частоти.

При цьому, важливою особливістю системних механізмів Android є керування життєвим циклом ресурсів [4, 5]. Оскільки модуль камери є одним із найбільш енергоємних компонентів пристрою, система суворо регламентує стани підключення та живлення апаратних складових (напр., приводів автофокусу та оптичної стабілізації). Управління сесіями через CameraCaptureSession дозволяє динамічно змінювати параметри сенсора, проте

ОС вимагає обов’язкового звільнення апаратних ресурсів при переході застосунку у фоновий режим. Це гарантує стабільність системи, запобігає перегріву та забезпечує передбачуваний користувацький досвід (UX) у мобільних застосунках.

Така структурована організація підсистеми дозволяє реалізувати складні алгоритми цифрової обробки зображень на базі стандартних механізмів ОС, мінімізуючи при цьому ймовірність виникнення конфліктів при доступі до апаратного забезпечення.

1.2. Системні стандарти Camera API та інструменти Jetpack

Розвиток програмних засобів керування камерою в ОС Android пройшов шлях від базових інтерфейсів до складних низькорівневих систем, що було зумовлено стрімким прогресом апаратного забезпечення мобільних пристроїв. Перша версія програмного інтерфейсу, Camera API, базувалася на спрощеній моделі управління, де програміст мав доступ лише до обмеженого набору параметрів. Цей стандарт не дозволяв реалізувати складні сценарії обробки зображень (напр., багатокадрову експозицію для HDR-зйомки, ручне фокусування на макрооб'єктах, серійну зйомку у форматі RAW, складні алгоритми сегментації об'єктів у реальному часі, тощо), оскільки він приховував від розробника внутрішні процеси конвеєра ISP.

Із виходом Android 5.0 було впроваджено Camera2 API [6], що докорінно змінило підхід до розробки. Новий стандарт надав можливість попіксельного доступу до даних та ручного керування кожним етапом захоплення: від швидкості затвора та значень ISO до кривих тональної компресії. Однак за потужність довелося заплатити надзвичайною складністю коду [7]. Наприклад, для ініціалізації простого вікна попереднього перегляду розробник повинен власноруч керувати об’єктами CameraCaptureSession, CaptureRequest та складними станами поверхонь виводу Surface. Будь-яка помилка в управлінні

потоками або станами життєвого циклу призводила до критичного завершення роботи застосунку або блокування апаратного модуля на рівні всієї ОС.

Головним викликом при використанні Camera2 API стала апаратна фрагментація. Виробники пристроїв впроваджують різні ступені відповідності цьому стандарту в межах рівня апаратної абстракції HAL. Для класифікації можливостей конкретного фізичного модуля використовується показник Hardware Support Levels [8], який визначає глибину контролю над процесом зйомки та доступність інноваційних функцій конвеєра:

- LEGACY – рівень, що охоплює пристрої, які функціонують виключно у режимі зворотної сумісності з Camera API. За такої архітектури система виступає посередником, що транслює сучасні запити у формат, зрозумілий застарілим драйверам. Це зумовлює відсутність доступу до більшості інноваційних можливостей сучасного конвеєра: зокрема, не підтримується вивід сирих даних матриці (RAW), не реалізовано динамічне пооб’єктне керування параметрами експозиції для кожного окремого кадру, а швидкість роботи системи фокусування суттєво знижується через відсутність прямого доступу до апаратних прискорювачів обробки зображень;
- LIMITED – категорія пристроїв представляє проміжний етап, де апаратне забезпечення підтримує лише частину специфікацій Camera2. На відміну від режиму сумісності з Camera API, тут розробнику доступні базові методи ручного керування, проте існують жорсткі ліміти на швидкість серійної зйомки та кількість одночасних потоків виводу. Основне обмеження полягає у відсутності гарантованої підтримки складних режимів обробки та неможливості використання деяких апаратних функцій оптичної стабілізації через програмні бар'єри на рівні драйвера виробника;
- FULL – повна підтримка апаратним забезпеченням стандарту Camera2. Пристрої даної категорії дозволяють керувати кожним етапом захоплення зображення, включаючи ручне налаштування сенсора, контроль

параметрів спалаху та стабілізації для кожного запиту окремо. Важливою перевагою є можливість паралельної генерації декількох потоків даних з високою роздільною здатністю без втрати кадрової частоти, що є критичним для застосунків із високими вимогами до продуктивності;

- LEVEL_3 – найвищий рівень апаратної інтеграції, що зазвичай притаманний передовим моделям. Окрім можливостей рівня FULL, він додає підтримку специфічних режимів, таких як перехоплення потоку YUV_420_888 з нульовою затримкою (Zero Shutter Lag) та можливість виводу RAW-даних паралельно з іншими потоками. Це дозволяє реалізувати в застосунку складні обчислювальні алгоритми фотографії на рівні системних камер виробника.

Таблиця 1.1

Порівняння рівнів апаратної підтримки Camera2 API

Можливість	LEGACY	LIMITED	FULL	LEVEL_3
Режим зворотної сумісності	Так	Ні	Ні	Ні
Ручне керування (ISO, витримка)	Обмежено	Базово	Повно	Повно
Паралельні потоки (HD/4K)	Ні	Обмежено	Так	Так
Підтримка RAW	Ні	Ні	Опційно	Так
Zero Shutter Lag (ZSL)	Ні	Ні	Ні	Так

Розробнику доводилося писати окремі гілки коду для кожного рівня підтримки, що робило підтримку застосунку надзвичайно ресурсомісткою. Вирішенням таких суперечностей став вихід бібліотеки CameraX, яка входить до складу інструментального стека Android Jetpack [9]. Вона побудована як інтелектуальна надбудова над Camera2, що бере на себе автоматичне вирішення проблем сумісності. Використання CameraX звільняє розробника від необхідності ручної перевірки специфікацій апаратної підтримки для кожного

пристрою. Бібліотека використовує механізм Camera2 Interop, який автоматично адаптує функціонал застосунку під можливості пристрою. Якщо апаратний модуль підтримує лише рівень LIMITED, CameraX самостійно оптимізує конвеєр, замінюючи відсутні апаратні функції програмними аналогами або коректно обмежуючи доступні режими зйомки, щоб запобігти критичним помилкам.

Детальна ієрархія взаємодії програмного забезпечення з апаратними ресурсами пристрою через бібліотеку Jetpack представлена на рис. 1.1. Тут Application Layer виступає верхнім рівнем, що формує запити до високорівневих компонентів бібліотеки, які згодом транслюються у системні команди для Camera2 API та драйверів HAL.

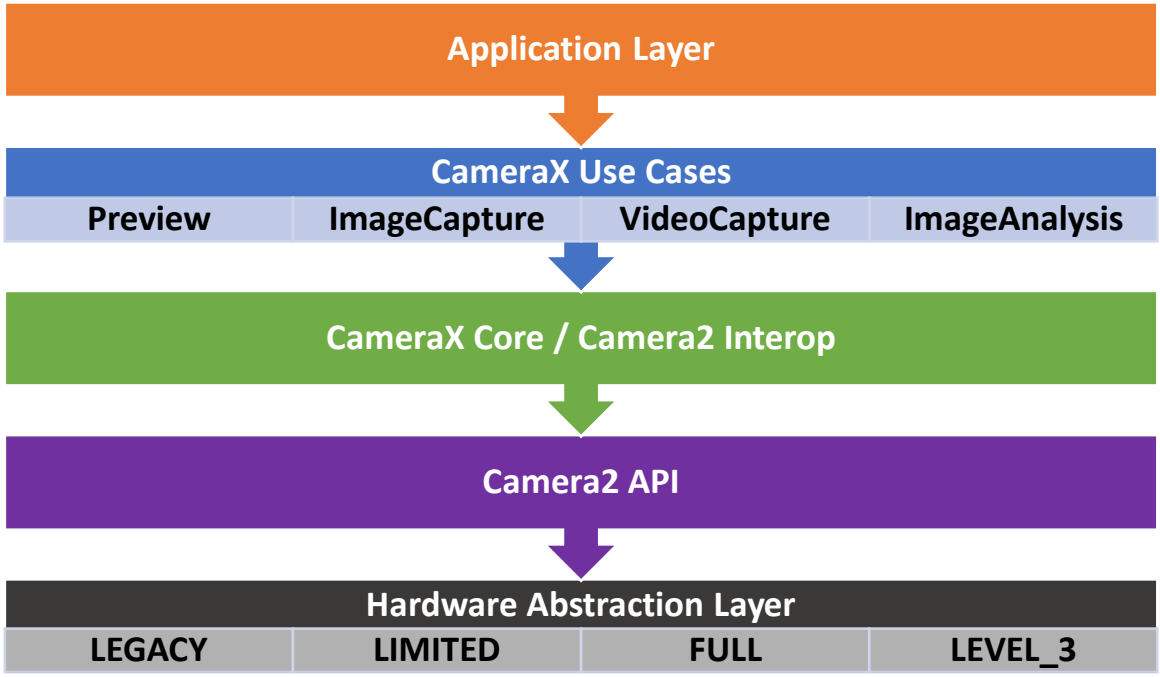


Рисунок 1.1. Архітектурна ієрархія CameraX та її взаємодія з Camera2 API.

Як показано на рисунку, архітектура CameraX побудована на принципі відокремлення логіки використання (Use Cases) від низькорівневого керування пристроєм. Це дозволяє розробнику фокусуватися на високорівневих об’єктах, що відповідають конкретним бізнес-задачам, тоді як рівень CameraX Core / Camera2 Interop автоматично адаптує ці вимоги під можливості апаратного

модуля (від LEGACY до LEVEL_3). Основними варіантами використання, що реалізують цей підхід, є:

- Preview – використовується для відображення візуального потоку в реальному часі. CameraX самостійно визначає оптимальну роздільну здатність, враховуючи співвідношення сторін екрана та технічні ліміти апаратного модуля;
- ImageCapture – здійснює фотофіксацію. Компонент включає складну логіку керування затвором, налаштування спалаху та вибору формату стиснення, забезпечуючи максимальну якість вихідного файлу при мінімальних затримках;
- VideoCapture – забезпечує запис відео- та аудіопотоку. Модуль автоматизує процес кодування та збереження даних у медіа-контейнери (напр., MP4), дозволяючи розробнику керувати якістю запису через налаштування профілів, таких як HD, FHD або 4K;
- ImageAnalysis – надає доступ до нестиснених кадрів (зазвичай у форматі YUV_420_888) для проведення математичних розрахунків або обробки алгоритмами комп’ютерного зору. Важливо, що аналіз відбувається паралельно з відображенням прев'ю в окремому фоновому потоці, не навантажуючи основний потік застосунку.

Окрім вирішення базових проблем сумісності, бібліотека надає доступ до пропрієтарних алгоритмів через систему Vendor Extensions (рис. 1.2), яка дозволяє інтегрувати в додаток специфічні налаштування від виробників (напр., нічний режим «Night Sight» від Google, покращення портретів від Samsung) через уніфікований інтерфейс ExtensionsManager. Це нівелює потребу у написанні пропрієтарного коду для кожного бренду смартфонів, що раніше було головною перешкодою для створення кросплатформних Android-застосунків високої якості.

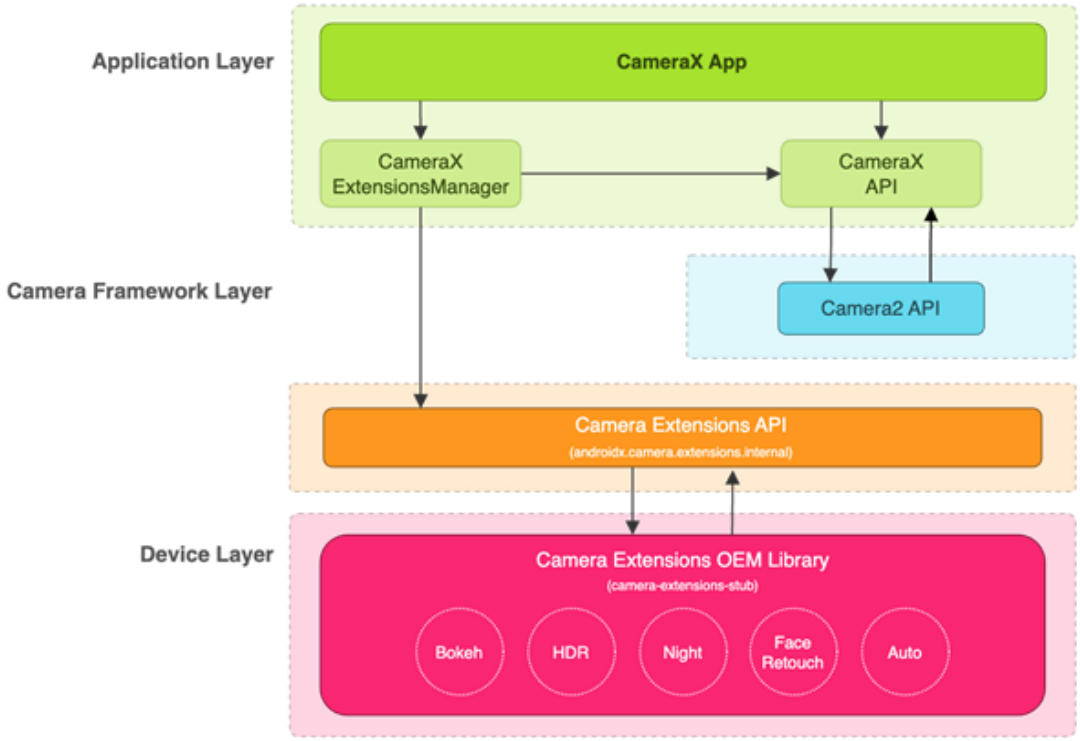


Рисунок 1.2. Архітектура Camera Extensions (ExtensionsManager).

Рисунок 1.3. Функції за замовчуванням, доступні в CameraX.

Варто зауважити, що CameraX інтегрується безпосередньо в життєвий цикл компонентів Android (LifecycleOwner). Це дозволяє автоматизувати процеси відкриття сесії при старті активності та, що більш важливо, миттєвого звільнення камери при переході застосунку в фоновий режим. Це усуває цілий клас помилок, пов’язаних із витоками пам’яті та некоректним станом обладнання, що робить програмне забезпечення значно стабільнішим.

Такий підхід до побудови програмного інтерфейсу перетворює розробку мультимедійних застосунків із процесу постійної боротьби з багами фрагментації на процес проектування логіки взаємодії користувача з камерою.

Оскільки архітектура CameraX побудована на базі Camera2 API, вона докорінно відрізняється від застарілого підходу Camera1. У той час як перша версія API вимагала від розробника ручного керування станами пристрою, CameraX автоматизує ці процеси, виступаючи високорівневим шаром абстракції. Для розуміння еволюції інструментарію та механізмів сумісності варто розглянути, як концепції попереднього покоління трансформуються у сучасну логіку Use Cases (рис. 1.4):

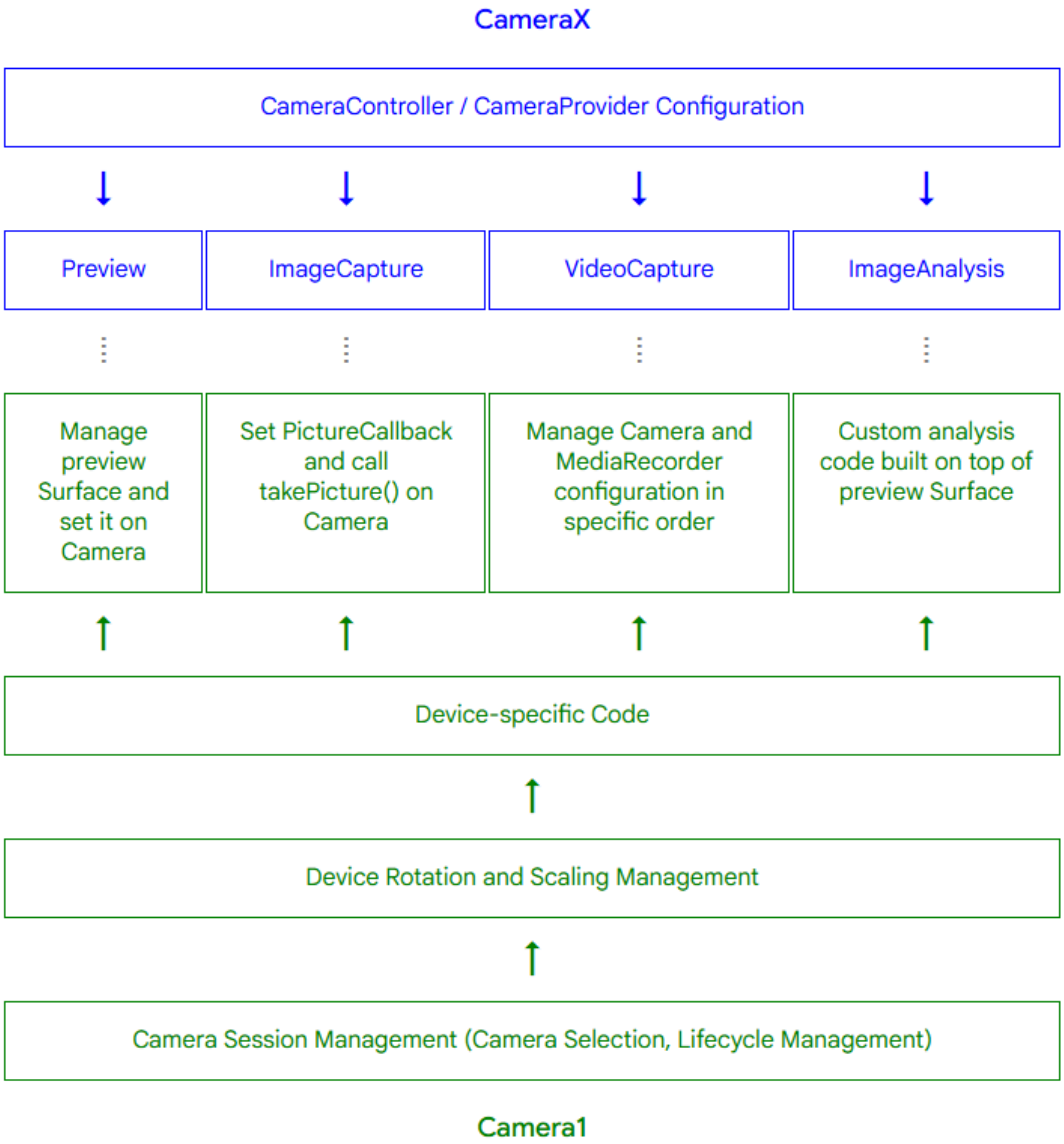


Рисунок 1.4. Порівняння концепцій Camera1 та CameraX.

Підсумовуючи аналіз засобів керування камерою, можна констатувати послідовну трансформацію підходів: від базового функціоналу Camera1 та надмірної низькорівневої складності Camera2 до високорівневої автоматизації в CameraX. Порівняльна характеристика цих інтерфейсів, наведена в табл. 1.2, демонструє переваги переходу до концепції варіантів використання, що дозволяє нівелювати ризики фрагментації апаратного забезпечення та спростити розробку складних мультимедійних систем.

Таблиця 1.2.

Порівняльна характеристика Camera1, Camera2 та CameraX.

Характеристика	Camera1 API	Camera2 API	CameraX Jetpack
Рівень абстракції	Високий (спрощений)	Дуже низький	Високий (на основі Use Cases)
Керування станом	Ручне (через об'єкт Camera)	Складне (скінченний автомат)	Автоматичне (Lifecycle-aware)
Сумісність (фрагментація)	Базова	Вимагає ручної перевірки HAL	Автоматична (через Interop)
Сценарії використання	Жорстко задані	Повністю кастомізовані	Модульні (Preview, Capture тощо)
Складність коду	Низька	Дуже висока	Мінімальна
Підтримка Vendor Extensions	Відсутня	Вимагає окремої імплементації	Уніфікована (ExtensionsManager)

1.3. Кодування та контейнеризація цифрових зображень

Ефективність збереження візуальної інформації в сучасних Android-пристроях базується на використанні складних алгоритмів стиснення,

реалізованих безпосередньо на рівні апаратного забезпечення (SoC). Вибір конкретного методу кодування визначає компроміс між підсумковою якістю контенту та обсягом займаного простору, що дозволяє мінімізувати навантаження на систему без суттєвої втрати візуальної чіткості:

- дискретне косинусне перетворення (DCT) та стандарт JPEG – основа стиснення нерухомих зображень. Алгоритм видаляє високі частоти, до яких людське око має низьку чутливість. При використанні ImageCapture у CameraX система автоматично підбирає коефіцієнти квантування на апаратному рівні, що дозволяє досягти оптимального балансу між розміром файлу та відсутністю видимих артефактів;
- субдискретизація колірності (Chroma Subsampling) – технологія селективного зниження роздільної здатності колірної інформації при збереженні повної деталізації яскравості. Оскільки людський зір сприймає яскравість значно чіткіше за колір, більшість мобільних сенсорів передають дані у форматі YUV 4:2:0. У цьому форматі яскравісна складова (Luma) зберігається повністю, а колірна (Chroma) – у зменшеній роздільній здатності. Таке рішення, реалізоване на рівні ISP, дозволяє значно скоротити обсяг інформації та економити пропускну здатність шини пам'яті пристрою;
- прогнозування та деревоподібна структура блоків (HEVC/H.265) – метод високоефективного кодування відеопотоку, що базується на алгоритмах внутрішньокадрового та міжкадрового передбачення. При відеозаписі через модуль VideoCapture, кодек HEVC (H.265) забезпечує на 50 % ефективніше стиснення порівняно із застарілим стандартом H.264 (AVC). Це досягається завдяки використанню деревоподібної структури блоків кодування (CTU) змінного розміру (до 64x64 пікселів), що дозволяє системі детально опрацьовувати складні текстури та динамічні сцени при суттєво меншому бітрейті;
- RAW-дані та формат DNG – спосіб отримання та зберігання первинної інформації безпосередньо з фотоматриці у форматі Bayer pattern. На рівнях

апаратної підтримки FULL/LEVEL_3 розробник отримує доступ до «сирих» даних, які не пройшли етап обробки процесором зображення ISP. Пакування таких даних у формат DNG дозволяє зберегти 10- або 12-бітну глибину кольору без втрат, пов'язаних з алгоритмами стиснення, що надає максимальні можливості для професійної пост-обробки та корекції експозиції без появи артефактів.

Логічним завершенням процесу обробки є пакування сформованих медіапотоків у цілісну файлову структуру, що реалізується через механізми контейнеризації. Якщо кодування відповідає за стиснення сигналу, то вибір стандарту контейнера визначає рівень сумісності файлу з програмними плеєрами, швидкість доступу до даних та ефективність інтеграції супровідних метаданих:

- MPEG-4 (MP4) – універсальна структура на базі ієрархічної системи блоків. Завдяки механізму розміщення заголовка метаданих на початку файлу, стандарт забезпечує можливість миттєвого відтворення відео без очікування його повного завантаження;
- HEIF (High Efficiency Image File) – сучасний формат на базі контейнера BMFF, що використовує кодек HEVC для зберігання кадрів. Він дозволяє об'єднувати в одному файлі не лише статичне зображення, а й карти глибини, послідовності знімків та аудіосупровід, споживаючи вдвічі менше пам'яті порівняно з JPEG;
- Exif (Exchangeable Image File Format) – стандарт інтеграції метаданих, що забезпечує фіксацію технічних параметрів зйомки усередині медіафайлів. Це дозволяє зберігати інформацію про координати GPS, параметри апаратної стабілізації та фокусну відстань, що необхідно для подальшої каталогізації та цифрової обробки;
- DNG (Digital Negative) – відкрита архітектура на базі формату TIFF/EP, що слугує для уніфікації RAW-даних. Вона забезпечує зберігання розширеного динамічного діапазону та повної колірної гами сенсора без

викривлень, які виникають при використанні споживчих форматів стиснення.

1.4. Організація системного сховища та доступу до медіаданих

У сучасній архітектурі Android робота з файлами базується на принципах ізоляції даних та диференційованого доступу. Еволюція системного сховища призвела до впровадження концепції розділеного сховища (Scoped Storage), яка радикально змінює парадигму взаємодії застосунку з файловою системою. Головною метою цього підходу є захист приватності користувача через обмеження прямого доступу програм до довільних папок на накопичувачі.

Для забезпечення цілісності даних та підтримки високого рівня безпеки, ОС Android реалізує багаторівневу ієрархію пам'яті [10]. Кожен рівень має власні протоколи доступу та життєвий цикл файлів, що критично впливає на логіку роботи з медіаконтентом:

- внутрішнє сховище (App-specific storage) – ізольована область пам'яті, що включає внутрішні директорії у файловій системі /data/data/ та спеціальні папки на зовнішньому накопичувачі. Це закрите середовище призначене для зберігання конфіденційних даних застосунку, тимчасових логів або кешованих зображень. Доступ до цієї зони не потребує отримання дозволів від користувача (Runtime Permissions), проте встановлює жорстке обмеження: файли є невидимими для інших програм (напр., системної галереї) і автоматично видаляються разом із деінсталяцією застосунку;
- спільне сховище (Shared Storage) – сукупність публічних директорій, структурованих за типом контенту: Pictures/, DCIM/ (для фото) та Movies/ (для відеозаписів). Саме сюди спрямовуються фінальні результати роботи модулів ImageCapture та VideoCapture бібліотеки CameraX. Дані на цьому рівні мають постійний характер і залишаються в системі навіть після видалення відповідного застосунку. Організація цього простору через механізм Scoped Storage гарантує, що застосунок може вільно записувати

власні медіафайли, але потребує спеціального доступу для читання або модифікації файлів, створених іншими програмами.

Основним інструментом для реєстрації, каталогізації та пошуку медіафайлів у системі є MediaStore API. Це високорівнева реляційна база даних, яка автоматично індексує контент на накопичувачі, дозволяючи іншим програмам (напр., системній Галереї, хмарним сховищам) миттєво «бачити» новостворені файли без необхідності повного сканування пам'яті.

Функціонування цієї системи базується на трьох фундаментальних механізмах:

- контент-провайдери (Content Providers) – системна абстракція, що виступає посередником між файловою системою та застосунком. В сучасній моделі безпеки прямі шляхи до файлів вважаються небезпечними та нестабільними. Замість них розробник оперує посиланнями Content URI. Це дозволяє системі динамічно керувати правами доступу: застосунок отримує не «адресу на диску», а унікальний токен-ідентифікатор, який підтверджує право на маніпуляцію конкретним об'єктом медіатеки;
- ContentValues та декларація метаданих – спеціалізований об'єкт-контейнер, що використовується для опису характеристик файлу ще до його фактичної появи на носії. У ContentValues розробник фіксує критичні параметри: дисплейне ім'я, MIME-тип (напр., image/jpeg або video/mp4), часову мітку та відносний шлях у публічному сховищі. Такий підхід дозволяє системі заздалегідь зарезервувати місце в базі даних, валідувати формат та коректно класифікувати майбутній файл у відповідну категорію (Зображення, Відео чи Аудіо);
- статус відкладеного завершення (Pending Status) – інтелектуальний режим ізоляції файлу, що реалізується через прапорець IS_PENDING. Цей стан є критично важливим під час тривалих операцій запису, зокрема фіксація 4K-відеозапису через VideoCapture. Доки цей статус активний, медіафайл залишається «невидимим» для сторонніх процесів, а виключне право на доступ має лише застосунок-творець. Це запобігає системним конфліктам,

коли сторонні медіа-плеєри або сервіси синхронізації намагаються зчитати ще незавершений або пошкоджений потік даних.

Використання описаних механізмів MediaStore API у поєднанні з архітектурними можливостями CameraX дозволяє створити стійку систему обробки медіаданих. Такий підхід забезпечує не лише високу швидкість реєстрації файлів у системі, а й гарантує повну відповідність сучасним вимогам безпеки Android щодо захисту персональних даних користувача.

Отже, розглянуті теоретичні аспекти архітектури CameraX, методи кодування та принципи організації системного сховища формують необхідний фундамент для практичної реалізації мобільного застосунку. Детальний аналіз цих технологій дозволяє перейти до етапу проектування та програмної реалізації функціональних модулів системи, що буде розглянуто у наступному розділі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1. Формування функціональних вимог

Проектування будь-якої складної програмної системи, зокрема мобільного застосунку для роботи з медіаданими, починається з детального аналізу та формулювання функціональних вимог. Ці вимоги визначають поведінку системи та перелік сервісів, які вона повинна надавати кінцевому користувачеві для розв'язання поставлених задач.

Для забезпечення гнучкості розробки та можливості подальшої модифікації коду, вимоги до застосунку було структуровано за модульним принципом, розділивши їх на критично важливі (базові) та розширювані (додаткові).

До переліку базових функціональних вимог, що складають ядро системи, відносяться:

- управління життєвим циклом камери – система повинна коректно ініціалізувати апаратні ресурси пристрою, враховуючи стани Activity (створення, зупинка, відновлення), щоб уникнути витоків пам'яті та конфліктів доступу;
- візуалізація потоку даних (Preview) – забезпечення плавного відображення відеопотоку на екрані з частотою кадрів, достатньою для комфортного сприйняття користувачем, та автоматичною адаптацією під орієнтацію екрана;
- коректна фіксація кадрів – реалізація механізму захоплення статичних зображень із гнучким залученням стандартних алгоритмів автофокусу та налаштування експозиції, що надаються бібліотекою CameraX.

Архітектура застосунку також базується на принципах модульності, що передбачає створення уніфікованих точок розширення для впровадження методів цифрової обробки. Це забезпечує наступні можливості:

- потоковий аналіз (Image Analysis) – можливість перехоплення кадрів у форматі YUV_420_888 або RGBA для проведення обчислювальних операцій (напр., визначення середньої яскравості або розпізнавання графічних образів);
- програмна фільтрація – реалізація логіки накладання візуальних ефектів через систему фільтрів-аналізаторів, що дозволяють здійснювати маніпуляції як з піксельною матрицею (напр., корекція кольорів, трансформації), так і складніші трансформації;
- динамічне керування – наявність адаптивного інтерфейсу, що забезпечує взаємодію з модулями обробки та зміну їх параметрів у режимі реального часу.

Окрему групу складають вимоги до інтеграції з екосистемою Android та збереження результатів роботи:

- транзакційність запису – використання механізму IS_PENDING для гарантування того, що файл не буде відкритий іншими програмами до моменту завершення всіх маніпуляцій з його вмістом;
- системна реєстрація – автоматичне індексування кожного створеного об’єкта в базі даних MediaStore із заповненням коректних метаданих (напр., дата створення, ім’я файлу, MIME-тип тощо).

Для формалізації описаних вимог та візуалізації взаємодії користувача із системою використовується діаграма прецедентів Use Case, яка дозволяє наочно продемонструвати функціональні можливості проекту. Основними компонентами моделі є:

1. Користувач – ініціатор взаємодії, який керує процесом захоплення зображення, перемикає активні модулі камери та взаємодіє з отриманим результатом.
2. Застосунок:
    - базові прецеденти:
        - «Запустити камеру»: початкова точка входу в застосунок;
        - «Зробити знімок»: ключова функція фіксації контенту;

- «Змінити активний модуль»: вибір між фронтальною та основною камерами;

- розширюючі прецеденти (<<extend>>):
  - «Застосувати цифровий фільтр»: цей сценарій доповнює основний процес зйомки. Використання зв’язку extend у діаграмі підкреслює, що ця дія є опціональною та може бути модифікована або вимкнена без порушення цілісності головного процесу;
  - «Налаштувати параметри аналізу»: дозволяє користувачу впливати на алгоритми обробки в реальному часі;
  - «Переглянути мініатюру останнього фото»: опціональний сценарій зворотного зв'язку, що активується після успішного збереження знімка, що дозволяє користувачу здійснити миттєву візуальну верифікацію результату цифрової обробки та забезпечує швидкий доступ до останнього створеного медіафайлу.
- Залежні прецеденти (<<include>>):
  - «Зберегти в медіатеку»: обов'язкова системна дія, що ініціюється автоматично після успішного завершення циклу захоплення та обробки зображення для фіксації результату в пам'яті пристрою, включаючи автоматичне формування метаданих (орієнтація, дзеркальність) для забезпечення коректного відображення у системній галереї.

3. Системна галерея – медіатека для зберігання отриманих фотознімків.

Графічне представлення діаграми прецедентів наведено на рис. 2.1:

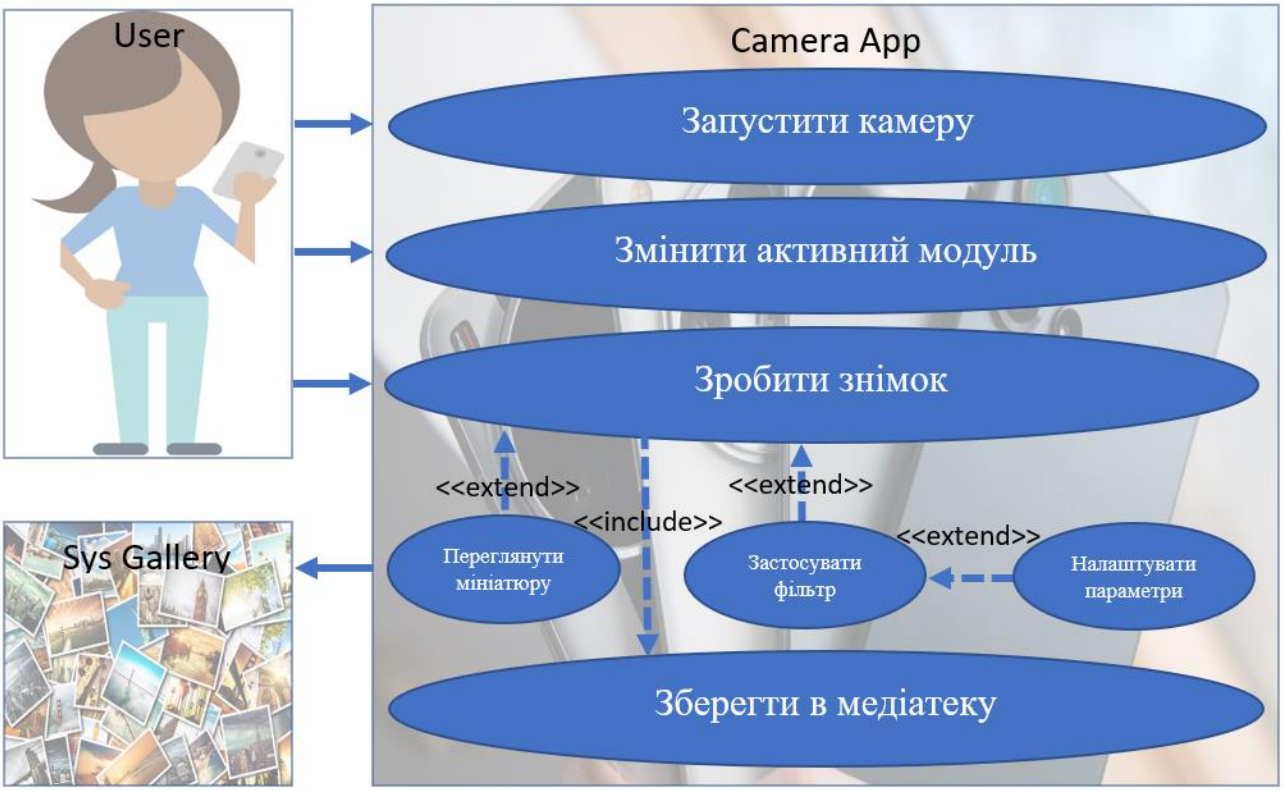


Рисунок 2.1. Діаграма прецедентів системи керування процесами зйомки та обробки медіаданих: прямокутник – межі системи; овал – прецедент (Use Case) (окрема функція або сценарій); суцільна лінія – зв'язок (взаємодія з функціями або зовнішніми сервісами); штрих-лінія – залежність між прецедентами (<<extend>> – доповнення функціоналу, <<include>> – обов’язкове включення).

Таким чином, чітка структуризація функціональних вимог дозволяє сформувати цілісне бачення системи, де базові можливості фіксації медіаданих органічно поєднуються з інструментами їхньої цифрової обробки та механізмами збереження стану інтерфейсу. Використання модульного підходу при визначенні вимог не лише спрощує процес проектування архітектури, але й закладає фундамент для створення інтероперабельної системи, здатної до коректної взаємодії з компонентами ОС Android та адаптації під специфічні інженерні задачі. Це гарантує, що розроблений програмний продукт володітиме високим рівнем відмовостійкості й масштабованості, забезпечуючи надійну

взаємодію між користувачем, програмними алгоритмами та апаратними ресурсами мобільної платформи.

2.2. Архітектура системи

Ефективність мобільного застосунку для обробки медіаданих безпосередньо залежить від обраної архітектурної моделі. Враховуючи специфіку ОС [11] Android та високу ресурсномісткість операцій з камерою, проектування системи було реалізовано на принципах життєво-циклової залежності (Lifecycle-aware architecture). Такий підхід базується на використанні сучасних декларативних методів, що дозволяють мінімізувати пряме втручання в низькорівневе керування апаратними ресурсами пристрою, забезпечуючи стабільність роботи при зміні станів застосунку.

Центральним елементом архітектури є об’єкт ProcessCameraProvider, який реалізує паттерн «Одинак» (Singleton) для гарантування єдиного та стабільного доступу до ресурсів камери в межах усього життєвого циклу застосунку. Даний об'єкт виступає ключовим вузлом ієрархії, який делегує доступ до сенсорів камери, суворо враховуючи поточний стан LifecycleOwner (Activity або Fragment). Це дозволяє вирішити ряд критичних інженерних завдань:

- автоматизація ініціалізації – архітектура гарантує, що камера активується та розпочинає стрімінг даних лише тоді, коли графічний інтерфейс стає видимим та активним для користувача. Це виключає ситуації, коли ресурси споживаються у фоновому режимі без реальної потреби;
- запобігання конфліктам – система автоматично відстежує життєвий цикл і вивільняє ресурси камери при згортанні застосунку або переході до іншої програми. Це критично важливо для ОС Android, де одночасний доступ декількох процесів до одного апаратного сенсора призводить до помилок типу «Camera Collision»;
- енергоефективність – завдяки автоматичній зупинці фонових обчислень та припиненню захоплення кадрів у стані onPause, застосунок суттєво

зменшує навантаження на центральний процесор та графічну підсистему, зберігаючи заряд акумулятора пристрою;

- адаптація до змін конфігурації – використання ProcessCameraProvider дозволяє системі безшовно перепідключати потоки даних при повороті екрана або зміні розміру вікна, автоматично перераховуючи параметри відображення без ризику зависання інтерфейсу.

Взаємодія між компонентами системи організована через конвеєрну структуру об’єктів UseCase, які працюють паралельно, використовуючи спільний ресурс камери через абстракцію CameraSelector. Кожен модуль виконує специфічне завдання обробки потоку даних:

- модуль попереднього перегляду (Preview) – реалізує відображення потокового відео на PreviewView у реальному часі. Використання SurfaceProvider дозволяє автоматично узгоджувати роздільну здатність камери з фізичними пікселями екрана, оптимізуючи навантаження на графічну підсистему за рахунок апаратного прискорення виводу;
- модуль фіксації зображень (ImageCapture) – призначений для отримання статичних кадрів високої роздільної здатності. Модуль підтримує програмну чергу запитів, що дозволяє виконувати захоплення даних із врахуванням метаданих (EXIF) та поточних параметрів експозиції, балансу білого та фокусування без переривання основного відеопотоку;
- модуль потокового аналізу (ImageAnalysis) – виступає інструментом для програмної маніпуляції медіаданими. Він забезпечує доступ до буфера кадрів у форматі YUV_420_888, де колірна інформація розділена на канали яскравості (Y) та хроматичності (UV). Це дозволяє застосовувати алгоритми цифрової обробки, такі як фільтрація або аналіз піксельної матриці, у виділеному фоновому потоці, не впливаючи на частоту кадрів модуля Preview.

Для підтримки високої продуктивності під час виконання обчислювальних операцій над зображеннями, архітектура передбачає чіткий розподіл потоків виконання (Threading strategy):

- головний потік (Main Thread) – обслуговує виключно рендеринг графічного інтерфейсу та обробку подій користувача, що гарантує відсутність затримок візуальної частини;
- потік-аналізатор (Analysis Executor) – спеціалізований фоновий потік, виділений для роботи модуля ImageAnalysis. В ньому можлива реалізація алгоритмів цифрової обробки кадрів, що дозволяє виконувати складні математичні трансформації піксельної матриці паралельно з виводом зображення на екран. Використання окремого виконавця повністю ізолює обчислювальні процеси від Main Thread, завдяки чому критичне навантаження на процесор при роботі з «важкими» фільтрами не впливає на плавність інтерфейсу та не спричиняє затримок у відгуку програми на дії користувача;
- потік дискових операцій (Disk IO Executor) – асинхронний потік введення-виведення, повністю ізольований від обчислювальних процесів та графічного інтерфейсу. Він призначений для транзакційного запису готових медіафайлів у постійну пам'ять пристрою та їхньої коректної реєстрації в системному сховищі MediaStore. Оскільки взаємодія з MediaStore передбачає роботу з базою даних контенту Android та може спричиняти значні затримки при індексації нових файлів, винесення цих операцій в окремий потік є критичним. Це гарантує, що тривалі процедури фіналізації запису та очікування відповіді від системного контент-провайдера не призводитимуть до блокування Main Thread, забезпечуючи безперервну плавність інтерфейсу навіть у моменти збереження об'ємних медіаданих. Паралельно з цим, даний потік обслуговує запити бібліотеки Glide для асинхронного декодування, ресайзингу та кешування мініатюр зображень. Це дозволяє здійснювати операції читання та візуалізації контенту безпосередньо з файлової системи в обхід головного потоку, що виключає появу мікро-зависань при оновленні елементів керування.

Важливою складовою архітектурної стійкості застосунку є механізм керування станом (State Persistence). Для забезпечення цілісності

користувацького досвіду при зміні конфігурації (поворот дисплея) або витісненні процесу системою, реалізовано серіалізацію параметрів через об’єкт Bundle. Це дозволяє динамічно відновлювати:

- активний модуль камери (lensFacing) – запобігає мимовільному перемиканню між фронтальним та основним сенсорами;
- контекстну візуалізацію – відновлення посилання на останній оброблений кадр (lastSavedUri) гарантує цілісність відображення даних в інтерфейсі та миттєвий доступ до результату останньої операції зйомки.

Запропонована багатокомпонентна модель забезпечує необхідний баланс між високою продуктивністю та стабільністю системи в умовах обмежених ресурсів мобільного пристрою. Завдяки чіткій ізоляції задач у межах паралельних потоків та використанню декларативних механізмів керування життєвим циклом, вдалося створити гнучкий програмний каркас, стійкий до критичних помилок доступу та апаратних перевантажень. Таке архітектурне рішення формує надійну базу для впровадження будь-яких методів аналізу та трансформації медіаданих, оскільки дозволяє масштабувати функціональні можливості застосунку без порушення цілісності та чуйності користувацького інтерфейсу.

2.3. Конвеєр обробки та аналізу зображень

Ключовою особливістю розробленої архітектури є можливість динамічного перехоплення та модифікації медіапотоку без втручання в роботу основного графічного конвеєра. Для забезпечення високого рівня масштабованості системи та дотримання принципів SOLID, проектування модулів обробки реалізовано через абстракцію ImageAnalysis.Analyzer. Такий підхід дозволяє повністю відокремити інженерну задачу стабільного отримання кадру від безпосередньої математичної реалізації алгоритмів його аналізу чи трансформації.

Фундаментальним аспектом побудови конвеєра обробки є концепція «програмного слота», що формує суворо ізольоване середовище для інтеграції та виконання обчислювальних алгоритмів будь-якого рівня складності. У межах цієї моделі використання інтерфейсу аналізатора розглядається як архітектурний контракт, що забезпечує стабільну передачу кожного кадру у формі стандартизованого об'єкта ImageProxy. Моделювання такої взаємодії передбачає врахування стратегій управління зворотним тиском, що є критичним для систем реального часу. Це реалізується через наступні інженерні рішення:

- уніфікація вхідних даних – незалежно від апаратної специфіки камери чи особливостей сенсора, кожен алгоритм взаємодіє з уніфікованою структурою, яка інкапсулює не лише масиви пікселів, а й повний набір метаданих, включаючи часові мітки та параметри формату;
- динамічне регулювання черги кадрів – модель дозволяє системі гнучко адаптуватися до поточних обчислювальних потужностей пристрою, обираючи між режимом неблокуючого аналізу (де нові кадри ігноруються до завершення поточної ітерації) та чергою з фіксованою затримкою, що дозволяє уникнути перевантаження процесора;
- ізоляція обчислювальних ресурсів – впровадження окремого виконавця (Executor) гарантує, що операції над матрицею зображення жодним чином не впливають на плавність графічного інтерфейсу, оскільки потік аналізу має власний стек пам'яті та пріоритет виконання;
- автоматизований менеджмент пам'яті – система суворо регламентує життєвий цикл кожного кадру; об'єкт ImageProxy автоматично сигналізує конвеєру про готовність прийняти наступні дані лише після виклику методу закриття буфера, що виключає ризики переповнення оперативної пам'яті;
- арбітраж доступу до буферів – система реалізує механізм безпечного читання даних, при якому алгоритм аналізу отримує копію або захищений потік до піксельної матриці, що дозволяє виконувати складні

трансформації без ризику пошкодження цілісності потокового відео, яке паралельно відображається у модулі попереднього перегляду.

Така глибока абстракція механізмів передачі даних дозволяє перетворити процес аналізу на незалежний програмний модуль, який можна масштабувати, замінювати або оптимізувати без втручання в основний код управління апаратними ресурсами пристрою.

В той же час, процес програмної маніпуляції медіаданими базується на низькорівневому доступі до планарної структури зображення. Модель обробки передбачає роботу з «сирими» байтовими масивами, що дозволяє реалізувати будь-яку логіку трансформації без надмірних витрат на конвертацію форматів:

- сегментація площин – на відміну від обробки готових файлів, аналізатор отримує прямий доступ до трьох площин даних. Це дозволяє, наприклад, ізолювати Y-канал (яскравість) для виконання операцій детекції об'єктів або зміни контрастності, не витрачаючи ресурси на обробку колірних каналів;
- представлення кадру в пам'яті – кожен кадр розглядається як набір байтових масивів з певним кроком, що дозволяє коректно зчитувати інформацію навіть при наявності технічних відступів у пам'яті, забезпечуючи універсальність алгоритмів для різних моделей пристроїв;
- керована трансформація вихідного потоку – конвеєр забезпечує динамічну модифікацію параметрів захоплення, зокрема реалізацію алгоритму горизонтальної інверсії для фронтальних модулів зйомки та автоматичну корекцію цільового кута повороту кадру. Це дозволяє привести растрову структуру зображення у відповідність до фізичної орієнтації пристрою ще на етапі формування масиву даних;
- інтеграція метаданих обробки – система дозволяє впроваджувати додаткові інструкції у структуру EXIF-заголовків безпосередньо під час фіналізації кадру. Це гарантує, що результати цифрової обробки будуть коректно інтерпретовані будь-яким зовнішнім програмним забезпеченням для перегляду медіафайлів;

- гнучка алгоритмізація процесів обробки – архітектурне рішення на основі інтерфейсів забезпечує високий ступінь адаптивності при впровадженні різних обчислювальних методів. Це дозволяє інтегрувати як базові лінійні трансформації (напр., інверсію кольорових каналів або порогову бінаризацію), так і складніші матричні перетворення, адаптовані під конкретні задачі. Система залишається відкритою для впровадження модулів комп'ютерного зору, де результати аналізу кожного кадру можуть бути використані для динамічного коригування параметрів зйомки або логіки роботи всього застосунку в реальному часі;
- асинхронний обмін даними з інтерфейсом – модель передбачає, що оброблені результати аналізу повертаються в основний потік програми через механізми зворотного виклику (callbacks). Це дозволяє оновлювати елементи користувацького інтерфейсу (напр., графічні маркери або текстові дані) паралельно з відеопотоком, не знижуючи загальну продуктивність системи та зберігаючи високу плавність відображення кадру.

Запропонована модель конвеєра забезпечує баланс між високою продуктивністю та гнучкістю архітектури. Прямий доступ до планарних даних у поєднанні з асинхронним обміном дозволяє мінімізувати апаратні витрати, створюючи надійну базу для аналізу зображень у реальному часі без втрати плавності інтерфейсу.

2.4. Збереження результатів у системному сховищі

Даний підрозділ логічно продовжує аналіз системного сховища, проведений у підрозділі 1.4, проте зміщує акцент із загальних принципів безпеки Android на безпосереднє інженерне впровадження цих механізмів. Якщо в теоретичній частині було розглянуто концепцію Scoped Storage та роль MediaStore API як зовнішніх регуляторів приватності, то на етапі проектування тут розробляється внутрішній алгоритм, який перетворює ці абстрактні

обмеження у чітко детерміновану послідовність операцій. Ключова відмінність полягає в тому, що замість огляду рівнів пам'яті, проектується механізм персистентного збереження, який інтегрує транзакції медіаконвеєра безпосередньо в життєвий цикл захоплення кадру.

Це досягається шляхом ініціалізації об’єктів OutputFileOptions, які ще до моменту спрацювання затвора визначають цільове призначення даних. Замість створення проміжних тимчасових файлів, система використовує об'єкт ContentValues для превентивної декларації параметрів майбутнього знімка. Такий підхід дозволяє реалізувати алгоритм генерації унікальних імен на основі системного часу та MIME-типу, що виключає конфлікти іменування та мінімізує операції копіювання між буферами пам'яті. Вся взаємодія будується на отриманні унікального Content URI, завдяки чому модуль камери залишається повністю незалежним від фізичної структури папок пристрою, оперуючи лише токеном-ідентифікатором від системи.

Під час безпосереднього запису, архітектура використовує низькорівневі оптимізації Android для автоматизації потокового виводу в медіатеку. Важливим інженерним рішенням тут є динамічне керування прапорцем IS_PENDING, гарантуючи, що до повного завершення запису, медіафайл залишається ізольованим, захищаючи цілісність даних від втручання сторонніх процесів. Оскільки параметри виводу (напр., орієнтація, дзеркалювання, метадані, тощо) фіксуються заздалегідь, фінальний файл не потребує додаткового пост-процесингу і коректно інтерпретується будь-яким стороннім додатком.

Завершальним етапом цієї моделі є архітектурна обробка результатів через інтерфейс зворотного зв'язку OnImageSavedCallback. Після верифікації транзакції, система автоматично скидає статус очікування, роблячи файл доступним для індексації сервісами MediaStore, і передає отриманий URI назад у конвеєр. Це ініціює ланцюжок вторинної обробки: асинхронне завантаження створеного медіафайлу засобами бібліотеки Glide для генерації прев’ю-мініатюри та оновлення стану Activity. Фіксація отриманого URI у структурі

SavedInstanceState завершує цикл збереження, гарантуючи доступність результату навіть після перестворення компонентів інтерфейсу.

Така спроектована модель робить процес збереження прозорим для користувача та стійким до оновлень безпеки, оскільки вона повністю базується на рекомендованих сервісних абстракціях ОС Android.

Підсумовуючи результати другого розділу, можна стверджувати, що розроблена архітектура формує масштабовану модель керування медіаданими, яка базується на чіткій формалізації вимог та модульному принципі побудови системи. Завдяки життєво-цикловій парадигмі на основі ProcessCameraProvider, досягнуто високої стабільності та енергоефективності системи, де активація апаратних ресурсів суворо корелює зі станом активності застосунку. Ключовим досягненням стала багатопотокова стратегія, яка ізолює математичні трансформації піксельної матриці, операції введення-виведення та візуалізацію кешованих даних від головного потоку, гарантуючи плавність інтерфейсу. Проектування конвеєра через абстракцію «програмного слота» та впровадження механізмів збереження стану забезпечує стабільну роботу з різними модулями камер та надійне відновлення контексту після змін конфігурації пристрою. Фінальна модель збереження через MediaStore API забезпечує відповідність стандартам безпеки Android та цілісність файлів. Створений каркас не лише розв'язує задачі фільтрації зображень, а й закладає фундамент для подальшого розширення функціоналу системи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

3.1. Стек технологій

Для практичної реалізації спроектованої архітектури було обрано сучасний стек інструментів розробки, який забезпечує максимальну сумісність із платформою Android та високу продуктивність при роботі з медіаданими.

В якості середовищам розробки було обрано Android Studio. Це зумовлено тим, що вона є офіційною та основною IDE для ОС Android, яка забезпечує найповнішу підтримку екосистеми Google. Це дозволило використовувати інтегровані інструменти профілювання, які в реальному часі відстежують навантаження на центральний процесор та використання оперативної пам'яті під час роботи камери та асинхронного запису файлів. Використання саме цього IDE гарантує повну сумісність із системними бібліотеками Jetpack та коректну роботу налагоджувача при аналізі конвеєрів обробки зображень.

Основними технологічними компонентами проекту є:

- мова програмування Java – обрана як фундаментальна мова розробки для Android. Використання Java [12] дозволило чітко реалізувати принципи об’єктно-орієнтованого проектування, описані в другому розділі, та забезпечити стабільну роботу з низькорівневими буферами даних у модулях аналізу зображень;
- бібліотека Jetpack CameraX (v. 1.3.4) – вибір стабільної версії зумовлений її надійністю та повною підтримкою функцій ImageAnalysis та ImageCapture, які є критичними для реалізації конвеєра обробки даних. Використання CameraX дозволило автоматизувати управління ресурсами сенсора відповідно до життєвого циклу застосунку, що мінімізує ризики витоку пам'яті та конфліктів при зверненні до апаратного забезпечення;
- бібліотека Glide (v. 4.16) – використана як основний інструмент для обробки та візуалізації медіа-ресурсів. В межах проекту Glide відповідає за

асинхронне декодування створених JPEG-файлів, їх кадрування та генерацію мініатюр, що запобігає блокуванню головного потоку при роботі з дисковою підсистемою;

- спеціалізована бібліотека Google Guava (ListenableFuture) – інтегрована для управління асинхронними операціями. Оскільки ініціалізація камери в Android є ресурсомістким процесом, що виконується у фоновому режимі [13], використання об'єкта ListenableFuture дозволяє системі «підписатися» на результат підключення до сенсорів. Це гарантує, що застосунок не заблокує графічний інтерфейс під час очікування відповіді від апаратної частини, а розгорне камеру лише тоді, коли внутрішні сервіси будуть повністю готові до роботи.

```
val camerax_version = "1.3.4"
implementation ("com.google.guava:guava:30.1.1-android")
implementation("androidx.camera:camera-view:$camerax_version")
implementation ("androidx.camera:camera-camera2:$camerax_version")
implementation ("androidx.camera:camera-lifecycle:$camerax_version")
implementation ("androidx.camera:camera-extensions:$camerax_version")
implementation ("androidx.camera:camera-core:$camerax_version")
implementation("com.github.bumptech.glide:glide:4.16.0")
```

Рисунок 3.1. Фрагмент файлу build.gradle із конфігурацією програмних залежностей проекту (бібліотеки Jetpack CameraX, Google Guava та Glide).

Застосунок розроблено з використанням Android SDK 36, що дозволяє інтегрувати найновіші протоколи безпеки та оптимізовані механізми роботи з медіаконтентом. При цьому, встановлений поріг мінімальної версії SDK 26 (Android 8.0) гарантує працездатність системи на більшості актуальних мобільних пристроїв. Вибір SDK 26 як базового рівня є обґрунтованим інженерним рішенням, оскільки саме з цієї версії в ОС Android забезпечується стабільна підтримка сучасних графічних архітектур та розширене керування фоновими завданнями, що є критичним для застосунків, які активно використовують апаратні ресурси камери.

Отже, інтеграція Android Studio (SDK 26–36), мови Java та бібліотек CameraX, Glide і Guava дозволила створити оптимізований застосунок з асинхронним конвеєром обробки медіаданих без блокування інтерфейсу. Такий вибір компонентів гарантує стабільну роботу системи на більшості актуальних пристроїв та високу швидкість взаємодії користувача з апаратною частиною камери.

3.2. Інтерфейс користувача

Проектування інтерфейсу базується на використанні контейнера ConstraintLayout, що дозволяє створити сучасну, гнучку та продуктивну візуальну ієрархію з мінімальною вкладеністю елементів. Головним компонентом виступає PreviewView, який інтегрований у верстку як базовий повноекранний шар для трансляції відеопотоку з камери в режимі реального часу.

Під площиною попереднього перегляду реалізовано блок керування, що складається з трьох ключових елементів, розташованих у нижній частині екрана:

- центральний елемент – кнопка зйомки (captureButton), яка виконує основний функціонал застосунку. Вона займає центральну позицію по горизонталі, забезпечуючи зручний доступ для користувача, незалежно від того, якою рукою він тримає пристрій. Таке розміщення обумовлене стандартами ергономіки мобільних інтерфейсів, оскільки центральна зона в нижній частині екрана є найбільш доступною для швидкого натискання, що дозволяє стабільно утримувати смартфон під час зйомки;
- лівий елемент – інтерактивний компонент мініатюри (photoThumbnail). Кнопка має умовну активність, виконуючи роль візуального індикатора останнього створеного знімка. Програмна логіка передбачає, що цей компонент стає активним та заповнюється контентом лише після першого успішного натискання кнопки зйомки, створюючи чіткий зворотний

зв’язок про виконання операції. Розташування ліворуч від центру є класичним рішенням для мобільних інтерфейсів, що дозволяє візуально відокремити архівний контент (минулі дії) від органів активного керування сенсором, забезпечуючи інтуїтивну навігацію між режимом зйомки та режимом перегляду;

- правий елемент – кнопка динамічного перемикання камер (flipCameraButton). Цей компонент відповідає за зміну активного оптичного модуля між основним та фронтальним сенсорами. Розміщення кнопки праворуч від центрального вузла зйомки створює необхідний візуальний баланс та завершує симетрію блоку керування. Такий розподіл дозволяє розмежувати зони відповідальності: у той час як ліва сторона фокусується на роботі з уже створеними даними, права сторона відведена під оперативне керування параметрами самого вхідного сигналу. Це мінімізує ризик випадкових натискань та забезпечує швидку реконфігурацію медіапотоку безпосередньо в процесі роботи з пристроєм.

Рисунок 3.2. UI мобільного застосунку.

Особлива увага в програмній реалізації приділена обробці системних відступів (Window Insets), оскільки у сучасних версіях Android, зокрема при використанні SDK 36, системні панелі навігації та статусу можуть частково або повністю перекривати інтерактивні елементи керування. Для вирішення цієї проблеми в проекті впроваджено слухач OnApplyWindowInsetsListener, який забезпечує динамічний розрахунок параметрів інтерфейсу. Програма в реальному часі отримує об'єкт Insets, що містить точні розміри системних панелей у пікселях для конкретного пристрою, на основі чого виконується адаптація відступів. Зокрема, нижній відступ кнопки зйомки автоматично розраховується за формулою systemBars.bottom + 30 пікселів. Такий підхід гарантує, що кнопка завжди буде знаходитися над панеллю жестів або системними кнопками навігації, забезпечуючи коректний користувацький досвід та стабільну працездатність інтерфейсу на пристроях із різними конфігураціями екранів.

Таким чином, застосування плоского контейнера ConstraintLayout та повноекранного шару PreviewView забезпечило створення високопродуктивного та гнучкого графічного інтерфейсу камери. Ергономічне симетричне розташування елементів нижнього блоку керування чітко розмежувало функції роботи з медіаданими та налаштування оптичних модулів. Водночас динамічна обробка системних відступів через OnApplyWindowInsetsListener гарантує стабільну доступність усіх інтерактивних компонентів на екранах будь-якої конфігурації.

3.3. Налаштування та перевірка дозволів доступу

Функціонування розробленого застосунку критично залежить від наявності апаратного модуля камери, що чітко прописано на рівні маніфесту застосунку. Це вказує системі на неможливість роботи додатка без фізичного сенсора зйомки. Додатково в налаштуваннях параметрів апаратного забезпечення

вказано, що наявність системи автоматичного фокусування не є обов'язковою. Такий підхід забезпечує максимальну сумісність: застосунок використовує можливості автофокусу для покращення чіткості зображень на сучасних сенсорах, проте залишається повністю працездатним і на бюджетних пристроях з оптикою фіксованого фокусу.

Оскільки робота з камерою та файловою системою належить до критичних операцій, у проекті реалізовано систему запиту дозволів безпосередньо під час виконання програми. Це гарантує, що додаток не отримає доступу до приватних даних користувача без його явного підтвердження.

Зважаючи на це, програмна реалізація базується на адаптивному масиві REQUIRED_PERMISSIONS. Для забезпечення сумісності з різними версіями Android застосовано умовну логіку: на пристроях до Android 9 включно запитується доступ до камери та зовнішнього сховища (WRITE_EXTERNAL_STORAGE), тоді як для новіших версій (API 29+) – лише до камери. Це обумовлено переходом ОС на архітектуру Scoped Storage, яка не потребує явного дозволу на запис у власні медіа-директорії.

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" android:maxSdkVersion="28"/>
<uses-feature android:name="android.hardware.camera" android:required="true"/>
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
```

Рисунок. 3.3. Фрагмент файлу AndroidManifest із налаштуванням доступу до апаратних ресурсів.

Перевірка статусу здійснюється за допомогою методу allPermissionsGranted(), який використовує функцію ContextCompat.checkSelfPermission. У разі відсутності дозволу викликається системне вікно запиту через ActivityCompat.requestPermissions, що призупиняє активність до отримання відповіді від користувача.

```
private boolean allPermissionsGranted() {
    for (String permission : REQUIRED_PERMISSIONS) {
        if (ContextCompat.checkSelfPermission( context: this, permission) != PackageManager.PERMISSION_GRANTED) {
            return false;
        }
    }
    return true;
}
```

Рисунок 3.4. Метод allPermissionsGranted() для перевірки статусу наданих дозволів.

Кінцевий результат обробляється у callback-методі onRequestPermissionsResult, де за допомогою коду ідентифікації 101 застосунок визначає подальший хід виконання: ініціалізація камери або завершення роботи додатка з відповідним сповіщенням. Такий підхід забезпечує стабільну роботу застосунку та відповідає сучасним стандартам безпеки Android.

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == REQUEST_CODE_PERMISSIONS) {
        boolean allGranted = true;
        for (int result : grantResults) {
            if (result != PackageManager.PERMISSION_GRANTED) {
                allGranted = false;
                break;
            }
        }
        if (grantResults.length > 0 && allGranted) {
            Toast.makeText( context: this,  text: "Всі дозволи надано", Toast.LENGTH_SHORT).show();
            startCamera();
        } else { // завершення роботи програми, якщо дозволи не надано
            Toast.makeText( context: this,  text: "Не надано всі дозволи", Toast.LENGTH_SHORT).show();
            finish();
        }
    }
}
```

Рисунок 3.5. Метод onRequestPermissionsResult() для обробки відповіді користувача на запит дозволів.

Таким чином, реалізована адаптивна система перевірки та динамічного запиту дозволів забезпечує високий рівень безпеки й стабільне керування апаратними ресурсами. Завдяки розмежуванню логіки доступу до сховища між різними поколіннями API (до і після впровадження Scoped Storage) додаток

суворо дотримується сучасних політик конфіденційності Android. Водночас гнучкі вимоги до фокусування та залізна інтеграція обробників у життєвий цикл гарантують передбачувану поведінку програми на пристроях із будь-яким рівнем апаратного забезпечення.

3.4. Реалізація фотозйомки

Центральним функціональним блоком застосунку є метод takePhoto(), реалізація якого базується на чітко визначеній послідовності операцій обробки даних. Процес створення знімка починається з підготовки логічної структури майбутнього файлу в пам'яті пристрою. Для формування унікальних імен файлів використано клас SimpleDateFormat із шаблоном yyyyMMdd_HHmmss, що запобігає конфліктам імен при серійній зйомці та забезпечує хронологічне сортування в галереї. Замість прямого маніпулювання файловою системою, у проекті застосовано MediaStore API через об’єкт-контейнер ContentValues. Це дозволяє передати системі Android ключові метадані: відображуване ім'я файлу та його MIME-тип. Важливою особливістю є врахування архітектури Scoped Storage для пристроїв з Android 10 та вище: за допомогою ключа RELATIVE_PATH фотографії спрямовуються у виділену директорію Pictures/CameraX-Photos, що гарантує коректну роботу з пам’яттю згідно з сучасними вимогами безпеки.

```
String namePhoto = new SimpleDateFormat( pattern: "yyyyMMdd_HHmmss", Locale.US).format(new Date());
ContentValues contentValues = new ContentValues();
contentValues.put(MediaStore.MediaColumns.DISPLAY_NAME, namePhoto);
contentValues.put(MediaStore.MediaColumns.MIME_TYPE, "image/jpeg");
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
    contentValues.put(MediaStore.Images.Media.RELATIVE_PATH, "Pictures/CameraX-Photos");
}
ImageCapture.Metadata metadata = new ImageCapture.Metadata();
metadata.setReversedHorizontal(lensFacing == CameraSelector.LENS_FACING_FRONT);
ImageCapture.OutputFileOptions outputFileOptions = new ImageCapture.OutputFileOptions.Builder(
        getContentResolver(),
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
        contentValues
)
.setMetadata(metadata)
.build();
```

Рисунок 3.6. Фрагмент коду методу takePhoto() для налаштування метаданих, параметрів сховища та корекції дзеркального ефекту.

Після налаштування параметрів збереження ініціюється метод imageCapture.takePicture(). Його роботу реалізовано як асинхронну операцію, що виконується в окремому потоці за допомогою ContextCompat.getMainExecutor(this). Такий підхід є критично важливим для мобільних додатків, оскільки процес запису важких медіаданих на диск не повинен блокувати головний потік інтерфейсу. Для керування процесом запису створюється об’єкт OutputFileOptions, який об’єднує підготовлені метадані та інструкції для бібліотеки CameraX. Зокрема, у метадані записується стан дзеркального відображення для фронтальної камери, щоб отримане «селфі» відповідало вигляду на екрані попереднього перегляду.

```
imageCapture.takePicture(outputFileOptions, ContextCompat.getMainExecutor( context: this),
        new ImageCapture.OnImageSavedCallback() {
            @Override
            public void onImageSaved(@NonNull ImageCapture.OutputFileResults outputFileResults) {
                lastSavedUri = outputFileResults.getSavedUri();
                Glide.with( activity: MainActivity.this) RequestManager
                        .load(lastSavedUri) RequestBuilder<Drawable>
                        .placeholder(android.R.color.darker_gray)
                        .circleCrop()
                        .into(photoThumbnail);
                Toast.makeText( context: MainActivity.this,  text: "Фото збережено: " + lastSavedUri, Toast.LENGTH_SHORT).show();
            }
            @Override
            public void onError(@NonNull ImageCaptureException exception) {
                Toast.makeText( context: MainActivity.this,  text: "Помилка: " + exception.getMessage(), Toast.LENGTH_SHORT).show();
            }
        });
```

Рисунок 3.7. Метод imageCapture.takePicture() для асинхронного виконання зйомки та візуалізації результату через зворотний виклик.

Контроль за результатом операції забезпечується через інтерфейс зворотного виклику OnImageSavedCallback. При успішному завершенні запису метод onImageSaved повертає сформований системою Uri, який стає ключовим для подальшої візуалізації. Для миттєвого оновлення мініатюри останнього знімка в інтерфейсі інтегровано бібліотеку Glide. Вона виконує асинхронне декодування зображення, що дозволяє уникнути переповнення оперативної пам'яті (OutOfMemory Error) при роботі з файлами високої роздільної здатності. За допомогою методу circleCrop() бібліотека виконує програмну кругову обрізку зображення безпосередньо під час завантаження, що забезпечує естетичний вигляд кнопки-мініатюри. У разі виникнення помилок (напр., апаратного збою або відсутності вільного місця) спрацьовує метод onError, який перехоплює виняток і виводить діагностичне повідомлення користувачу, зберігаючи при цьому стабільність роботи всього застосунку.

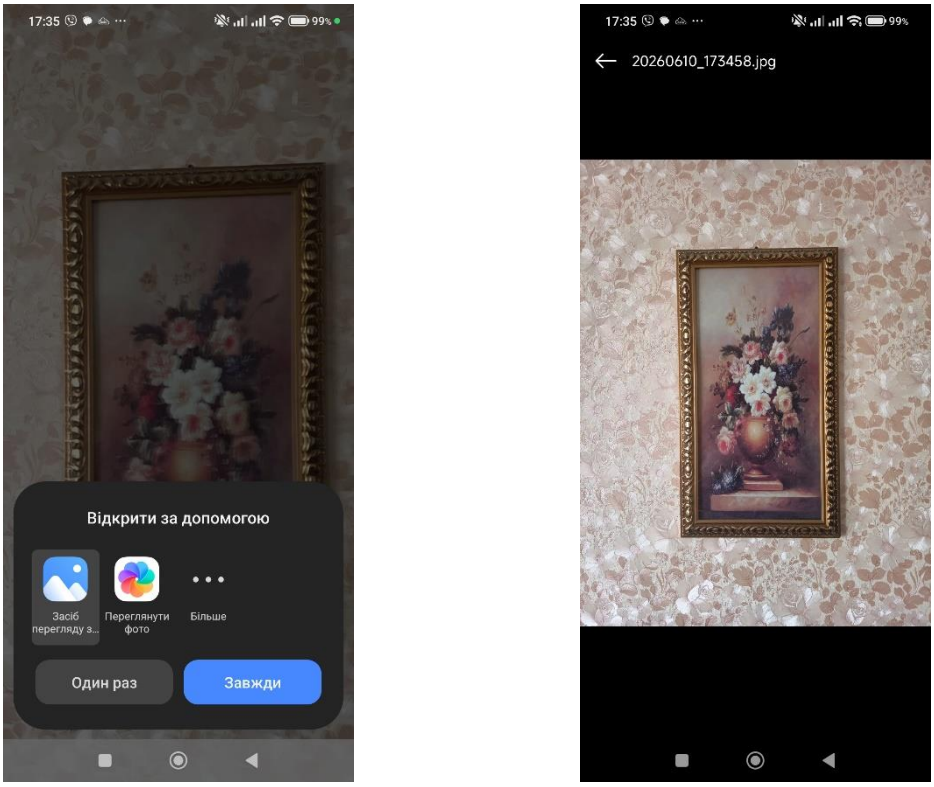
Рисунок 3.8. Збережена кнопка-мініатюра.

Окрім безпосереднього збереження знімка, у застосунку реалізовано додаткову функціональність для покращення взаємодії з користувачем. Вона дозволяє переглянути створене фото у повному розмірі за допомогою методу openGallery(), який викликається при натисканні на кнопку-мініатюру. В основі цього рішення лежить використання системних інтентів. Такий підхід дозволяє не перевантажувати додаток власними інструментами для перегляду медіафайлів, а делегувати це завдання стороннім спеціалізованим програмам-галереям, які вже є на пристрої.

```
private void openGallery() {
    if (lastSavedUri != null) {
         android.content.Intent intent = new android.content.Intent(android.content.Intent.ACTION_VIEW);
        intent.setDataAndType(lastSavedUri,  type: "image/jpeg");
        intent.addFlags(android.content.Intent.FLAG_GRANT_READ_URI_PERMISSION);
        try {
           startActivity(intent);
        } catch (android.content.ActivityNotFoundException e) {
            Toast.makeText( context: this,  text: "Додаток для перегляду фото не знайдено", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Рисунок 3.9. Метод openGallery() для делегування перегляду зображення зовнішнім застосункам за допомогою системи інтентів.

Програмна реалізація базується на дії ACTION_VIEW, яка сигналізує ОС про необхідність знайти відповідний застосунок для відображення переданих даних. Важливою технічною деталлю є використання прапорця FLAG_GRANT_READ_URI_PERMISSION. Оскільки сучасні версії Android використовують модель ізольованого сховища (пісочниці), цей прапорець надає сторонній програмі тимчасовий дозвіл на читання конкретного файлу за його Uri. Для забезпечення відмовостійкості реалізовано перехоплення винятку ActivityNotFoundException, що гарантує стабільну роботу програми навіть у випадку, якщо на пристрої відсутні засоби перегляду зображень.



а) б)

Рисунок 3.10. Вибір стороннього застосунку (а) для перегляду останнього збереженого фото (б).

Для забезпечення можливості динамічної зміни активного оптичного модуля пристрою, в застосунку реалізовано метод перемикання камер, в основі

якого лежить логіка циклічного перерахунку та переініціалізації константи lensFacing. Процес перемикання ініціюється користувачем через графічний елемент інтерфейсу і керується подійною моделлю. Оскільки бібліотека CameraX оперує абстракціями високого рівня, безпосередній вибір апаратного сенсора здійснюється через об'єкт CameraSelector, конфігурація якого прямо залежить від поточного стану змінної lensFacing. Математична та логічна суть перерахунку полягає в інверсії поточного значення: за допомогою умовного оператора програма перевіряє поточну константу, і якщо активною була основна камера (CameraSelector.LENS_FACING_BACK), значення динамічно змінюється на фронтальну (CameraSelector.LENS_FACING_FRONT), і навпаки. Зміна значення змінної є лише підготовчим етапом, оскільки для фізичного перемикання відеопотоку необхідна повна перебудова конвеєра даних. Для цього, викликається метод прив'язки камери до життєвого циклу додатка, де спочатку за допомогою cameraProvider.unbindAll() примусово розриваються всі попередні зв'язки для поточного об'єктива (відключаються Preview та ImageCapture). Після очищення ресурсів створюється новий екземпляр CameraSelector із оновленим значенням lensFacing. Нова конфігурація асинхронно передається до ProcessCameraProvider, який повторно зв'язує оновлений селектор камери з життєвим циклом поточної активності, забезпечуючи плавне перенаправлення медіапотоку з іншого сенсора на існуючий графічний елемент PreviewView.

Прямим наслідком активації фронтального оптичного модуля є необхідність програмної корекції отриманого зображення для збереження природного сприйняття кадру користувачем. За замовчуванням, під час зйомки на передню камеру Android-пристрої відображають картинку на екрані дзеркально (як у звичайному дзеркалі), проте апаратна матриця фіксує «реальне» (неперевернуте) зображення. Якщо зберегти його без змін, фінальний JPEG-файл у Галереї виявиться інвертованим, що призведе до ефекту «перевернутого тексту» та порушення композиції. Для усунення цього дефекту, в момент перемикання на фронтальну камеру активується логіка

модифікації метаданих знімка за допомогою методу metadata.setReversedHorizontal(true). Ця команда не виконує важких попіксельних операцій над матрицею «на льоту», що могло б уповільнити роботу додатка. Замість цього, вона записує спеціальний прапорець інверсії безпосередньо в конфігураційний об'єкт ImageCapture.Metadata. Під час фіксації кадру CameraX зчитує цей параметр і застосовує апаратне горизонтальне віддзеркалення безпосередньо в процесі кодування масиву байтів у формат JPEG. У результаті такого причинно-наслідкового зв'язку збережена фотографія в публічному медіа-сховищі повністю відповідає тому зображенню, яке користувач бачив на екрані попереднього перегляду в момент натискання кнопки зйомки.

Для забезпечення безперервності користувацького досвіду при зміні конфігурації пристрою (зокрема, під час повороту екрана, який викликає повне руйнування та повторне створення поточної активності), у проекті реалізовано комплексний механізм збереження стану системи за допомогою контейнера Bundle. У межах перевизначеного методу onSaveInstanceState() здійснюється фіксація критично важливих параметрів сесії: унікального ідентифікатора ресурсу (lastSavedUri) та поточної конфігурації об'єктива (lensFacing), що визначає вибір між основною та фронтальною камерами. Під час повторної ініціалізації компонента через метод onRestoreInstanceState() або конструктор onCreate(), збережені дані вилучаються з тимчасового сховища для відновлення логіки додатка. На основі вище згаданого елемента lensFacing, система автоматично перезапускає життєвий цикл ProcessCameraProvider, ініціюючи коректний вибір модуля камери без участі користувача. Паралельно з цим, збережений шлях lastSavedUri передається у фоновий потік, де здійснюється асинхронне відновлення графічного елемента кнопки-мініатюри засобами бібліотеки Glide, що запобігає блокуванню UI-потоку під час декодування зображення та виключає появу порожніх графічних контейнерів або скидання налаштувань інтерфейсу до початкового стану.

Рисунок 3.11. Візуалізація збереження стану UI-компонентів при повороті екрана

Таким чином, програмна реалізація фотозйомки базується на побудові відмовостійкого асинхронного конвеєра обробки та збереження медіаданих. Використання MediaStore API у поєднанні з архітектурою Scoped Storage та інструментами Glide дозволило повністю ізолювати важкі дискові операції й процеси декодування зображень від головного потоку, зберігши максимальну плавність інтерфейсу. Водночас інтеграція механізмів віддзеркалення фронтальної матриці, безпечної міжпроцесної трансляції інтенту ACTION_VIEW та фіксації стану сесії через Bundle забезпечила створення ергономічного, стабільного та захищеного застосунку, повністю адаптованого до вимог сучасних версій ОС Android.

3.5. Цифрова обробка медіаданих

Процес фіксації зображення мобільним пристроєм та його подальша візуалізація в інтерфейсі застосунку потребують застосування методів цифрової обробки сигналів і оптимізації растрових даних. Першим етапом цифрової модифікації є трансформація сирого масиву даних з апаратної матриці

(фронтального оптичного модуля) у кінцевий стиснутий файл. За замовчуванням сенсор фіксує «реальне» зображення, тоді як на екрані попереднього перегляду для природного сприйняття воно відображається дзеркально. Без додаткової обробки це призводить до інверсії тексту та порушення геометрії сцени у фінальному файлі.

Для усунення цього ефекту, на рівні конвеєра обробки даних, застосовано низькорівневу модифікацію структури файлу за допомогою ImageCapture.Metadata. Метод setReversedHorizontal(true) ініціює внесення специфічного прапорця інверсії безпосередньо в заголовки метаданих перед початком компресії. Під час фіксації кадру підсистема CameraX зчитує цей параметр і виконує апаратне горизонтальне віддзеркалення безпосередньо в процесі математичного кодування масиву байтів у формат JPEG. Такий підхід дозволяє уникнути ресурсомістких програмних попіксельних операцій над уже готовим растром, мінімізуючи навантаження на центральний процесор пристрою, і гарантує, що результуючий файл повністю відповідатиме зображенню з екрана попереднього перегляду.

Додатковим критично важливим аспектом низькорівневої підготовки медіаданих на рівні конвеєра є динамічна корекція просторової орієнтації кадру. Оскільки апаратний сенсор мобільного пристрою є статично фіксованим, зміна положення смартфона у просторі (альбомна чи портретна орієнтація) без програмного втручання призводить до збереження растра із порушенням кута відображення відносно користувача. Для нівелювання цього ефекту в архітектурі підсистеми CameraX реалізовано динамічний розрахунок параметра targetRotation. Інформація про поточний кут нахилу пристрою зчитується в режимі реального часу через слухач орієнтації (OrientationEventListener) і трансформується у константи повороту (180°, 90°, 0° або 270°). Обчислена конфігурація цільового повороту передається в об'єкт ImageCapture, що змушує підсистему апаратно скоригувати орієнтацію масиву байтів безпосередньо під час фінальної компресії даних. Такий підхід гарантує, що результуючий JPEG-файл у сховищі матиме правильну орієнтацію та буде коректно

інтерпретований системними в'юверами Галереї незалежно від фізичного кута нахилу пристрою в момент зйомки.

Другим етапом цифрової обробки є динамічне перетворення отриманого JPEG-файлу для потреб інтерфейсу користувача. Оскільки сучасні фотоматриці генерують зображення високої роздільної здатності з великим об'ємом даних, спроба прямого завантаження такого растра в оперативну пам'ять для відображення мініатюри неминуче призводить до критичної помилки переповнення пам'яті (OutOfMemoryError). Для запобігання цьому в застосунку реалізовано асинхронний конвеєр обробки зображень на базі бібліотеки Glide.

Варто зазначити, що загальна інфраструктура цифрової обробки медіапотоку закладається ще на етапі конфігурації юзкейсів камери, описаних у попередньому підрозділі 3.4. Зокрема, інтеграція підсистеми ImageAnalysis забезпечує розширену детекцію та обробку кадрів на рівні окремих буферів зображень. Хоча в поточній версії застосунку пріоритет віддано оптимізації статичних даних, архітектурна наявність ImageAnalysis створює ізольований шар для подальшого масштабування системи, що дозволяє інтегрувати будь-які методи фільтрації (напр., бінаризацію, перетворення у градації сірого, тощо) або модулі комп'ютерного зору безпосередньо в процесі зчитування растрового зображення.

Загалому, цифровий алгоритм обробки в цьому контексті складається з трьох послідовних кроків:

- даунсемплінг – програмне декодування стиснутого JPEG-потоку виконується адаптивно. Glide обчислює коефіцієнт масштабування та зчитує з файлу лише ту кількість пікселів, яка необхідна для фізичного розміру графічного контейнера кнопки-мініатюри, суттєво зменшуючи матрицю масиву Bitmap в оперативній пам'яті;
- растрове перетворення – за допомогою методу circleCrop() над декодованою матрицею пікселів «на льоту» виконується математична операція обрізки. Алгоритм накладає кругову маску на прямокутний масив

даних, занулюючи альфа-канал (прозорість) пікселів, що виходять за межі радіуса, для досягнення необхідного візуального ефекту;

- кешування обробленого растрового зображення – результат цифрової трансформації зберігається у виділеному кеші оперативної пам'яті. Це виключає потребу в повторному декодуванні та обробці важкого JPEG-файлу при відновленні стану інтерфейсу (напр., під час зміни орієнтації екрана), забезпечуючи миттєвий рендеринг графіки.

Рисунок 3.12. Схема конвеєра цифрової обробки зображення: від апаратного кодування до растрової трансформації.

Таким чином, реалізований комплекс методів цифрової обробки забезпечив ефективну оптимізацію медіапотоку на всіх етапах його проходження. Низькорівневе керування метаданими кадру дозволило апаратно усунути дефекти дзеркальності та просторової дезорієнтації знімків без додаткового навантаження на центральний процесор. Водночас застосування триетапного алгоритму фільтрації, масштабування та кешування растрових

даних повністю убезпечило систему від переповнення оперативної пам'яті та заклало надійну архітектурну основу для подальшого розширення функціоналу комп'ютерного зору.

3.6. Тестування та оцінка продуктивності

Фінальним етапом розробки програмного забезпечення є проведення комплексу тестових випробувань для перевірки коректності функціонування архітектурних рішень та оцінки параметрів продуктивності застосунку в реальних умовах експлуатації. Головною метою тестування було підтвердження стабільності роботи інтерфейсу, перевірка збереження стану компонентів під час зміни конфігурації пристрою та аналіз сумісності підсистеми камери.

Тестування обробки подій життєвого циклу. Одним із найбільш критичних аспектів для мобільних систем на базі ОС Android є коректна обробка деструкції та повторного створення об'єктів Activity і Fragment під час зміни конфігурацій (зокрема, зміни орієнтації екрана з портретної на альбомну або примусового згортання застосунку у фоновий режим). Тестування проводилося шляхом багаторазового циклічного повороту пристрою безпосередньо під час активного стримінгу відеопотоку з камери та під час збереження медіафайлів.

Результати тестування підтвердили, що завдяки інтеграції архітектурних компонентів, чутливих до життєвого циклу (Lifecycle-aware components), система безпосередньо керує станом конвеєра. При згортанні застосунку ресурси апаратної матриці камери миттєво вивільняються, що запобігає блокуванню сенсора іншими процесами та економить заряд акумулятора. При поверненні до застосунку або зміні орієнтації екрана сесія ProcessCameraProvider ініціалізується повторно без візуальних затримок, а збережений шлях до поточного файлу та стан мініатюри інтерфейсу користувача повністю відновлюються із пам'яті без втрати даних.

Оцінка ізоляції міжпроцесної взаємодії та безпеки контексту. Окремим вектором випробувань став аналіз поведінки застосунку при взаємодії із зовнішніми системними компонентами, зокрема під час трансляції інтенту ACTION_VIEW для перегляду знімків у сторонній Галереї. Тестування зафіксувало автономну деструкцію процесу Галереї при переході ОС у фоновий режим (натискання системної кнопки «Додому»). Було встановлено, що дана поведінка є не архітектурною помилкою застосунку, а результатом штатної роботи підсистеми безпеки та оптимізації сучасних версій ОС Android. З метою економії оперативної пам'яті (захист від витоків ресурсів під час роботи CameraX) та дотримання концепції ізольованих сховищ Scoped Storage, ОС примусово відкликає тимчасові права на читання файлу (FLAG_GRANT_READ_URI_PERMISSION) у сторонніх утиліт при згортанні таску. Для забезпечення нативності інтерфейсу, збереження цілісності стеку задач та попередження конфліктів у пам'яті, у проєкті було свідомо реалізовано стандартну безпрапорцеву модель передачі керування. Це гарантує, що при повторному розгортанні застосунку з робочого столу користувач повертається до пріоритетного потоку камери, тоді як закриття Галереї кнопкою «Назад» коректно відновлює попередній стан MainActivity без деструкції поточних даних сесії.

Аналіз стабільності та плавності інтерфейсу. Оскільки операції запису важких JPEG-файлів у сховище за допомогою MediaStore API та наступний рендеринг растрових зображень є ресурсомісткими дисковими та обчислювальними процесами, було проведено профілювання додатка за допомогою інструменту Android Profiler. Основним критерієм оцінки була плавність відтворення графічного інтерфейсу (FPS).

Метрики профілювання показали, що пряме винесення логіки запису файлів у сховище та асинхронного конвеєра обробки зображень бібліотеки Glide в окремі фонові потоки виконавчого пулу повністю розвантажило головний потік застосунку Main Thread. Під час інтенсивної фіксації кадрів та паралельного збереження даних графічний інтерфейс користувача зберігає

стабільну частоту оновлення на рівні 60 FPS без мікрофризів чи затримок рендерингу. Одночасно з цим, реалізований алгоритм адаптивного даунсемплігу в Glide дозволив утримати споживання оперативної пам'яті у межах безпечної норми, повністю виключивши ризик виникнення критичних помилок переповнення пам'яті OutOfMemoryError.

Аналіз сумісності та кросплатформеності. Суттєвою проблемою розробки під екосистему Android є високий рівень фрагментації ринку апаратного забезпечення (різні виробники камер, драйвери та специфікації оптичних модулів). Для перевірки кросплатформеної стійкості було проведено тестування застосунку на базі хмарних тестових лабораторій (Firebase Test Lab) та фізичних пристроях різних брендів (включаючи Note-серію під управлінням сучасних систем HyperOS та Android).

Використання високорівневої абстракції Jetpack CameraX дозволило перенести вирішення внутрішніх помилок сумісності на рівень самої бібліотеки Google. Застосунок успішно пройшов випробування на пристроях із різними апаратними конфігураціями та специфікаціями оптичних модулів. За попередніми розрахунками та статистикою сумісності API, розроблене програмне забезпечення гарантує стабільну, передбачувану та ідентичну роботу конвеєра цифрової обробки та захоплення медіаданих на більш ніж 90 % сучасних активних Android-пристроїв у світі.

Отже, проведене комплексне тестування підтвердило високу стабільність, безпеку та продуктивність розробленого мобільного застосунку. Завдяки інтеграції компонентів, чутливих до життєвого циклу, асинхронному винесенню важких дискових операцій у фонові потоки та оптимізації растрових даних, вдалося досягти стабільної плавності інтерфейсу на рівні 60 FPS та виключити ризики переповнення пам'яті. Успішні випробування у Firebase Test Lab та на фізичних пристроях довели високу кросплатформену стійкість системи, що гарантує ідентичну й передбачувану роботу конвеєра зйомки на більшості актуальних пристроїв під управлінням ОС Android.

Таким чином, у третьому розділі виконано практичну реалізацію та комплексне тестування мобільного застосунку для фотозйомки на базі середовища Android Studio, мови Java та сучасного SDK 36. Інтеграція високорівневої архітектури Jetpack CameraX у поєднанні з бібліотеками Glide та Google Guava дозволила створити відмовостійкий асинхронний конвеєр обробки медіаданих, повністю ізольований від головного потоку інтерфейсу. Спроектований на базі ConstraintLayout ергономічний інтерфейс із динамічною обробкою відступів OnApplyWindowInsetsListener, адаптивною системою runtime-дозволів та механізмами збереження стану через Bundle забезпечив стабільну роботу на пристроях із будь-якою конфігурацією екрана. Низькорівневе керування метаданими кадру апаратно усунуло дефекти дзеркальності й дезорієнтації знімків без навантаження на CPU, а триетапний алгоритм обробки растра у Glide повністю убезпечив систему від помилок OutOfMemoryError. Фінальне профілювання та кросплатформені випробування у Firebase Test Lab підтвердили високу безпеку, автономність керування ресурсами камери та стабільну плавність графічного інтерфейсу на рівні 60 FPS, що гарантує передбачувану роботу застосунку на більш ніж 90 % активних Android-пристроїв.

ВИСНОВКИ

У даній роботі здійснено проєктування, програмну реалізацію та інструментальне тестування Android-застосунку для фотозйомки та цифрової обробки медіаданих. На основі виконаних етапів розробки та отриманих експериментальних даних сформульовано наступні висновки:

1. Здійснено теоретичний аналіз предметної області мобільної розробки, методів взаємодії з апаратними медіасенсорами та специфіки фрагментації ринку Android-пристроїв, що дозволило обґрунтувати вибір архітектурної абстракції Jetpack CameraX для забезпечення високої кросплатформеної сумісності.

2. Спроектована архітектура на основі Use Case-моделі та інструментів асинхронності забезпечила повну ізоляцію важких дискових операцій від головного потоку інтерфейсу, що дозволило мінімізувати ризики блокування інтерфейсу під час роботи з медіапотоками, заклавши основу для стабільного функціонування та відмовостійкості системи.

3. Розроблено та реалізовано мобільний застосунок, основними функціями якого є високоефективне захоплення статичних зображень, динамічне перемикання між фронтальним та основним оптичними модулями, автоматична адаптація графічного інтерфейсу під конфігурацію пристрою, а також безпечне збереження файлів з урахуванням сучасних runtime-політик.

4. Інтегровано конвеєр цифрової обробки медіаданих, що базується на модифікації метаданих знімка для апаратного усунення дефектів дзеркальності й просторової дезорієнтації, а також на механізмах оптимізації завантаження зображень (масштабування, геометричне маскування та кешування) задля захисту оперативної пам'яті від переповнення.

5. Проведено комплексне тестування життєвого циклу компонентів, перевірку кросплатформеної стійкості програми на різноманітних апаратних конфігураціях, а також інструментальне профілювання системи, яке

підтвердило відсутність витоків пам'яті та зафіксувало високу стабільність і плавність відтворення графічного інтерфейсу користувача під навантаженням.

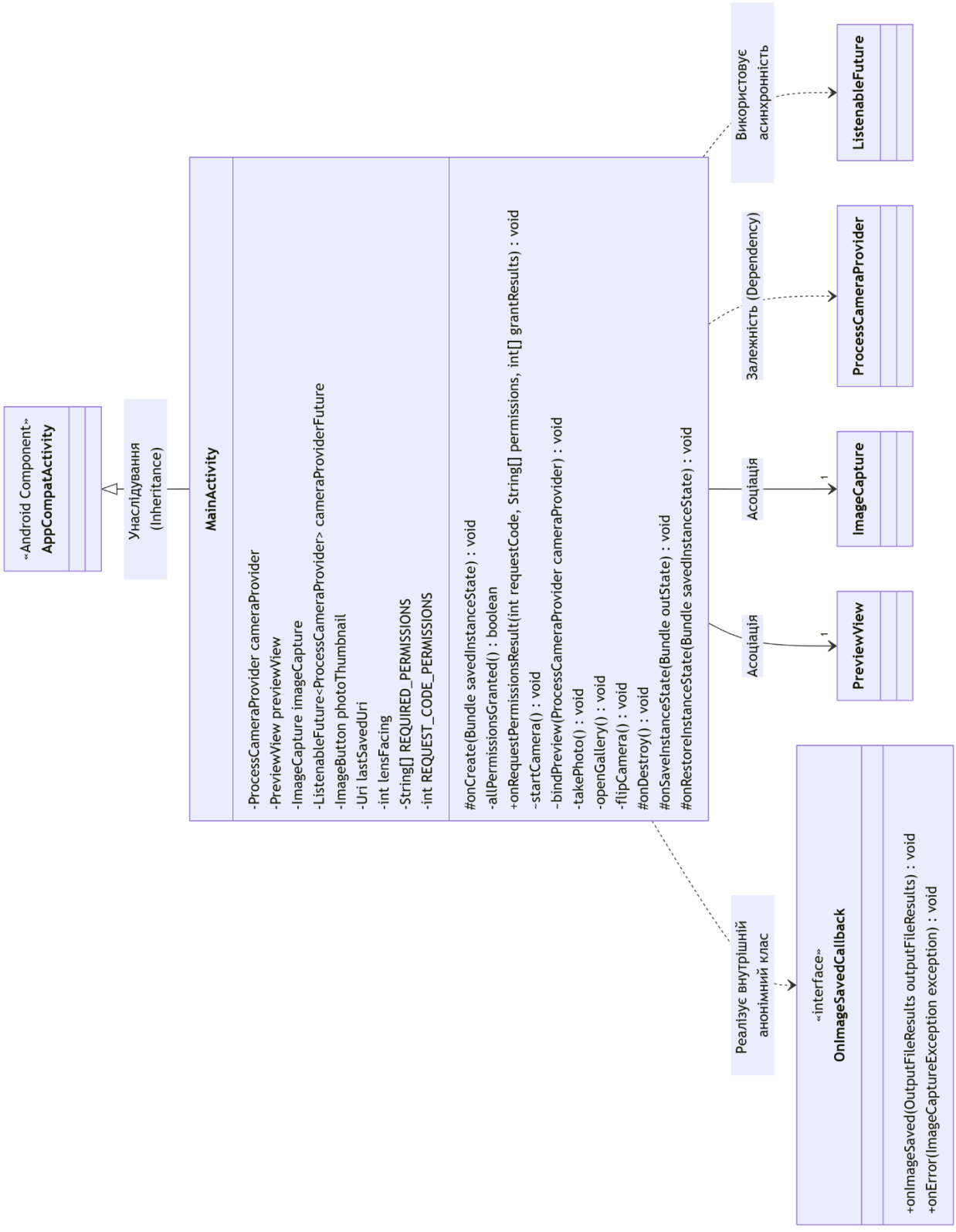
Таким чином, створений програмний продукт успішно вирішує актуальну інженерну задачу швидкої та ресурсоефективної фотофіксації в мобільних системах. Завдяки збалансованому розподілу обчислень між фоновими потоками та апаратною частиною смартфона, застосунок демонструє високу відмовостійкість і стабільність рендерингу. Наявність у структурі ізольованого шару аналізу кадрів робить розроблену архітектуру перспективним і легко масштабованим базисом для подальшої інтеграції засобів комп'ютерного зору або модулів інтелектуальної обробки зображень у реальному часі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

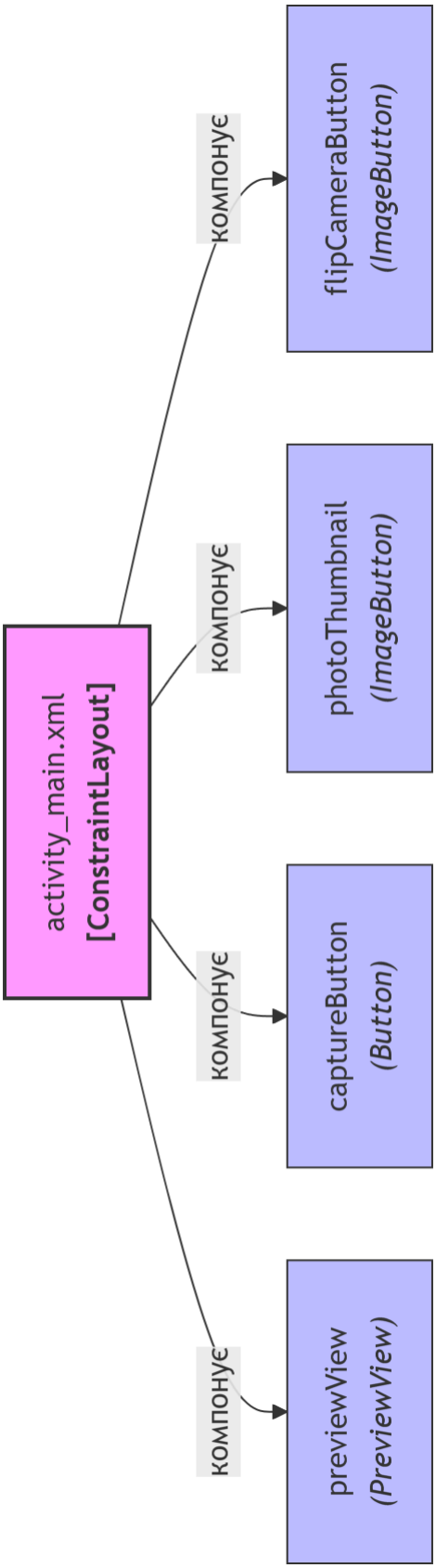
1. Tanenbaum A., Bos H. Modern Operating Systems. 4th Ed. Amsterdam : Vrije Universiteit, 2015. 1101 p.
2. Tanenbaum A., Woodhull A. Operating Systems: Design and Implementation. 3th Ed. Amsterdam : Vrije Universiteit, 2006. 1080 p.
3. Roger Y. Android System Programming. Packt Publishing Ltd, 2017. 470 p.
4. Burton M. Android App Development for Dummies. Third edition. Wiley, 2015. 440 p.
5. Phillips B. Android Programming: The Big Nerd Ranch Guide. 4th edition / Phillips B., Stewart C., Marsicano K. / Big Nerd Ranch Guides, 2019. 624 p.
6. Boyer R., Mew K. Android Application Development Cookbook. Second Edition. Packt Publishing, 2016. 428 p.
7. Darwin I. Android Cookbook: Problems and Solutions for Android Developers, Second Edition (Grayscale Indian Edition). Shroff/O'Reilly, 2017. 768 p.
8. CameraX devices. URL: https://developer.android.com/media/camera/camerax/devices (дата звернення: 10.03.2026).
9. Smyth N. Jetpack Compose 1.5 Essentials: Developing Android Apps with Jetpack Compose 1.5, Android Studio, and Kotlin. Payload Media. 636 p.
10. Makan K. Android Security Cookbook. Packt Publishing Limited, 2013. 350 p.
11. Stallings William. Operating Systems: Internals and Design Principles, 9th Ed. Pearson, 2018.
12. Friesen J. Learn Java for Android Development. 3rd ed. Apress, 2014. 1190 p.
13. Галагуз Т. О., Ляшук Т. Г. Архітектура та системні механізми керування камерою в Android. Сучасні інформаційні технології: від теорії до практики (СІнТех-2026) : тези доп. І Міжнар. наук.-практ. конф. (м. Луцьк, 22–23 травня 2026 р.). Луцьк : ЛНТУ, 2026. С. 382. URL: https://mintech-conf.lntu.edu.ua/f/MInTech2026-theses.pdf.

ДОДАТКИ

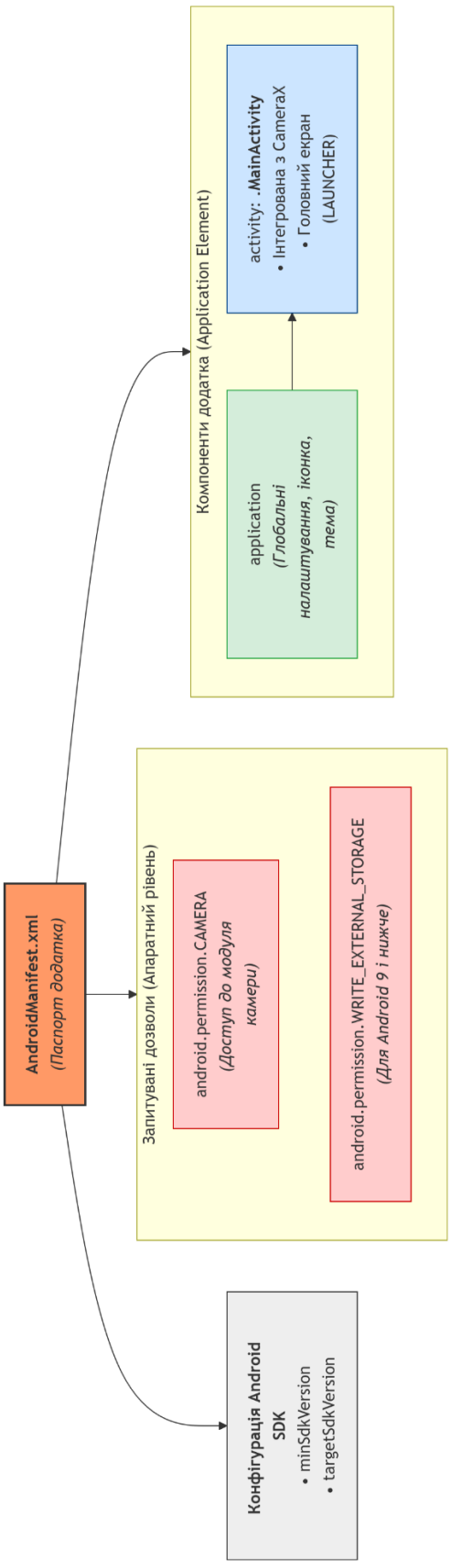
Додаток А. UML-діаграма класів



Додаток Б. Діаграма UI-компонентів



Додаток В. Структурна схема конфігураційного файла



Додаток Г. Структурна схема програмних залежностей та конфігурації проєкту