Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного забезпечення для генерування та відновлення QR-кодів з використанням кодів Ріда-Соломона»

Виконала: студентка IV курсу
групи 4ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Сидорук А. О.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Романюк О. В.
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф ЗІ Куперштейн Л. М.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ
д. т. н. проф. Романюк О. В.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

УЗГОДЖЕНО
Директор ТОВ "Електрон"
_____
« 21 » березня 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____О. Н. Романюк
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Сидорук Анні Олександрівні

1. Тема роботи – Розробка програмного забезпечення для генерування та відновлення QR-кодів з використанням кодів Ріда- Соломона.
Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.
2. Строк подання студентом роботи 2 червня 2025.
3. Вихідні дані до роботи: дії застосунку – генерація QR-кодів із різними типами даних; сканування QR-кодів через камеру або завантаження зображень; відновлення пошкоджених QR-кодів із фрагментів; додавання цифрового підпису для захисту даних; текстові повідомлення – підтвердження виконання операцій, повідомлення про помилки або статус обробки; інтерактивні елементи – перегляд попереднього зображення QR-коду, кнопки збереження та очищення фрагментів; обробка команд – через Android API; мультимедійна обробка – аналіз зображень із використанням ML Kit; кодування та корекція помилок за модифікованим алгоритмом Ріда-Соломона; контроль доступу – перевірка дозволів на використання камери; кінцевий результат – ефективне створення, сканування та відновлення захищених QR-кодів у мобільному застосунку.
4. Зміст текстової частини: вступ; аналіз стану питання та обґрунтування завдання; розробка методів та алгоритмів роботи мобільного застосунку; розробка програмного забезпечення мобільного застосунку для генерування та відновлення QR-кодів; тестування програмного забезпечення; висновки; список використаних джерел; додатки.
5. Перелік ілюстративної частини: титульний слайд; актуальність теми; мета, об'єкт дослідження та предмет дослідження; постановка задач дослідження; наукова новизна отриманих результатів; порівняльний аналіз аналогів; діаграма прецедентів архітектури застосунку; діаграма класів моделі системи; метод та

діаграма послідовності модифікованих кодів Ріда-Соломона, метод та блок-схема алгоритму відновлення QR-коду з фрагментів; порівняльний аналіз мов програмування; порівняльний аналіз середовищ розробки; діаграма послідовності і протип модуля генерування QR-коду; діаграма послідовності модуля генерування QR-коду, функціональне тестування мобільного застосунку, тестування продуктивності мобільного застосунку, апробація, публікація та впровадження матеріалів бакалаврської кваліфікаційної роботи; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Романюк О. В., к.т.н., доц. каф. ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та обгрунтування завдання	25.03.25 – 31.03.2025	виконано
2	Розробка архітектури мобільного застосунку	31.03.25 – 05.04.2025	виконано
3	Розробка методу електронного цифрового підпису	05.03.2025 – 07.04.2025	виконано
4	Модифікація кодів Ріда-Соломона	07.04.2025 – 12.04.2025	виконано
5	Розробка методу відновлення QR-кодів	12.04.2025 – 23.04.2025	виконано
6	Розробка програмного забезпечення мобільного застосунку для генерування та відновлення QR-кодів	23.04.2025 – 01.05.2025	виконано
7	Тестування програмного забезпечення	01.05.2025 – 10.05.2025	виконано
8	Оформлення матеріалів до захисту БКР	10.05.2025 – 30.05.25	виконано

Студент _____ **А. О. Сидорук**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **О. В. Романюк**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.93:003.26

Сидорук А. О. Розробка програмного забезпечення для генерування та відновлення QR-кодів з використанням кодів Ріда-Соломона: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025 – 66 с.

На укр. мові. Бібліогр.: 37 назв; рис.: 43; табл.: 6.

У бакалаврській кваліфікаційній роботі обґрунтовано актуальність, практичну цінність та мету розробки мобільного застосунку для роботи з QR-кодами, що забезпечує генерацію, зчитування та відновлення з підвищеною безпекою та стійкістю до пошкоджень.

Розроблено методи та алгоритми реалізації основних функцій застосунку, зокрема архітектуру з модулями генерації, зчитування та відновлення, удосконалений алгоритм Ріда-Соломона з полями Галуа 2^{16} для відновлення кодів до 40% пошкоджень, а також цифровий підпис на основі SHA256withRSA. Застосунок реалізовано мовою Kotlin у середовищі Android Studio з бібліотеками Android SDK, ML Kit та криптографічними модулями. Проведено тестування, підготовлено інструкцію користувача та визначено системні вимоги.

Результати розробки застосунку для роботи з QR-кодами можуть бути застосовані в логістиці, цифровій ідентифікації та електронному документообігу.

Ключові слова: QR-коди, цифровий підпис, коди Ріда-Соломона, квішинг, мобільний застосунок, Kotlin.

ABSTRACT

UDC 004.93:003.26

Sydoruk A. O. Development of Software for Generating and Recovering QR Codes Using Reed-Solomon Codes: Bachelor's Qualification Work in the Specialty 121 – Software Engineering, Educational Program – Software Engineering. Vinnytsia: VNTU, 2025 – 66 p.

In English. References: 37 titles; fig.: 21; tables: 6.

The bachelor's qualification work justifies the relevance, practical value, and purpose of developing a mobile application for working with QR codes, providing generation, reading, and recovery with enhanced security and damage resistance.

Methods and algorithms for implementing the application's core functions were developed, including an architecture with modules for generation, reading, and recovery, an improved Reed-Solomon algorithm with Galois fields 2^{16} for recovering codes with up to 40% damage, and a digital signature based on SHA256withRSA. The application was implemented in Kotlin using Android Studio with Android SDK, ML Kit, and cryptographic libraries. Testing was conducted, a user manual was prepared, and system requirements were defined.

The results of the QR code application development can be applied in logistics, digital identification, and electronic document management.

Keywords: QR codes, digital signature, Reed-Solomon codes, quishing, mobile application, Kotlin.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі де швидкий доступ до інформації це не просто можливість, а вітальна необхідність. Один з найпоширеніших способів для негайної передачі інформації це QR-коди. Якщо просто озирнутись то вони оточують нас повсюди, QR-коди для оплати починаючи від води в автоматах і проїзду в громадському транспорті, донатами для військових зборів у відео, сканування меню чи написання відгуків в кафе та ресторанах, а також для входу на різноманітні заходи, реєстрації в чергах, перегляду додаткової інформації на продуктах і навіть у вуличній рекламі.

Бренди здебільшого застосовують QR-коди для поєднання офлайн- і онлайн-взаємодії з користувачами. Це дає змогу перенаправити людей на їх сайти чи сторінки в соціальних мережах, а також ділитися різноманітними документами та іншими матеріалами, використовувати QR-коди як візитівки передаючи контактну інформацію тощо [1].

Відповідно до нещодавного дослідження опублікованого на сайті TEAM LEWIS щодо споживчого досвіду використання QR-кодів показало, що приблизно 68% споживачів у США використовували QR-коди принаймні один раз протягом останнього року. Якщо аналізувати у вікових групах то 51% міленіалів і 49% споживачів покоління Z використовують ці коди принаймні раз на тиждень. Більше того, 29% споживачів висловили занепокоєння щодо конфіденційності та безпеки своїх даних, сказавши, що вони віддадуть перевагу використанню безпечних і надійних QR-кодів [2].

Беручи до уваги широке розповсюдження QR-кодів у повсякденному житті людства виникає декілька пов’язаних з цим питань. По-перше, існує питання як якісно відновлювати сильно пошкодженні QR-коди, якщо є нагальна необхідність у терміновому використанні певного конкретного екземпляру і немає можливості знайти непошкоджений. По-друге, як при використанні відновленого екземпляра так і під час сканування оригінального, можна легко

зіткнутись з таким видом фішингової атаки, як Квішинг. QR-фішинг, також відомий як квішинг, є видом кіберзлочинності, що використовує популярність QR-кодів для обману користувачів. У цьому шахрайстві кіберзлочинці створюють шкідливі QR-коди, які при скануванні перенаправляють користувачів на шахрайські веб-сайти або автоматично завантажують шкідливе програмне забезпечення. Такий вид шахрайства використовує природну довіру людей до QR-кодів і їх зручність, оскільки користувачі не бачать вмісту за QR-кодом до його сканування. Це робить квішинг небезпечним інструментом для викрадення особистої інформації та поширення шкідливих програм [3].

Таким чином, традиційні методи відновлення і забезпечення безпеки QR-кодів не є достатньо ефективними і не можуть задовольнити потреби користувачів в достатній мірі, тому виникає потреба модифікувати існуючий алгоритм кодів Ріда-Соломона та покращити безпеку створюваних і зчитуваних QR-кодів за допомогою цифрового підпису, що зумовлює актуальність розробки.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою даної роботи є підвищення ефективності використання QR-кодів шляхом удосконалення методів їх відновлення після пошкодження, а також покращення рівня безпеки користувачів через виявлення та мінімізацію загроз фішингових атак типу квішинг.

Відповідно до поставленої мети потрібно вирішити такі **завдання:**

- провести аналіз стану предметної області та аналогічних застосунків для генерації QR-кодів із підвищеним рівнем безпеки та стійкості до пошкоджень;

- розробити метод та алгоритм цифрового підпису QR-кодів з безпечним зчитуванням;

- розробити метод та алгоритм відновлення пошкоджених QR-кодів на основі кодів Ріда-Соломона, що забезпечує можливість відновлення коду з кількох пошкоджених фрагментів;

- виконати програмну реалізацію генерації та зчитування QR-кодів із цифровим підписом, інтегрувавши модифікований алгоритм Ріда-Соломона для підвищення відмовостійкості;

- протестувати функціональність системи та оцінити її ефективність за критеріями цілісності даних, захищеності від фішингових атак і відновлення з пошкоджень.

Об’єктом дослідження є процес генерування, відновлення, зчитування та захисту QR-кодів у системах передачі інформації.

Предметом дослідження є методи і засоби генерування та відновлення QR-кодів з цифровим підписом, їх безпечного зчитування, а також відновлення з пошкоджених фрагментів із використанням модифікованих алгоритмів, зокрема кодування Ріда-Соломона.

Методи дослідження. У процесі дослідження застосовувалися: теорія алгоритмів для розробки основних алгоритмів роботи застосунку; теорія об’єктно-орієнтованого програмування для розробки мобільного застосунку на платформі Android; теорія кодування та декодування інформації для перетворення даних, що вбудовуються в QR-код; методи тестування програмного забезпечення для перевірки працездатності програмних модулів системи; комп’ютерне моделювання і логування для аналізу внутрішніх процесів роботи застосунку та виявлення помилок; методи оцінки надійності та захищеності QR-кодів для моделювання атак типу квішинг і перевірки стійкості до підміни або модифікації вмісту коду.

Новизна отриманих результатів.

1. Удосконалено метод цифрового підпису QR-кодів, який на відміну від відомих дозволяє здійснювати перевірку їх автентичності та забезпечує захист від фішингових атак, зокрема квішингу, що дозволило підвищити рівень довіри до джерела інформації та гарантувати цілісність даних у процесі зчитування

QR-коду.

2. Удосконалено метод відновлення пошкоджених QR-кодів на основі кодів Ріда-Соломона, у якому, на відміну від класичних підходів, використано поля Галуа розмірності 2^{16}, що дозволило значно підвищити точність та ефективність відновлення даних у QR-кодах навіть за умов значних пошкоджень.

3. Запропоновано нову архітектуру мобільного застосунку, побудовану на базі Android із використанням мови Kotlin, особливістю якої є охоплення повного циклу роботи з QR-кодами – від генерації та підпису до сканування, перевірки й відновлення, що дало можливість що дало можливість створити єдину безпечну екосистему для роботи з QR-кодами на мобільному пристрої.

Практична цінність отриманих результатів. Практична цінність отриманих результатів полягає в тому, що на основі розроблених у бакалаврській кваліфікаційній роботі теоретичних рішень створено алгоритми та програмні засоби для генерації та відновлення QR-кодів із цифровим підписом та використанням удосконалених кодів Ріда-Соломона, безпечного зчитування, виявлення підробок і відновлення пошкоджених кодів із фрагментів. Запропоноване рішення може бути застосоване у сферах, що вимагають високої достовірності та захисту інформації: медицині, логістиці, електронному документообігу, цифровій ідентифікації тощо.

Впровадження. Впровадження результатів досліджень підтверджується відповідним актом та використовується в такій організації:

– Техцентр «Електрон» для підвищення надійності обробки та перевірки автентичності даних у системі електронного документообігу.

Особистий внесок здобувача. Усі наукові результати отримано здобувачем самостійно. У працях, опублікованих у співавторстві, студенту належать: розробка концепції застосування Midjourney для створення дизайн-макетів мобільних інтерфейсів [4]; удосконалення алгоритму корекції QR-кодів шляхом модифікації кодів Ріда–Соломона для роботи з полем GF 2^{16}, що підвищує їх надійність та опірність до пошкоджень [5]; проєктування та

впровадження методу перевірки автентичності QR-кодів із використанням електронного цифрового підпису за алгоритмом SHA256withRSA [6].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Описані у дослідженні положення доповідались на таких конференціях:

− IV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації - 2024» (Одеса, 2024);

− LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025);

− Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікації. За тематикою дослідження опубліковано 3 наукові праці у збірниках матеріалів конференцій.

1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ

1.1 Аналіз стану мобільних застосунків для генерування і відновлення QR-кодів

У 2025 році QR-коди залишаються одним з найпоширеніших засобів обміну інформацією, автентифікації, оплати та маркетингу. За даними ресурсу QRCodeChimp, близько 44,6% користувачів Інтернету віком від 16 до 64 років щомісяця сканують принаймні один QR-код, а загальне використання таких кодів щороку зростає на 22%. Серед найпопулярніших мобільних застосунків для роботи з QR-кодами варто згадати TrendMicro Secure QR Reader, QR & Barcode Reader by Gamma Play та QR Code Reader від TeaCapps. Вони дозволяють користувачам швидко зчитувати та створювати коди, однак, як правило, не містять функцій відновлення пошкоджених кодів або перевірки їх автентичності [7].

Поточний стан застосунків виявляє кілька суттєвих недоліків. По-перше, більшість мобільних додатків не здатні відновити частково пошкоджені або спотворені QR-коди, що може бути критичним у випадках, коли немає доступу до оригіналу. По-друге, відсутній надійний захист від фішингових атак – квішингу, які полягають у тому, що зловмисники генерують шкідливі коди, здатні перенаправити користувача на небезпечні сайти або встановити шкідливе програмне забезпечення. По-третє, QR-коди, які використовуються сьогодні, рідко містять механізми перевірки автентичності. Також існує ризик встановлення шкідливих застосунків із магазинів додатків, які маскуються під звичайні сканери.

Причини виникнення цих труднощів багатогранні. По-перше, QR-коди за своєю природою не повинні мати складної логіки перевірки безпеки або відновлення. По-друге, користувачі часто не усвідомлюють потенційних ризиків, пов’язаних із QR-кодами, і не вживають заходів безпеки. По-третє, на ринку мобільного програмного забезпечення немає уніфікованих стандартів

безпечної роботи з QR-кодами. Нарешті, простота створення коду робить його зручним інструментом для зловмисників.

Зазначені проблеми мають суттєве значення. По-перше, збільшення використання QR-кодів прямо пропорційно збільшує вразливість користувачів до атак. По-друге, несанкціоноване перенаправлення або витік особистих даних може призвести до фінансових збитків та порушення конфіденційності. По-третє, широке поширення небезпечних або нефункціональних QR-кодів може знизити довіру до цієї технології як такої.

Отже, існує гостра потреба у створенні застосунків, що забезпечують не лише базову генерацію та зчитування кодів, а й підтримують цифровий підпис, відновлення з пошкоджених частин та вбудовані механізми перевірки автентичності.

1.2 Порівняльний аналіз застосунків для створення QR-кодів

Для визначення конкурентних переваг мобільного застосунку для генерації та відновлення QR-кодів, було розглянуто три популярні застосунки: QR Code Generator & Scanner (QRX), QR TIGER, та QRazyBox. Порівняльний аналіз функціоналу кожного із цих застосунків дозволить з’ясувати їхні слабкі та сильні сторони, а також визначити, які саме функції є найбільш корисними для користувачів. Це допоможе створити базу для подальшого вдосконалення та розвитку мобільних застосунків для управління QR-кодами, забезпечуючи їм зручний та ефективний інструмент для створення, зчитування та відновлення кодів.

QRX – це мобільний застосунок, який поєднує в собі базові функції генерації та сканування QR-кодів, орієнтуючись на простоту використання і швидкість доступу до інформації. Він дозволяє створювати QR-коди для різноманітних типів даних: контактів, текстових повідомлень, електронних листів, посилань на веб-сайти тощо. Однією з корисних функцій є можливість сканування кодів не лише за допомогою камери, але й безпосередньо зображень, які вже збережені в галереї користувача. Інтерфейс додатку

інтуїтивно зрозумілий, а швидкість обробки QR-кодів робить його зручним у повсякденному використанні.

Попри широкий набір базових можливостей, QRX має і певні обмеження. Він не підтримує функцію цифрового підпису, що знижує рівень безпеки створених кодів у контексті захисту від підробки чи фішингових атак. Також відсутній механізм відновлення пошкоджених або частково втрачених QR-кодів, що обмежує застосунок у критичних умовах, де потрібна висока надійність і відмовостійкість. Таким чином, QRX підходить здебільшого для побутового або неформального використання, але не є оптимальним рішенням для задач, де потрібна підвищена безпека і робота з пошкодженими кодами [8] (див. рис. 1.1).

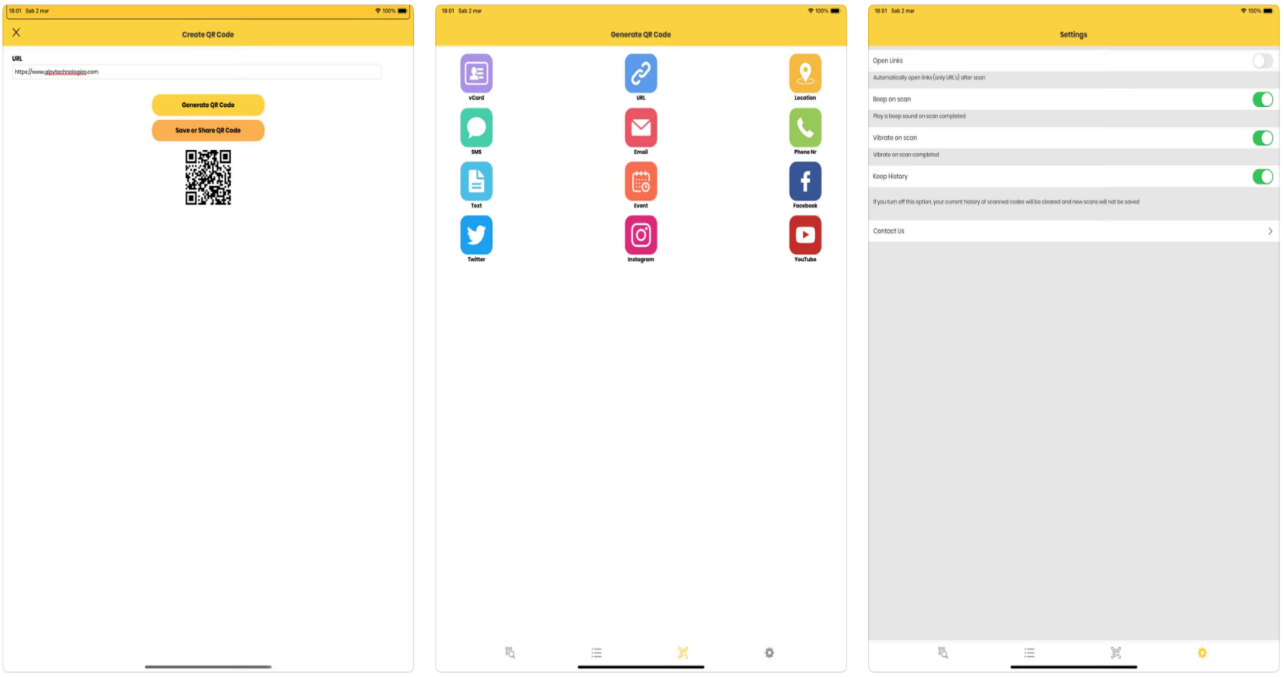


Рисунок 1.1 – Застосунок «QRX»

QR TIGER – це популярний онлайн-сервіс та мобільний додаток для створення динамічних і статичних QR-кодів з розширеним функціоналом. Застосунок підтримує генерацію кодів для широкого спектра цілей: URL-посилань, Wi-Fi, візиток, PDF-файлів, подій, соціальних мереж тощо. Однією з головних переваг є можливість створення динамічних QR-кодів, які можна редагувати навіть після їх друку, а також відстеження сканувань, що

особливо цінно для маркетингових кампаній. Додаток також дозволяє брендувати QR-коди – змінювати кольори, вставляти логотипи, налаштовувати рамки, що покращує візуальну привабливість [9].

Однак QR TIGER, попри потужний набір інструментів для комерційного та промислового використання, не має вбудованої підтримки цифрових підписів, що обмежує його придатність для задач, пов’язаних із захистом від підробки або підтвердження автентичності коду. Крім того, застосунок не надає функцій для відновлення пошкоджених QR-кодів, що знижує його стійкість у ситуаціях, коли дані зображення частково втрачені або зазнали механічних пошкоджень. Таким чином, хоча QR TIGER добре підходить для бізнесу та маркетингу, він не забезпечує необхідного рівня надійності й безпеки для критичних чи чутливих застосувань (див. рис. 1.2).

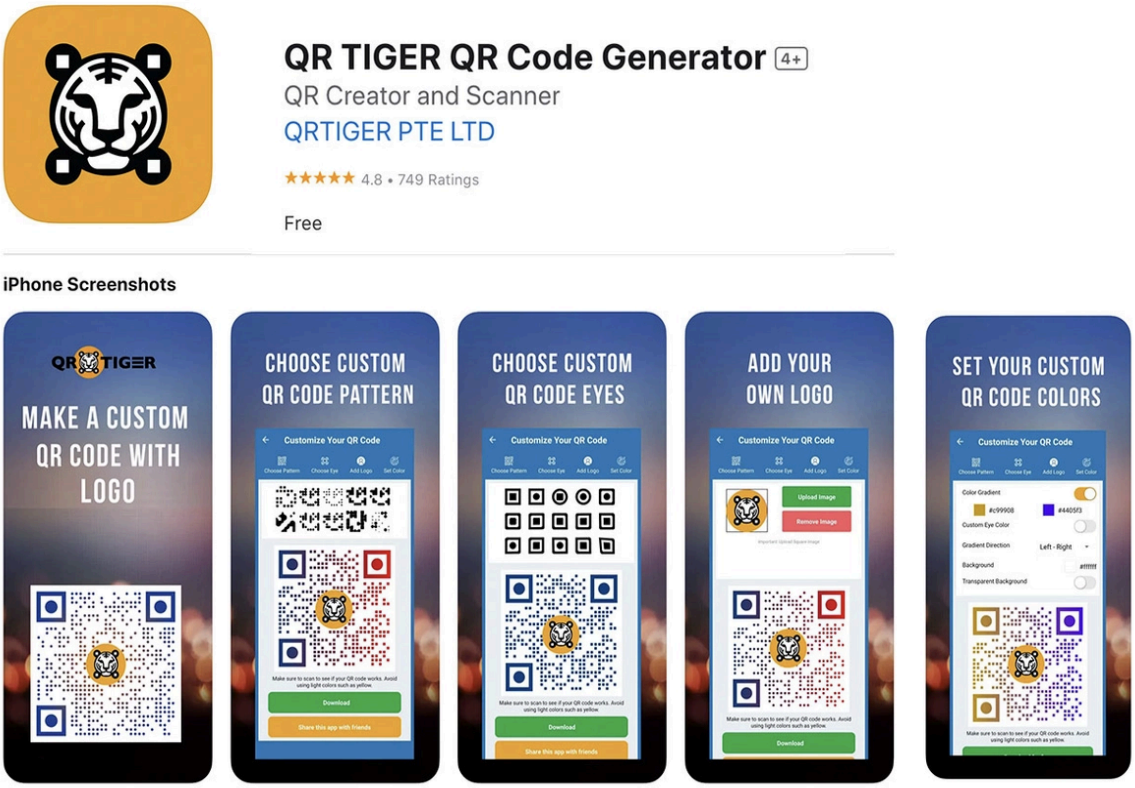


Рисунок 1.2 – Застосунок «QR TIGER»

QRazyBox – це функціональний веб-сервіс, орієнтований на створення та тестування QR-кодів з глибшими можливостями налаштування. Він дозволяє не лише згенерувати код із заданим текстом або посиланням, а й експериментувати з різними параметрами корекції помилок, розміром модуля, типом QR-коду тощо. Також QRazyBox дає змогу проводити аналіз вмісту коду і вручну змінювати внутрішню структуру QR, що особливо корисно для дослідницьких або навчальних цілей. Інтерфейс сервісу більше орієнтований на технічних користувачів або розробників [10].

Проте, попри гнучкість у налаштуваннях, QRazyBox не передбачає вбудовану підтримку цифрового підпису або шифрування інформації в коді, що робить його менш придатним для сценаріїв, де важлива перевірка автентичності. Також у сервісі відсутня можливість автоматичного відновлення пошкоджених QR-кодів з фрагментів або частин, що обмежує його застосування у випадках, коли цілісність коду має критичне значення. Загалом QRazyBox є зручним інструментом для тестування й розробки, але не забезпечує достатнього рівня безпеки для захищеної передачі даних (див. рис. 1.3).

Рисунок 1.3 – Веб-сервіс «QRazyBox»

У результаті аналізу аналогів було створено таблицю 1.1 для порівняння їхнього функціоналу з мобільним застосунком «Advanced QR Scanner», що було розроблено під час виконання бакалаврської кваліфікаційної роботи. Порівняння підкреслює унікальність «Advanced QR Scanner», який поєднує комплексний набір функцій, що відсутні в аналогах, таких як QRX, QR TIGER та QRazyBox, забезпечуючи вищий рівень безпеки та надійності для користувачів.

Таблиця 1.1 – Функціональні характеристики застосунків

Критерій	QRX	QR TIGER	QRazyBox	Advanced QR Scanner
Підтримка цифрових даних із підписом	-	-	-	+
Можливість сканувати зображення з галереї	+	+	-	+
Відновлення пошкоджених QR-кодів	-	-	+	+
Захист від QR-фішингу	-	+	-	+
Можливість попереднього перегляду вмісту	+	+	-	+
Підтримка роботи з кількома фрагментами коду	-	-	-	+
Сумарний коефіцієнт	2	3	1	6

Аналізуючи дані таблиці 1.1, можна зробити висновок, що мобільний застосунок Advanced QR Scanner демонструє найвищий рівень функціональної насиченості – він підтримує 6 із 6 розглянутих ключових функцій. Для порівняння, QR TIGER реалізує лише 3 функції, QRX – 2 функції, а QRazyBox – лише 1. Таким чином, власна розробка перевершує найближчого конкурента на 50%, що підтверджує її технологічну перевагу та більшу практичну цінність для

користувача. Це свідчить про інноваційність підходу, спрямованого на безпеку, гнучкість та стійкість до пошкоджень QR-кодів.

1.3 Аналіз методів розв'язання задачі

Коди Ріда-Соломона є важливим класом блокових лінійних кодів, що використовуються для виявлення та виправлення помилок у цифрових даних. Вони вперше були представлені у 1960 році Ірвінгом Рідом та Густавом Соломоном і з того часу знайшли широке застосування у супутниковому зв'язку, зберіганні даних, компакт-дисках, а також у QR-кодах. Коди Ріда-Соломона є підкласом кодів Боуза-Чоудхурі-Хоквінгема, зокрема їхнім багатовимірним узагальненням, і будуються над скінченними полями – полями Галуа.

Коди Ріда-Соломона використовують інтерполяційний поліном для кодування повідомлення, представленого у вигляді коефіцієнтів, які обчислюються в точках скінченного поля Галуа F_q, де $q=2^m$. Для повідомлення з k символів будується поліном $P(x)$ ступеня не вище $\boldsymbol{k-1}$, який оцінюється в n різних точках поля $(n \leq q - 1)$, формуючи код довжиною n. Такий код позначається як $RS(n, k)$.

Кодове слово задається вектором:

$$c = (P(\alpha 1), P(\alpha 2), \ldots, P(\alpha n)), \# \qquad (1.1)$$

де c – кодове слово, P(x) – інтерполяційний поліном, α_i – точки поля Галуа, $i = 1, 2, \ldots, n$.

Ключова властивість кодів Ріда-Соломона полягає в їх здатності виправляти до t t t помилок у кодовому слові, де кількість виправних помилок визначається формулою: $t = [(n - k) \div 2]$,

де n – довжина коду, k – кількість інформаційних символів. Ця властивість забезпечує ефективне виправлення пакетних помилок, коли пошкоджуються великі групи бітів, що йдуть підряд.

Процес декодування реалізується за допомогою алгоритмів, таких як алгоритм Берхлекемпа-Массі або євклідовий алгоритм, які знаходять поліном помилки *e(x)* та його позиції. Поліном помилки задовольняє рівняння:

$$S(x) = e(x) \cdot \Lambda(x) mod x^{2t}, \# \quad (1.2)$$

де *S(x)* – синдромовий поліном, $\Lambda(x)$ – локатор помилок, $e(x)$ – величини помилок, t – максимальна кількість виправних помилок. Ця формула дозволяє точно визначити позиції та значення помилок у кодовому слові.

Коди Ріда-Соломона будуються над розширеними скінченними полями, зокрема над полями виду $F2^m$, тобто такими, які мають 2^m елементів. Такі поля дають можливість зберігати не лише біти, а й цілі байти, наприклад, при $m = 8$, або навіть 16-бітові слова при $m = 16$. Використання більших полів, як-от $F2^{16}$, дозволяє значно підвищити надійність кодування, що є особливо важливим для систем, де дані зберігаються або передаються з високим ризиком пошкоджень.

Одним із найвідоміших застосувань кодів Ріда-Соломона стала міжпланетна програма «Вояджер», яка передавала зображення та наукові дані із зовнішніх планет Сонячної системи. Оскільки сигнал від зондів був дуже слабким і спотвореним космічним шумом, коди Ріда-Соломона забезпечили надзвичайно ефективне виправлення помилок, дозволивши зберегти цінну наукову інформацію після багаторазового стирання, шуму й перешкод.

У сучасних технологіях коди Ріда-Соломона широко використовуються в QR-кодах, де вони виконують функцію виправлення помилок при скануванні. У стандартному QR-коді застосовуються коди $RS(255, k)$ над полем $F2^8$, що дозволяє коригувати до 30% пошкодженої інформації в залежності від рівня Error Correction Capacity. Наприклад, для рівня відновлення «H» QR-код може відновити до 30% втрачених або пошкоджених символів. Алгоритм декодування у QR-кодах виконується апаратно або програмно, зазвичай із застосуванням методу Ченя-Лі або Берхлекемпа [11].

1.4 Постановка задач для розробки мобільного застосунку для генерування і відновлення QR-кодів

З огляду на аналіз функціональних можливостей застосунків для генерації та відновлення QR-кодів, були визначені такі ключові завдання:

- провести аналіз стану предметної області та аналогічних застосунків для генерації QR-кодів із підвищеним рівнем безпеки та стійкості до пошкоджень;
- розробити метод та алгоритм цифрового підпису QR-кодів з безпечним зчитуванням;
- розробити метод та алгоритм відновлення пошкоджених QR-кодів на основі кодів Ріда-Соломона, що забезпечує можливість відновлення коду з кількох пошкоджених фрагментів;
- виконати програмну реалізацію генерації та зчитування QR-кодів із цифровим підписом, інтегрувавши модифікований алгоритм Ріда-Соломона для підвищення відмовостійкості;
- протестувати функціональність системи та оцінити її ефективність за критеріями цілісності даних, захищеності від фішингових атак і відновлення з пошкоджень.

Ці завдання спрямовані на розробку функціоналу для ефективного створення, відновлення та безпечного використання QR-кодів, з урахуванням потреб користувачів та вимог до безпеки.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі дослідження був здійснений аналіз сучасного стану мобільних застосунків для генерування та відновлення QR-кодів, з акцентом на такі платформи як QRX, QR TIGER та QRazyBox. Вивчення цих систем показало, що хоча вони мають певні переваги, такі як підтримка динамічних QR-кодів та зручність використання, існують суттєві недоліки. Однією з основних проблем є відсутність підтримки цифрового підпису, неможливість

відновлення пошкоджених кодів та недостатній захист від фішингових атак. Це підкреслює важливість створення мобільного застосунку, який би забезпечував високий рівень безпеки, надійності та зручності для користувачів.

На основі цього аналізу було визначено конкретні завдання, які необхідно реалізувати для успішного запуску нового продукту. Серед них – розробка мобільного застосунку, що підтримує цифровий підпис, має можливість відновлення пошкоджених кодів і захист від фішингових атак. Особлива увага приділяється інтеграції апаратних можливостей пристроїв для забезпечення стабільної та ефективної роботи. Також планується ретельне тестування системи, щоб забезпечити її стабільність, безпеку та користувацьку зручність у реальних умовах. Метою даної роботи є підвищення ефективності використання QR-кодів шляхом удосконалення методів їх відновлення після пошкодження, а також покращення рівня безпеки користувачів через виявлення та мінімізацію загроз фішингових атак.

Результати аналізу та порівняння з існуючими системами підтверджують, що розробка мобільного застосунку з підтримкою цифрового підпису, відновлення пошкоджених QR-кодів та захистом від фішингових атак може значно покращити безпеку та ефективність роботи з QR-кодами. Це дозволить забезпечити високий рівень захисту для користувачів і стати важливим кроком до вдосконалення технологій роботи з QR-кодами.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ РОБОТИ МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Розробка архітектури

Архітектура мобільного застосунку – це низка рішень та дій, що дозволяють організувати коректну роботу програми. До неї входять як інтерфейси, так і структурні елементи, бази даних, стилі та інші компоненти [12].

Під час розробки програми було вирішено створити структурно нову архітектуру, яка могла б вмістити весь функціонал та сприяла кращій роботі і легшому розумінню даного мобільного застосунку. Для опису архітектури створюваного програмного забезпечення було обрано діаграму прецендентів. Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі, вона показує різні варіанти використання та різні типи користувачів системи і часто супроводжується іншими типами діаграм. Варіанти використання представлені колами або еліпсами. Актори часто зображуються у вигляді паличок [13].

Оскільки саме даний вид діаграми дає змогу якісно визначити взаємозв’язки між користувачами та основними функціональними можливостями мобільного додатку. Також це може спростити розуміння архітектури оскільки, діаграма прецендентів допомже ідентифікувати основні сценарії розподілити відповідальність між компонентами та забезпечити узгодженість між функціональними вимогами для покращення розуміння наступних кроків до їх подальшої реалізації.

На основі розробленої архітектури була створена діаграма прецедентів, яка відображає ключові взаємодії між акторами та функціональними можливостями системи. У центрі діаграми знаходяться два основні актори: «Користувач» та «Система-сканування» (див. рис. 2.1). Актор «Користувач представляє людину, яка взаємодіє з мобільним застосунком, виконуючи дії, такі як вибір фрагментів QR-коду чи перегляд результатів. Актор

«Система-сканування» уособлює автоматичні процеси, пов’язані зі скануванням і відновленням QR-кодів.

Рисунок 2.1 – Діаграма прецедентів

Основний прецедент «Відновити QR-код» є центральним у системі, саме він вказує на одну з цілей застосунку – об’єднання фрагментів для отримання повного QR-коду. Цей прецедент розширюється прецедентом «Керувати клочами», який включає вибір і обробку фрагментів зображень користувачем. Також прецедент «Відновити QR-код» включає допоміжний прецедент «Додати цифровий підпис», що забезпечить додатковий захист даних, а також «Перевірити цифровий підпис», який гарантує достовірність відновленого QR-коду. Окремо виділено прецедент «Сканувати QR-код», який виконується системою-скануванням для аналізу зображень.

Ось тому можна чітко розмежувати ролі акторів і прецеденти, що дає змогу зрозуміти архітектуру мобільного застосунку і як він працюватиме надалі.

2.2 Розробка моделі системи

Модель системи – це представлення реальної системи програмного застосунку в деякій формі, відмінній від форми їх реального існування, це абстракція, яка допомагає зрозуміти складні процеси та спрощує розробку

самого застосунку [14]. Для створення моделі системи, що могла б дати уявлення про те які об’єкти будуть присутні в системі їх взаємодію, та як будуть оброблятися дані в системі було обрано діаграму класів. Даний вид діаграми відображає статичні елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів може містити позначення для пакетів та може містити позначення для вкладених пакетів [15]. Вона дозволяє заздалегідь визначити відповідальності кожного класу, оптимізувати структуру даних і уникнути надлишковості в коді. Завдяки цьому розробники можуть ефективно організувати взаємодію між класами, для операцій у скінченному полі для обробки QR-кодів, створюючи надійну основу для подальшої реалізації системи. На рисунку 2.2 представлено розроблену діаграму класів для наочного представлення функціонування програмних модулів мобільного застосунку.

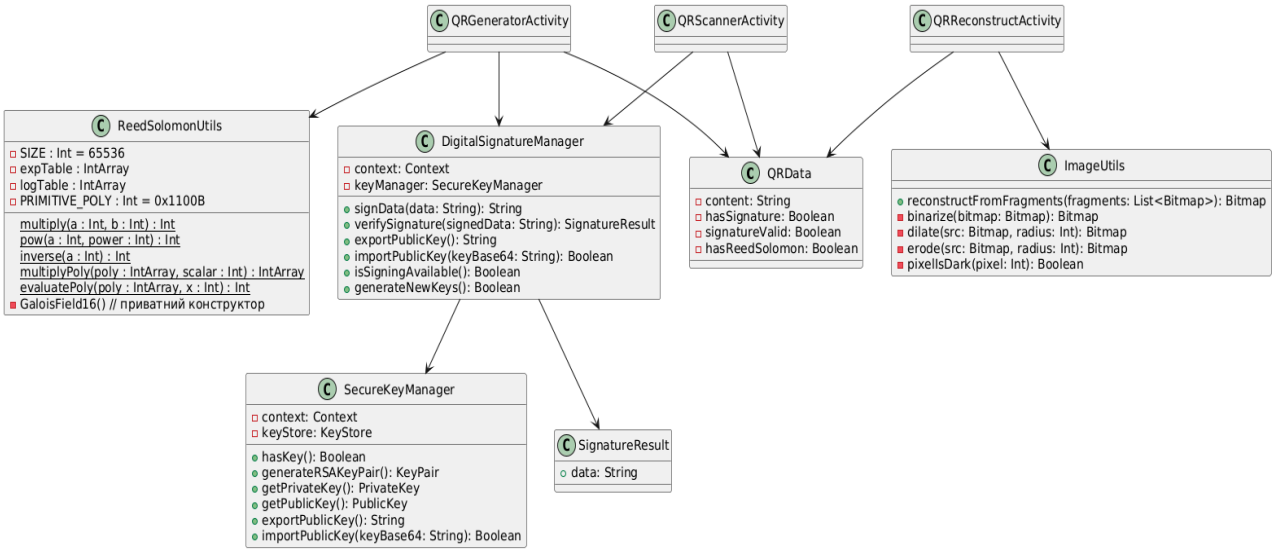


Рисунок 2.2 – Діаграма класів

Клас «QRData» слугує для зберігання даних QR-кодів, включаючи вміст самого QR-коду, прапорці наявності підпису, його валідності та використання корекції за Рідом-Соломоном. Клас «DigitalSignatureManager» забезпечує управління цифровими підписами, використовуючи «SecureKeyManager» для криптографічних операцій, а також повертає об’єкт «SignatureResult» із результатами перевірки. Клас «ImageUtils» відповідає за обробку зображень,

включаючи відновлення фрагментів, а «ReedSolomonUtils» реалізує операції в полі Галуа 2^{16}, що є основою для кодування даних.

Також на діаграмі можна побачити взаємозв'язки між класами та активностями. Так клас «QRGeneratorActivity» використовує активності «DigitalSignatureManager», «ReedSolomonUtils» і «QRData» для генерування QR-кодів, тоді як «QRScannerActivity» взаємодіє з «DigitalSignatureManager» і «QRData» для сканування. Клас «QRReconstructActivity» покладається на «ImageUtils» і «QRData» для відновлення QR-кодів із фрагментів. Зв'язки, такі як залежність «DigitalSignatureManager» від «SecureKeyManager» та повернення «SignatureResult», підкреслюють логічну організацію системи. Таким чином діагарама класів дала змогу зрозуміти роботу застосунку зсередини і взаємозв'язки між різними активностями.

2.3 Розробка методу цифрового підпису QR-коду

Електронний цифровий підпис – це вид електронного підпису отриманого за результатом криптографічного перетворення набору електронних даних, який додається дає змогу підтвердити його цілісність та ідентифікувати підписувача. Електронний цифровий підпис накладається за допомогою особистого ключа та перевіряється за допомогою відкритого ключа [16].

Під час написання коду для даного фрагменту програми було використано поєднання криптографічних алгоритмів SHA256, який використовується для хешування даних та RSA для шифрування цифрового підпису. Спочатку генерується довжина коду в 256 біт з вхідних даних з використанням Secure Hash Algorithm, згодом підключається алгоритм RSA і відбуваються чотири наступні кроки. Перш за все генерується ключ, під час якого обираються два простих числа, визначається модуль, застосовується функція Ейлера для, вибирається відкрита експонента і знаходиться приватна експонента оберненого за модулем числа. Другим кроком відбувається розподіл ключа на приватну і публічну частини, публічний ключ далі буде пересилатися за допомогою надійного каналу зв'язку Подальшими кроками є шифрування та

дешифруванння. На рисунку 2.3 зображено блок-схему алгоритму роботи цифрового підпису.

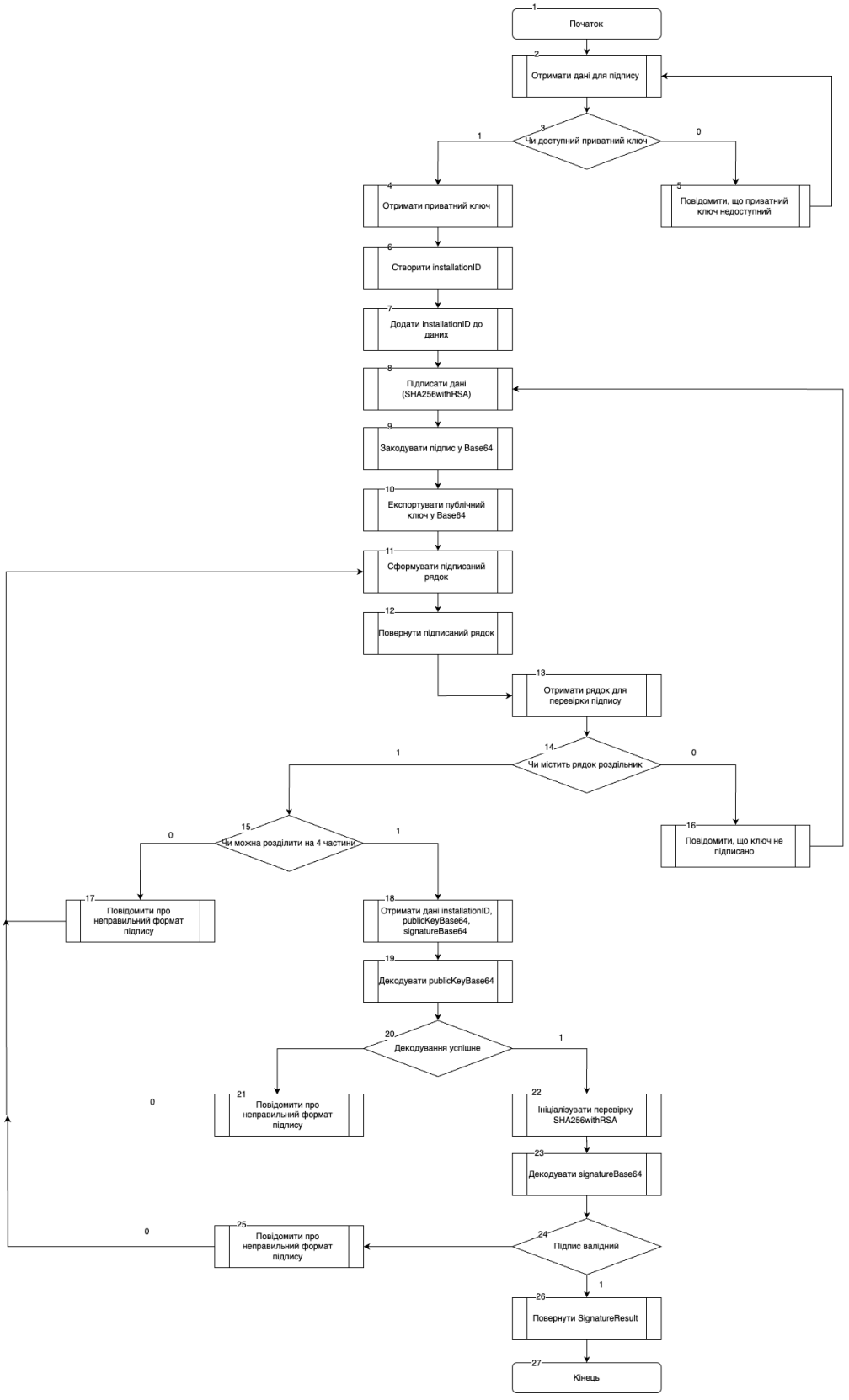


Рисунок 2.3 – Блок-схема алгоритму роботи цифрового підпису

Першим етапом роботи коду є отримання даних для підпису та перевірка доступності приватного ключа через «SecureKeyManager». Якщо ключ доступний, створюється новий унікальний «installationID», далі його необхідно додати до даних, та підписати за алгоритмом SHA256withRSA. Таким чином відбувається кодування підпису у формат Base64, наступним кроком йде експортування відкритого ключа у Base64 і з метою формування рядку у форматі data:SIGN:installationID:SIGN:publicKeyBase64:SIGN:signatureBase64. У разі відсутності приватного ключа користувачу повідомляється про помилку, і повертається значення null.

Після цього нагально необхідно перевірити підпис. Усе починається з отримання рядка аби перевірити та проаналізувати наявність роздільника :SIGN:. Якщо перевірка пройдена успішно і роздільник присутній, рядок треба розділяти на частини, і перевірити, чи є всі потрібні чотири компоненти: дані, «installationID», закодований відкритий ключ і підпис. У разі відсутності хоча б одного з компонентів перевірка припиняється з відповідним повідомленням про помилку. Якщо всі етапи пройшли без помилок, то відкритий ключ декодується, а опісля ініціалізується перевірка за алгоритмом SHA256withRSA, додаються дані та «installationID», а сам підпис надалі декодується з Base64 для подальшої валідації.

Важливо, що на цьому етапі використовується саме той алгоритм хешування та підпису, який було застосовано на стороні генерації. Наприкінці потрібно проаналізувати результат, отриманий у результаті перевірки. Якщо підпис валідний, повертається SignatureResult.Valid. У випадку помилки і невідповідності підпису буде встановлено результат SignatureResult.Invalid з метою повідомити про недійсність підпису. Якщо виникла інша помилка, наприклад, декодування ключа чи формат підпису неправильні, то буде повернено SignatureResult.Invalid із відповідним повідомленням про помилку. В кінці-кінців, якщо рядок не містить роздільника :SIGN:, то результатом буде повернення SignatureResult.NotSigned. Цей алгоритм повинен забезпечити надійну перевірку цілісності та автентичності даних у системі.

2.4 Модифікація кодів Ріда-Соломона

Стандартні коди Ріда-Соломона що використовуються для відновлення QR-кодів, зазвичай використовують поля Галуа 2^8, але якщо підвищити степінь хоча б до 16 степені, це може в рази покращити можливості відновлення втрачених даних. Напевно найголовнішою перевагою такого підходу, буде більша алфавітна потужність кодів Ріда-Соломона. Поля Галуа 2^8 містять собі 256 символів, на відміну від полів Галуа 2^{16}, які можуть вмістити аж 65 536. Тобто узагальнюючи можна сказати, що один символ коду зможе передати вдвічі більше інформації, та виправити більшу помилку у випадку пошкоджень більших за один байт.

Ще однією перевагою є те, що у випадку коли пошкодження не випадкові, а йдуть блоками, смугами абощо, то більші символи дозволять точніше локалізувати і виправити пошкодження подібного типу. Також не менш важливою є те що при збережені чи передачі файлів чи інших великих обсягів даних, модифіковані коди Ріда-Соломона дозволяють створення довших QR-кодів з більшим обсягом відновлюваної інформації.

Причиною чому дана модифікація не є широко застосованою є те що для не всі сканери можуть справитись зі зчитуванням такого типу QR-кодів. Проте оскільки даний мобільний застосунок зосереджується на збережені та відновленні конфіденційних даних, то це не є проблемою.

Для детальнішого опису алгоритму роботи змінених кодів Ріда-Соломона, було використано діаграму послідовності (див. рис. 2.4). Даний тип діаграми відображає взаємодії об'єктів впорядкованих за часом [17]. У ній проілюстровано взаємодію різних компонентів коду, спочатку йде виконання методу multiply(a, b) у скінченному полі Галуа 2^{16}. Тут відображається послідовність дій у часі, показуючи, як зовнішній користувач звертається до класу GaloisField16, а той, у свою чергу, використовує внутрішні таблиці logTable і expTable для обчислення результату. Загалом дана діаграма є доволі простою і зосереджена на ключовому методі множення, і описі реалізації операцій в полі Галуа 2^{16}.

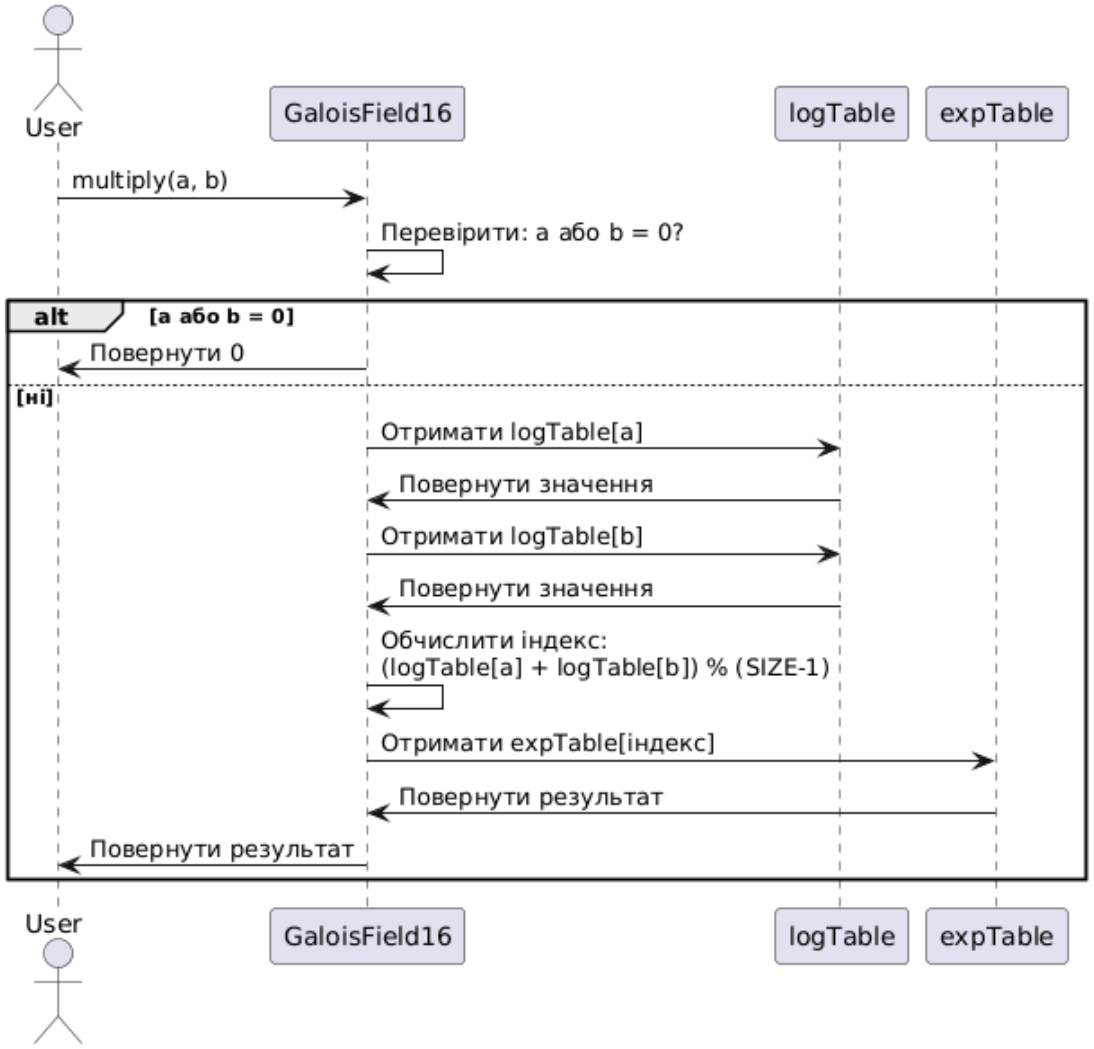


Рисунок 2.4 – Діаграма послідовності роботи модифікованих кодів Ріда-Соломона

Спочатку як користувач ініціює процес, для цього викликається метод multiply(a, b) класу GaloisField16. Цей виклик є початковим кроком, який повинен запустити логіку множення двох елементів у полі Галуа. Клас GaloisField16 при цьому виступає центральним об’єктом, саме він обробляє запит, і перш за все виконує внутрішню перевірку: чи дорівнює хоча б один із аргументів a або b нулю. Ця умова є критичною для поля Галуа, оскільки множення на нуль завжди дає нуль, що дасть змогу сильно оптимізувати обчислення.

У випадку якщо умова a або b = 0 виконується, клас GaloisField16 відразу має повернути результат 0 користувачу і завершити процес. Це альтернативна

гілка, яка відображає найпростіший сценарій, для уникнення складних обчислень. Таким чином цей підхід є дуже ефективним, бо він відповідає властивостям скінченного поля, де нульовий елемент не впливає на результат множення.

Інший можливий результат перевірки, це коли обидва аргументи відмінні від нуля, тоді діаграма демонструє детальний процес обчислення, цей сценарій є значно складнішим. Клас GaloisField16 звертається до таблиці logTable, щоб отримати логарифмічні значення для a і b. Ці значення повертаються з logTable назад до класу, де обчислюється сума логарифмів з модулем (SIZE-1), що є частиною алгоритму для множення в полях Галуа 2^{16}. Такий крок використовує властивість логарифмічних таблиць, які дозволять швидко виконати операції в полі.

Після цього клас GaloisField16 звертається до таблиці expTable, і передасть обчислений індекс, аби отримати результат множення. Таблиця expTable повертатиме відповідне значення, воно і буде кінцевим результатом операції множення в полі Галуа. Користувач отримає цей результат, і таким чином завершиться послідовність дій. Наведений підхід підкреслює, як таблиці expTable і logTable, ініціалізовані на основі примітивного полінома 0x1100B, забезпечують ефективне виконання операцій у полях Галуа 2^{16}.

Ця діаграма послідовності акцентує увагу на взаємодії між користувачем, класом GaloisField16 і його внутрішніми таблицями, звертаючи основну увагу на реалізаці. множення в скінченному полі Галуа 2^{16}. Тут вказані ключові кроки: перевірка нуля, звернення до таблиць і повернення результату.

2.5 Розробка методу відновлення QR-коду з частин

Метод відновлення з частин, частково базується на модифікації алгоритму Ріда-Соломона. Основна ідея полягає в тому, щоб спочатку розподілити QR-код на фрагменти, далі додати надлишкові дані для корекції помилок і так забезпечити можливість реконструкції оригінального коду за наявності достатньої кількості фрагментів. Важливо, щоб подібні фрагменти знаходились

якомога далі таким чином, аби при пошкодженні однієї частини QR-коду, була змога його відновити. На рисунку 2.5 зображено блок-схему алгоритму роботи відновлення QR-коду з частин.

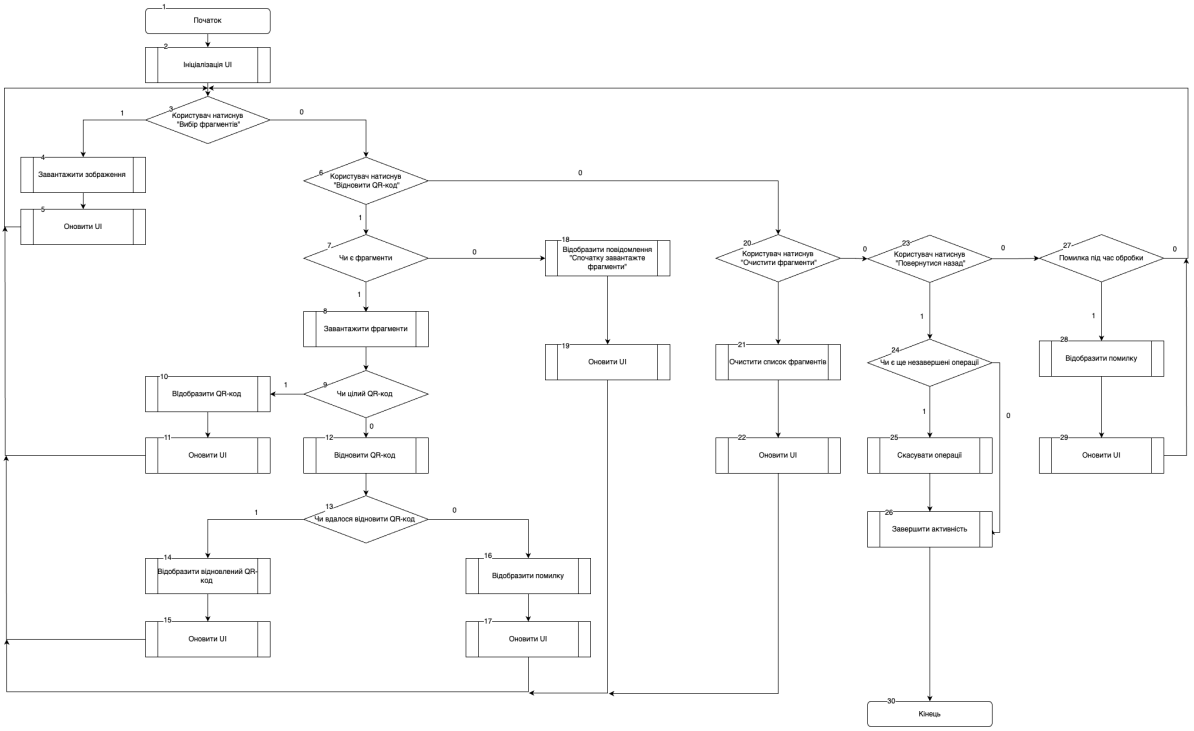


Рисунок 2.5 – Блок-схема алгоритму роботи відновлення QR-коду з частин

Алгоритм бере початок з ініціалізації порграми, де відкривається початковий інтерфейс користувача На даному етапі користувачу пропонується обрати дію «вибрати фрагмент», і залежно від його рішення, якщо кнопка буде натиснута, то з’явиться можливість завантажити зображення фрагмента QR-коду, після чого інтерфейс оновлюється для відображення завантаженого вмісту.

Далі програма проводить перевірку, чи було обрано фотографій для обробки, і якщо так, то проводиться аналіз, чи достатньо фрагментів для відновлення. У разі недостатньої кількості фрагментів користувача повертають до завантаження додаткових частин, супроводжуючи це відповідними повідомленнями. Якщо фрагментів достатньо, то наступним етапом є

відновлення QR-коду і подальше відображення його на екрані. У разі невдачі відновлення, користувач отримує помилку.

Також є можливість очистити фрагменти, при потребі, після чого інтерфейс знову відображає оновлений стан. Також є можливість повернутися назад до головного меню, під час даного процесу перевіряється наявність незавершених операцій, що дозволяє або продовжити відновлення, або очистити інформацію.

2.6 Висновки

Отже, у другому розділі було розроблено архітектуру мобільного застосунку за допомогою діаграми прецедентів, що визначає взаємодії між користувачем та основними функціями системи, такими як сканування та відновлення QR-кодів. Також створено модель системи за допомогою UML діаграми класів, яка забезпечує чітке розуміння структури та взаємозв’язків між компонентами. Крім того, розроблено метод цифрового підпису з алгоритмом SHA256withRSA для підвищення безпеки даних, а також модифіковано алгоритм Ріда-Соломона для ефективного відновлення пошкоджених QR-кодів. А також, описано метод відновлення QR-коду з частин, що гарантує зручну та надійну реконструкцію за наявності достатньої кількості фрагментів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ГЕНЕРУВАННЯ ТА ВІДНОВЛЕННЯ QR-КОДІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для розробки мобільного застосунку вибір мови програмування є неймовірно важливим рішенням. Можна з упевненістю стверджувати, що саме від цього залежатиме як фактична робота застосунку, так і його подальша доля, можливість оновлення чи додавання нового функціоналу. Тому приймаючи таке рішення варто досконально зважувати усі переваги та недоліки. Для попереднього розгляду було обрано три сучасні мови програмування та один фреймворк.

Swift є мовою програмування, розроблена Apple на основі мов C і Objective C, призначеною для створення додатків під iOS, macOS, watchOS і tvOS [18]. Слід зазначити що вона має легкий для розуміння синтаксис і підтримує як об’єктно-орієнтоване, так і функціональне програмування, що робить її простою для використання розробниками, що звикли писати на інших мовах. Також варто згадати те, що її компілятор є добре оптимізованим, а його функція безпечного з точки зору пам’яті коду зменшує ризик витоків. Однак з іншого боку Swift має вагомий недолік, а саме її обмежена сумісність із приладами поза iOS.

Flutter – це фреймворк об’єктно-орієнтованої мови програмування Dart, створений компанією Google [19]. Він дозволяє створювати програмні застосунки не обмежені жодною платформою. У Flutter базовим компонент це віджит, завдяки якому, як і завдяки швидкому рендерингу через Skia-рушій, ця мова програмування може бути лідером у гнучкому підході до створення інтерфейсів. Flutter є кросплатформеною мовою.

Мова програмування Kotlin статично типізована мова розроблена компанією JetBrains, хоча вона розроблена поверх JVM її також можна компілювати в JavaScript. Також перевагою Kotlin є підтримка null-safety і

відносно стислий синтаксис, що дає можливість зменшити кількість boilerplate-коду. Більш того оскільки ця мова програмування повністю сумісна з Java та інтегрована в Android Studio, то є можливість використовувати існуючі бібліотеки і фреймворки. Оскільки Kotlin підтримує корутини, це дає змогу забезпечити асинхронне програмування, що в свою чергу важливо для мобільних застосунків бо дає змогу обробляти дані в реальному часі та працювати з мережевими запитами [20].

Python, як і Kotlin, є об’єктно-орієнтованою мовою програмування з високим рівнем абстракції, що дозволяє швидко створювати прототипи [21]. Вона також має широкі можливості для реалізації шифрування та безпеки. Хоча за допомогою фреймворків Python можна розробляти кросплатформенні застосунки, обмежений контроль над нативними компонентами знижує доцільність його використання для мобільних додатків. Порівняння функціональних характеристик Swift, Flutter, Kotlin та Python наведено в таблиці 3.1.

Таблиця 3.1 – Таблиця порівняння функціональних характеристик мов програмування

Критерій	Swift	Flutter	Kotlin	Python
Глибока інтеграція з Android SDK та системними API	0	0	1	0
Нативна підтримка електронного цифрового підпису	0	0	1	1
Можливість низькорівневої роботи з QR-кодами та двійковими даними	1	1	1	0
Підтримка великих полів Галуа 2^{16} у сторонніх криптобібліотеках	0	0	1	1
Придатність до офлайн-підпису на мобільних пристроях без серверів	0	1	1	0
Документована підтримка Reed–Solomon з можливістю розширення	0	0	1	1
Підсумок	1	2	6	3

Для порівняння було обрано шість критеріїв. Ці критерії стосуються не тільки функціональні можливості мов, а й що найголовніше специфічних вимог до застосунку орієнтованого на роботу з полями Галуа та обробку і зберігання конфіденційної інформації. Перш за все, оскільки Kotlin є офіційною мовою для Android, то тільки з його допомогою можна реалізувати глибоку інтеграція з Android SDK, які необхідні для доступу до Android Keystore та інших системних сервісів. Другим критерієм є нативна підтримка електронного цифрового підпису, оскільки в додатку він буде використовуватись, то важливо мати підтримку криптографії. Третій критерій було обрано тому, що необхідно мати функціонал, для формування коду, який буде містити як відкритий ключ, так і підпис, мова програмування Python на мобільних пристроях обмежена, тому в ній ця можливість відсутня. Для того щоб реалізувати модифіковані коди Ріда-Соломона, критично необхідно мати до доступ до бібліотек, які дозволяють працювати з великими скінченними полями. Для застосунку який може опрацьовувати конфіденційні дані, також важливою є можливість роботи офлайн, яку надають Kotlin і Flutter, хоча останній тільки використовуючи платформенний код. Окрім того використовуючи коди Ріда-Соломона, потрібно мати змогу налаштовувати або розширювати бібліотеки кодування/декодування, тому останній критерій є не менш важливим. Отже, Kotlin отримує 6 балів із 6 і відповідає всім вимогам, тому буде обрано саме її.

Не менш важливим за вибір мови програмування, є вибір середовища розробки. Оскільки це не тільки впливає на продуктивність роботи розробника і швидкість розробки, а й на можливість протестувати застосунок чи інтегрувати його з різноманітними бібліотеками. Для попереднього розгляду було обрано 4 IDE.

Наразі найпопулярнішим середовищем мобільної розробки на Android є Android Studio, воно розроблено компанією Google на базі IntelliJ IDEA. У середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проєктування застосунків, що працюють на пристроях з екранами різної роздільності [22]. Крім того, Android

Studio підтримує інтеграцію з Gradle для автоматизації збірки проєктів, включає вбудований емулятор для тестування та надає розширені функції налагодження, що значно полегшують розробку та оптимізацію мобільних застосунків. Це важливо для створюваного мобільного додатку, оскільки забезпечення сумісності з різними пристроями та версіями Android є ключовим для надійного сканування, генерації та відновлення QR-кодів, а вбудовані інструменти тестування та налагодження дозволяють ефективно перевіряти функції безпеки, такі як цифровий підпис, та коректність роботи з пошкодженими фрагментами.

IntelliJ IDEA – багатофункціональна IDE від JetBrains, яка першою почала підтримку Kotlin. Вона добре підходить для розробки не лише Android-додатків, але й серверних, десктопних і багатоплатформних програм. IntelliJ дозволяє працювати з Kotlin Multiplatform [23]. Хоч вона не має такої тісної прив’язки до Android, як Android Studio, зате дає ширші можливості для застоснків, де потрібна багатокомпонентна логіка. Крім того, IDE пропонує потужні інструменти рефакторингу, автодоповнення коду та аналізу залежностей, що значно полегшує створення складних архітектур, таких як інтеграція криптографічних модулів чи обробки даних у реальному часі, що є важливим для мого застосунку з генерацією та відновленням QR-кодів.

Eclipse – це відома, але дещо застаріла IDE, що здобула популярність завдяки розробці на Java [24]. Хоча вона підтримує Kotlin за допомогою додаткових плагінів, інтеграція обмежена й менш стабільна порівняно з IntelliJ або Android Studio. Використання Eclipse для Kotlin можливе, але потребує додаткової конфігурації та часто поступається за зручністю.

VS Code – це легке, кросплатформне середовище, яке підтримує Kotlin через плагіни. Воно не є повноцінною IDE, але дозволяє редагувати, запускати й компілювати Kotlin-код у простих застосунках [25]. Його головна перевага це мінімалістичність і швидкість. Через обмежену підтримку Android SDK і складнощі з налаштуванням складних збірок, VS Code більше підходить для написання простих алгоритмів, тестування логіки або розробки Kotlin

Multiplatform-частин. Порівняння середовищ розробки Android Studio, IntelliJ IDEA, Eclipse та VS Code наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння середовищ розробки

Критерій	Android Studio	IntelliJ IDEA	Eclipse	VS Code
Офіційна підтримка Android SDK, Keystore	1	0	0	0
Повноцінна інтеграція Kotlin без додаткових налаштувань	1	1	0	0
Вбудований UI-редактор, Android емулятор	1	0	0	0
Підтримка легких скриптів або CLI-модулів	0	1	1	1
Можливість реалізації криптографії, QR та Gradle-збірок	1	1	1	0
Мінімальна підтримка Kotlin	0	0	1	1
Підсумок	4	3	3	2

Результати порівняння свідчать про явну перевагу Android Studio як основного середовища для розробки Kotlin-застосунку з підтримкою QR-кодів, криптографії та Android Keystore. Завдяки офіційній підтримці Android SDK, вбудованому емулятору, зручному UI-редактору та безшовній інтеграції Kotlin, Android Studio отримала найвищу оцінку. Таким чином, Android Studio є найбільш доцільним вибором.

3.2 Розробка модулю меню застосунку

Меню мобільного застосунку відіграє ключову роль у забезпеченні зручного та інтуїтивно зрозумілого досвіду користувача, оскільки саме з ним користувач стикається одразу після запуску програми. Воно є першим елементом, який визначає враження від застосунку, полегшує навігацію та надає швидкий доступ до основних функцій, що є особливо важливим для

застосунків, таких як мій, де користувачі потребують ефективного доступу до сканування, створення та відновлення QR-кодів. Чітке та логічно структуроване меню сприяє підвищенню залученості користувачів і зменшує час на освоєння функціоналу, що є критичним для забезпечення їхньої задоволеності. На рисунку 3.1 наведено прототип головного меню мобільного застосунку, який відображає його початковий дизайн та структуру.

Рисунок 3.1 – Прототип інтерфейсу меню мобільного застосунку

Прототип включає чотири основні опції: «Сканувати QR-код», «Створити QR-код», «Відновити QR-код з фрагментів» та заголовок «Назва застосунку» у верхній частині екрана. Кожна опція представлена у вигляді кнопки з чітким текстом, розміщеним вертикально один під одним, що забезпечує простоту використання. Дизайн виконано в мінімалістичному стилі з використанням сірих кнопок на світлому фоні, що створює контрастність і полегшує сприйняття, а також сприяє зручному розміщенню елементів на екранах різних розмірів. Планується додати кольори до дизайну: при світлій темі кнопки матимуть колір #63519F, а при темній темі – #CDBDFB, що забезпечить гармонійний вигляд і кращу адаптивність інтерфейсу.

3.3 Розробка модулів генерування QR-коду

Розробка модуля генерування QR-коду є важливим етапом створення мобільного застосунку, що дозволяє користувачам створювати QR-коди з різними типами даних. Процес генерування реалізовано через клас «QRGeneratorActivity», який координує введення даних користувачем, їх обробку та створення зображення QR-коду за допомогою бібліотеки «QRCodeWriter». Залежно від обраних налаштувань, модуль підтримує додавання цифрового підпису для забезпечення безпеки або корекцію помилок за допомогою модифікованого алгоритму Ріда-Соломона, що підвищує стійкість коду до пошкоджень. Для детального відображення логіки цього процесу використано діаграму послідовності, яка наочно демонструє взаємодію між компонентами, наведену на рисунку 3.2.

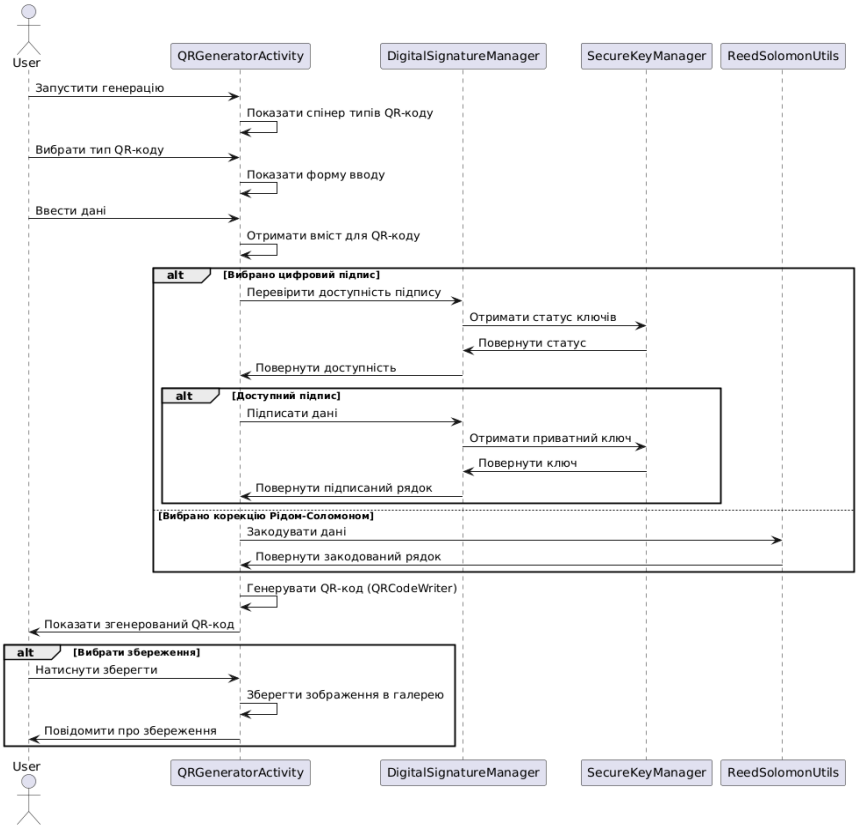


Рисунок 3.2 – Діаграма послідовності генерування QR-коду

Процес починається з дії користувача, який запускає генерацію через «QRGeneratorActivity». Спочатку відображається спінер для вибору типу

QR-коду, після чого користувач вводить дані в відповідну форму. Далі «QRGeneratorActivity» обробляє вміст: якщо обрано цифровий підпис, звертається до «DigitalSignatureManager» для перевірки доступності підпису через «SecureKeyManager», а при наявності ключа виконує підписання даних. Якщо ж активовано корекцію Ріда-Соломоном, дані передаються до «ReedSolomonUtils» для кодування. Після обробки «QRGeneratorActivity» генерує QR-код і відображає його користувачу, пропонуючи зберегти зображення в галерею з відповідним повідомленням про успішність операції. На основі цієї діаграми створено прототип модуля генерування QR-коду, представлений на рисунку 3.3.

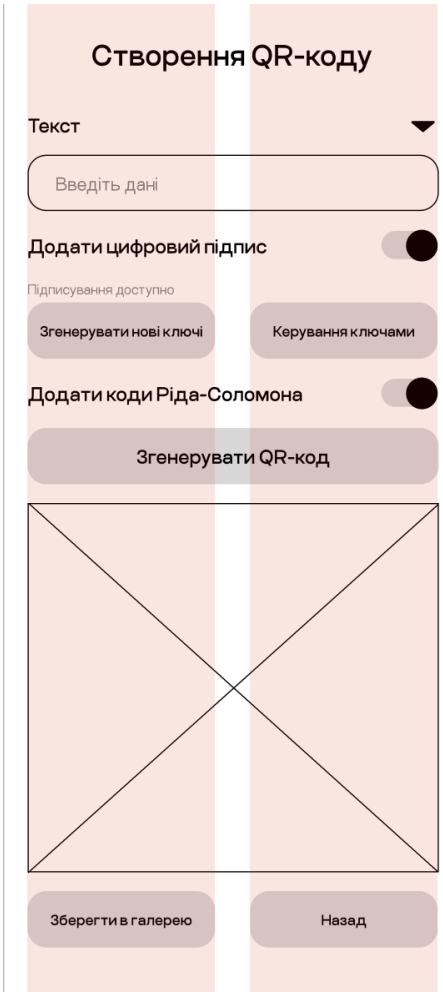


Рисунок 3.3 – Прототип модулю створення QR-коду

Прототип відображає інтерфейс із заголовком «Створення QR-коду» та двома основними секціями: вибір типу даних і додавання цифрового підпису чи корекції Ріда-Соломоном. Інтерфейс побудовано на сітці з двох колонок: ліва колонка містить опції вибору, а права – додаткові налаштування та кнопку генерації. Дизайн наразі виконано в мінімалістичному стилі з білим фоном і сірими елементами, але колірна схема для кнопок буде додана уже в фінальному дизайні застосунку.

3.4 Розробка модулів зчитування QR-коду

Модуль зчитування QR-кодів є потрібним компонентом мобільного застосунку, що забезпечує можливість сканування та обробки QR-кодів у реальному часі.

Діаграма діяльності – це тип UML-діаграми, який описує алгоритм виконання дій у вигляді потоку активностей [26]. Вона дозволяє наочно представити логіку обробки сценарію, включаючи умови, цикли, розгалуження та паралельні гілки. Саме діаграму діяльності було обрано для опису процесу відновлення QR-коду, оскільки вона дозволяє чітко відобразити залежність між перевірками, розгалуженнями та подальшими діями в інтерфейсі користувача, дану діаграму зображено на рисунку 3.4.

Цей модуль реалізовано через клас «QRScannerActivity», який використовує бібліотеку ML Kit для аналізу кадрів із камери та витягнення вмісту QR-коду. Процес включає перевірку цифрового підпису через «DigitalSignatureManager» для забезпечення автентичності даних, а також підтримку автоматичних дій, таких як відкриття URL або виконання команд, залежно від вмісту.

Окрім базового зчитування, модуль також реалізує обробку помилок – наприклад, некоректного формату даних з відповідним інформуванням користувача через інтерфейс. У разі успішної верифікації підпису застосунок продовжує виконання відповідної дії, пов’язаної з вмістом QR-коду.

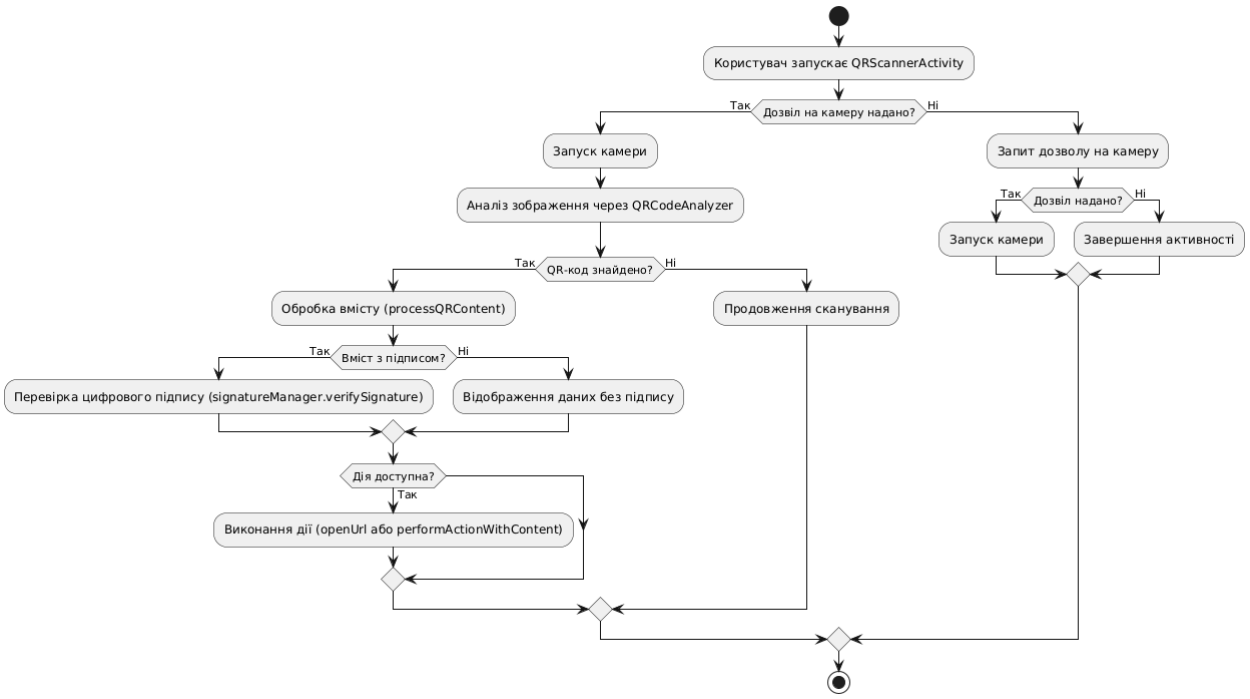


Рисунок 3.4 – Діаграма діяльності зчитування QR-коду

Перш за все, ініціалізується «QRScannerActivity», після чого перевіряється наявність дозволу на використання камери. Якщо дозвіл надано, камера запускається, і «QRCodeAnalyzer» аналізує кадри для отримання вмісту QR-коду. Якщо вміст починається з «PUBLIC_KEY:», імпортується публічний ключ, і інтерфейс оновлюється з результатом. У протилежному випадку перевіряється цифровий підпис: при валідному підписі інтерфейс показує дані з позначкою «Підпис дійсний», при недійсному – «Підпис недійсний», а при відсутності підпису – просто дані. Далі, якщо вміст є URL або дією, відображається кнопка дії, яку користувач може натиснути для виконання, інакше кнопка ховається. Якщо дозвіл на камеру відсутній, система запитує його: при позитивній відповіді камера запускається, при негативній – показує повідомлення про необхідність дозволу та завершує активність.

3.5 Розробка модулів відновлення QR-коду

Модуль відновлення реалізовано через клас «QRReconstructActivity», який дозволяє користувачам завантажувати фрагменти зображень, обробляти їх за допомогою класу «ImageUtils» для відновлення цілісного зображення, а потім

зчитувати вміст за допомогою «BarcodeScanning». Процес включає відображення прогресу та оновлення інтерфейсу з результатами, що забезпечує зручність використання. Для детального аналізу логіки роботи модуля застосовано діаграму послідовності, зображено на рисунку 3.5.

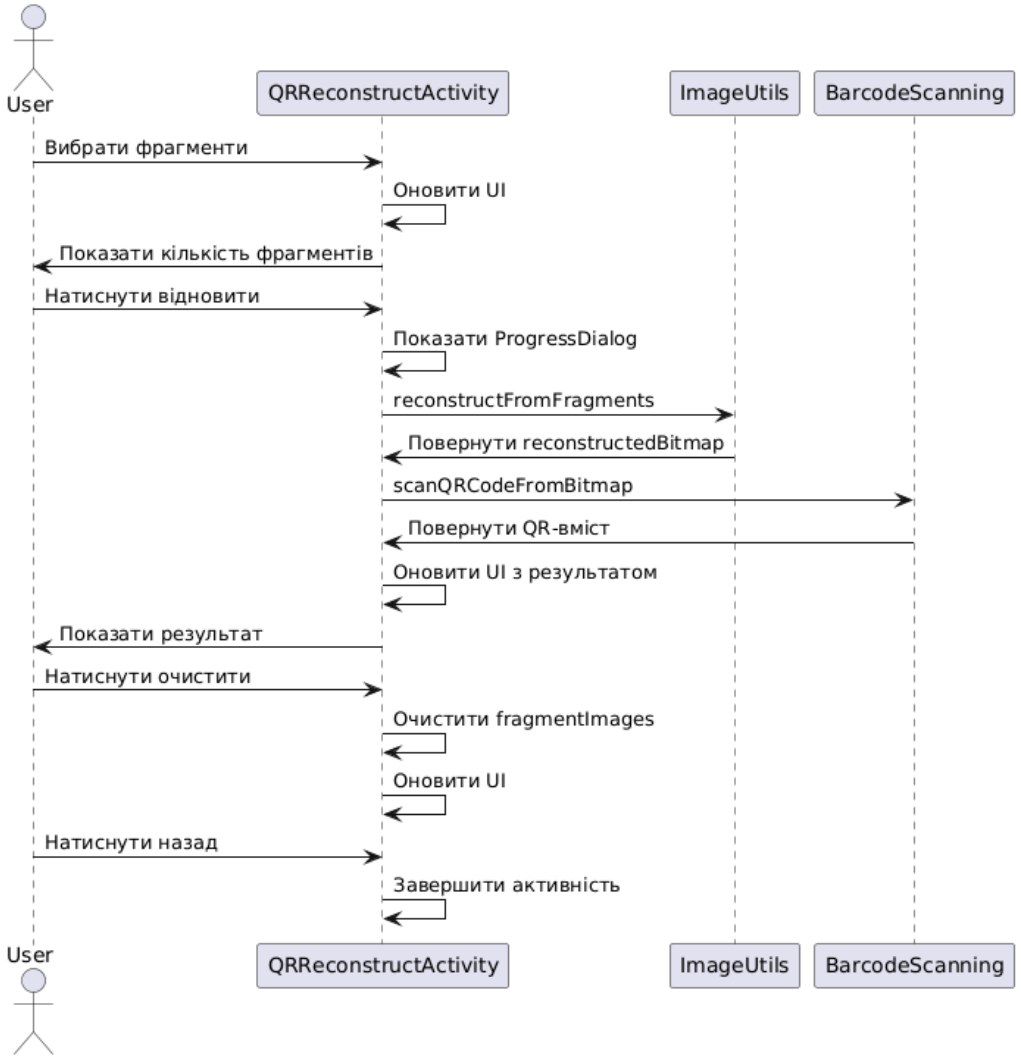


Рисунок 3.5 – Діаграма послідовності відновлення QR-коду

Спочатку користувач обирає фрагменти, після чого «QRReconstructActivity» оновлює інтерфейс і показує кількість завантажених фрагментів. При натисканні кнопки «відновити» відображається діалог завантаження, а «ImageUtils» виконує реконструкцію зображення , повертаючи відновлений «reconstructedBitmap». Далі «BarcodeScanning» аналізує зображення для отримання вмісту QR-коду, після чого «QRReconstructActivity»

оновлює інтерфейс із результатом і повідомляє користувача. На основі діаграми створено прототип модуля відновлення QR-коду, представлений на рисунку 3.6.

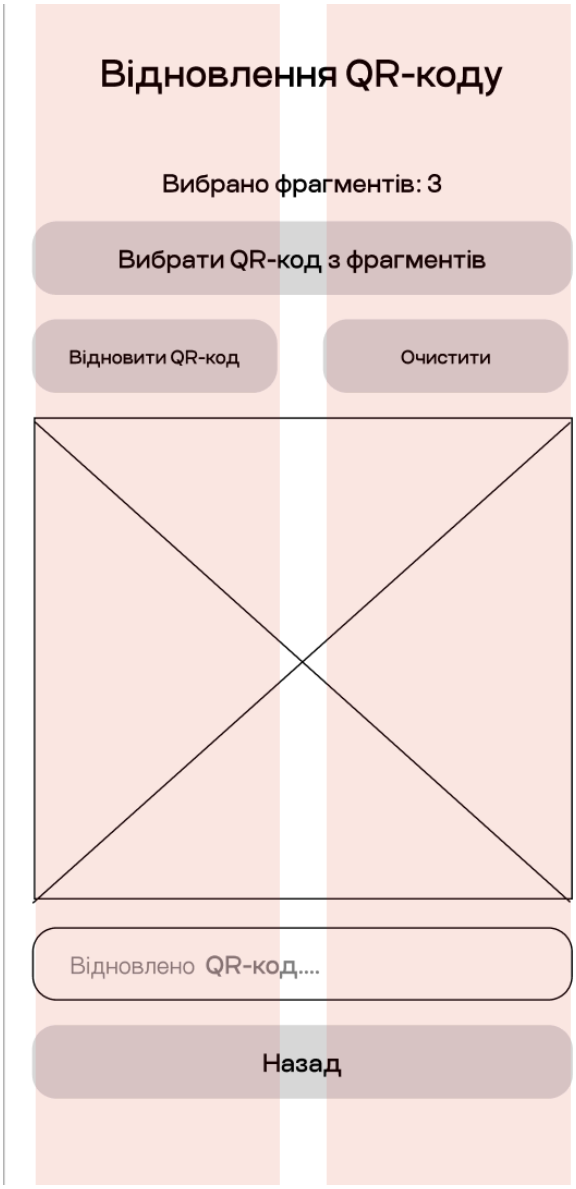


Рисунок 3.6 – Прототип модулю відновлення QR-коду

Прототип включає заголовок «Відновлення QR-коду», лічильник завантажених фрагментів та кнопки «Відновити QR-код з фрагментів» і «Очистити». Центральна частина відображає місце для відновленого зображення, а нижче розташована кнопка «Назад» для завершення.

3.6 Висновки

У третьому розділі проведено аналіз та обґрунтовано вибір засобів для реалізації мобільного застосунку, зокрема мови програмування Kotlin та середовища розробки Android Studio, які забезпечують оптимальну інтеграцію з Android SDK та підтримку криптографії. Також розроблено графічні схеми, діаграми послідовності та діяльності для ключових модулів, таких як меню, генерування, зчитування та відновлення QR-кодів, детально описавши їх функціональність і логіку роботи. Структура інтерфейсу, побудована на прототипах із сітковою організацією та планованою колірною схемою, гарантує зручність використання та ефективну взаємодію з користувачем.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

4.1 Аналіз методів тестування програмного забезпечення

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, спрямованого на забезпечення якості, надійності та відповідності продукту вимогам. Це систематичний процес, який включає виконання програми в контрольованих умовах для виявлення дефектів, помилок або відхилень від очікуваної поведінки. Тестування охоплює різні рівні, такі як юніт-тестування , інтеграційне тестування, системне тестування та приймальне тестування. Крім того, існують спеціалізовані типи, як-от продуктивність для оцінки швидкості та навантаження, та безпека для виявлення вразливостей.

Тестування є ключовим етапом у життєвому циклі розробки програмного забезпечення , який забезпечує, що продукт відповідає очікуванням користувачів і бізнес-вимогам. Воно допомагає зменшити ризики, пов’язані з помилками, які можуть призвести до фінансових втрат, пошкодження репутації або юридичних проблем.

Тестування має вирішальне значення з кількох причин, які підкріплені численними дослідженнями. По-перше, воно дозволяє виявляти та виправляти дефекти на ранніх етапах, що значно знижує витрати. Наприклад, за даними IBM, виправлення дефекту на етапі вимог коштує 1 одиницю, на етапі тестування 10 одиниць, а після випуску понад 100 одиниць [27]. Це підтверджує правило 1:10:100, яке вказує на експоненціальне зростання витрат на виправлення помилок на пізніх стадіях.

По-друге, тестування забезпечує відповідність продукту потребам користувачів, що підвищує їхню задоволеність і довіру. Наприклад, дослідження показують, що якісне тестування може зменшити кількість скарг від користувачів на 85-95% [28].

По-третє, воно знижує ризики, пов’язані з безпекою та продуктивністю, особливо в критичних системах, таких як медичні або фінансові програми.

Також, тестування допомагає компаніям відповідати нормативним вимогам, що є важливим у регульованих галузях.

Дослідження CISQ (2022) оцінює, що поганої якості програмне забезпечення коштує економіці США $2,41 трлн щорічно, і значна частина цих витрат могла б бути зменшена завдяки вдосконаленню тестування та забезпечення якості, що підкреслює економічну вигоду від інвестицій у ці процеси [29].

Тестування має значний вплив на кінцевий продукт, покращуючи його якість, стабільність і конкурентоспроможність. Воно зменшує кількість дефектів, які досягають користувачів, що знижує ймовірність негативних відгуків і витрат на підтримку. Наприклад, за даними Capers Jones, ретельне тестування може зменшити кількість помилок у релізній версії до менш ніж 5 на 1000 рядків коду [30], що є критичним для комерційного успіху.

Тестування також покращує стабільність і продуктивність, усуваючи проблеми, які можуть призвести до збоїв або повільної роботи. Це підвищує довіру користувачів, що сприяє лояльності та позитивним відгукам. Економічний ефект також значний: за даними IBM, кожен долар, вкладений у тестування, заощаджує $10 на етапі підтримки [27]. Крім того, ретельне тестування може заощадити до 30% загального бюджету проєкту, зменшуючи потребу в переробках і підтримці.

Тестування проводиться на різних етапах SDLC, і кожен етап має свої цілі та методи. На етапі розробки тестування інтегрується в процес створення програмного забезпечення, що дозволяє виявляти проблеми ще до завершення продукту. Наприклад, метод Test-Driven Development передбачає написання тестів перед кодом, що допомагає створювати більш надійні компоненти. Юніт-тестування перевіряє окремі модулі, тоді як інтеграційне тестування оцінює їхню взаємодію. За даними ISTQB, раннє тестування дозволяє виявити до 70% дефектів ще до переходу до наступних етапів [28]. Це значно знижує витрати, оскільки виправлення помилок на цьому етапі є дешевшим.

Після завершення основної розробки проводяться системне та приймальне тестування, щоб переконатися, що продукт відповідає всім вимогам і готовий до використання. Регресійне тестування перевіряє, чи не вплинули нові зміни на існуючу функціональність. Ці етапи дозволяють виявити більшість залишкових дефектів перед релізом. Тестування значно еволюціонувало з розвитком методологій Agile та DevOps, які стали домінуючими в сучасній розробці. Agile фокусується на співпраці, швидкій зворотній зв'язку та ітеративному розвитку, інтегруючи тестування в кожен спринт. Це включає юніт-тестування, інтеграційне тестування та приймальне тестування, часто з акцентом на автоматизацію для підтримки швидких циклів розробки.

DevOps розширює цей підхід, підкреслюючи безперервне тестування протягом усього процесу розробки та доставки. Тести автоматизовані і запускаються на кожному етапі Continuous Integration/Continuous Delivery, забезпечуючи, що лише якісний код надходить до продакшну. Це включає безперервне інтеграційне тестування, де зміни коду автоматично будуються, тестуються та перевіряються перед злиттям у основну гілку.

Ключові переваги включають:

– перенесення тестування на ранні етапи для раннього виявлення дефектів;

– автоматизовані тести запускаються безперервно, забезпечуючи швидку зворотню зв'язку;

– значна частина тестування автоматизована для підтримки частих релізів;

– тестувальники, розробники та оператори працюють разом, що спрощує процес та покращує якість.

Дослідження Puppet (2023) показує, що організації, які використовують DevOps, розгортають код 208 разів частіше і 106 разів швидше, ніж їхні аналоги. Крім того, комбіноване використання Agile та DevOps призводить до скорочення часу виходу на ринок, покращення якості та зменшення витрат [31].

Тестування мобільних застосунків – це комплексний процес оцінки та перевірки програмного забезпечення, призначеного для роботи на мобільних пристроях, таких як смартфони та планшети [32]. Воно спрямоване на забезпечення відповідності додатка вимогам, його стабільності, безпеки та зручності для користувачів. На відміну від тестування веб-додатків, тестування мобільних застосунків є складнішим через фрагментацію пристроїв, різноманітність операційних систем, розмірів екранів, мережевих умов і поведінки користувачів. Основні типи тестування включають:

1. Функціональне тестування, перевіряє, чи всі функції додатка працюють відповідно до специфікацій. Це включає тестування елементів користувацького інтерфейсу , баз даних, API та інших компонентів. Наприклад, перевіряється, чи коректно працює функція входу в систему або обробка платежів.

2. Тестування продуктивності, оцінює швидкість, стабільність і реакцію додатка в різних умовах, таких як низький заряд батареї, слабке мережеве з’єднання або високе навантаження. Це включає тестування навантаження, стрес-тестування та перевірку масштабованості.

3. Тестування безпеки, виявляє вразливості, які можуть загрожувати конфіденційності даних користувачів або функціональності додатка. Це особливо важливо для додатків, які обробляють чутливу інформацію, наприклад, банківські чи медичні дані. Тестування безпеки включає перевірку захисту від атак типу SQL-ін’єкцій, шифрування даних і безпечного зберігання.

4. Тестування сумісності, дає змогу переконатися, що додаток працює коректно на різних пристроях, розмірах екранів, роздільних здатностях і версіях операційних систем. Наприклад, додаток має працювати однаково добре на iPhone 14 з iOS 17 і на бюджетному Android-пристрої з Android 10.

5. Тестування зручності використання, оцінює, наскільки додаток є інтуїтивно зрозумілим і зручним для користувачів. Це включає перевірку дизайну інтерфейсу, навігації, розміру кнопок і загального користувацького досвіду.

6. Тестування локалізації, перевіряє, чи додаток коректно працює в різних регіонах і мовах, адаптуючись до культурних і мовних особливостей. Це включає перевірку перекладів, форматів дати, валюти та відповідність місцевим стандартам.

Тестування мобільних застосунків є критично важливим через величезну різноманітність мобільних пристроїв і операційних систем. У 2022 році в Google Play Store було доступно понад 3 мільйони додатків, а в Apple App Store – 1,6 мільйона, і ця кількість постійно зростає [33]. Кожен пристрій може мати унікальні характеристики, такі як розмір екрана, роздільна здатність, апаратне забезпечення та версія ОС. Тестування сумісності забезпечує коректну роботу додатка на всіх популярних платформах, що дозволяє охопити ширшу аудиторію та уникнути проблем із несумісністю. Без цього тестування користувачі можуть зіткнутися з помилками, які погіршують їхній досвід.

Користувачі мають високі очікування щодо якості мобільних додатків, і будь-які проблеми можуть призвести до їхнього видалення. Згідно з дослідженням BrowserStack, 94% користувачів видаляють додаток протягом 30 днів після встановлення, якщо він не відповідає їхнім очікуванням, а 70% роблять це через повільне завантаження або збої [34]. Тестування допомагає виявити та усунути такі проблеми, як повільна робота, зависання чи некоректне відображення інтерфейсу, що підвищує задоволеність користувачів і сприяє їх утриманню.

Помилки в мобільних додатках можуть призвести до значних фінансових втрат. Затримки у випуску через дефекти коштують компаніям значних сум: 75% компаній повідомляють про втрати понад 100 000 доларів щорічно через повільні релізи. Ефективне тестування дозволяє виявляти проблеми на ранніх етапах, зменшуючи витрати на виправлення помилок після випуску та прискорюючи вихід на ринок [34]. Це особливо важливо для компаній, які залежать від швидкого випуску нових функцій для збереження конкурентоспроможності.

Зі зростанням кіберзагроз, таких як витоки даних, тестування безпеки стало критично важливим. Мобільні додатки, які обробляють чутливу інформацію, наприклад, банківські або медичні дані, повинні відповідати стандартам, таким як Загальний регламент захисту даних або Каліфорнійський закон про захист прав споживачів. Недотримання цих стандартів може призвести до штрафів, юридичних проблем і втрати репутації. Тестування безпеки включає перевірку шифрування даних, захисту від атак типу SQL-ін’єкцій і безпечного зберігання інформації, що забезпечує захист користувачів.

Мобільний ринок є надзвичайно конкурентним, адже щодня в Google Play і App Store додаються понад 3000 нових додатків [34]. Якісне тестування дозволяє виокремити додаток серед конкурентів, забезпечуючи безперебійну роботу, інтуїтивний інтерфейс і високу продуктивність. Без належного тестування додаток може бути втрачений серед конкурентів, що призведе до втрати користувачів і доходів.

Ретельне тестування дозволяє виявити до 95% дефектів до релізу, що значно знижує кількість скарг від користувачів і витрати на підтримку після випуску. Наприклад, тестування функціональності та сумісності допомагає усунути проблеми, які можуть спричинити збої або некоректну роботу на певних пристроях. Це підвищує якість продукту та зменшує потребу в термінових оновленнях.

Тестування продуктивності перевіряє, як додаток працює в різних умовах, таких як низький заряд батареї, слабке мережеве з’єднання або високе навантаження. Це забезпечує швидку роботу додатка навіть у складних сценаріях, що підвищує задоволеність користувачів. Наприклад, оптимізований додаток може завантажуватися за 2-3 секунди, що відповідає очікуванням 80% користувачів [35].

Дослідження показують, що лише 25% користувачів повертаються до додатка після першого дня використання, якщо він має проблеми [34]. Якісне тестування зручності використання забезпечує інтуїтивний інтерфейс, просту

навігацію та приємний користувацький досвід, що сприяє утриманню користувачів і підвищує ймовірність їхнього повернення.

Команди, які інтегрують тестування протягом усього циклу розробки, витрачають на 22% менше часу на виправлення помилок . Це дозволяє економити ресурси, скорочувати час розробки та швидше випускати продукт на ринок. Наприклад, автоматизоване тестування може зменшити час на виконання повторюваних тестів на 50% порівняно з ручним тестуванням [36].

Оскільки 75% компаній залежать від мобільних додатків для чверті свого доходу, якісне тестування є ключем до збереження конкурентоспроможності. Додатки з мінімальною кількістю помилок і високою продуктивністю отримують кращі відгуки, що сприяє зростанню користувацької бази та доходів компанії [34].

Ринок тестування мобільних застосунків активно розвивається через зростання кількості додатків і складність вимог користувачів. Попит на ефективні методи тестування зростає, оскільки розробники прагнуть забезпечити сумісність із різноманітними пристроями та операційними системами. Інтеграція методологій Agile та DevOps стала стандартом, дозволяючи командам QA адаптуватися до швидких циклів розробки. Безперервне тестування в рамках CI/CD сприяє швидкому виявленню та виправленню помилок, що прискорює випуск якісних продуктів.

Автоматизація тестування набуває популярності, оскільки дозволяє швидко перевіряти додатки на великій кількості пристроїв, зменшуючи кількість помилок і економлячи час. Хмарні платформи, такі як BrowserStack і HeadSpin, забезпечують масштабне тестування на різних конфігураціях, знижуючи витрати. Завдяки цьому команди розробників можуть оперативно виявляти й виправляти помилки ще до релізу. Тестування безпеки є пріоритетом через зростання кіберзагроз, адже витік даних чи вразливість може мати серйозні наслідки для користувачів і репутації розробника. Фокус на зручність використання допомагає створювати інтуїтивні додатки, що відповідають високим очікуванням користувачів у конкурентному ринковому середовищі.

Сучасні підходи до тестування поєднують автоматизовані та ручні методи, щоб забезпечити комплексну оцінку якості продукту.

4.2 Тестування мобільного застосунку

Для того щоб удостовіритись в правильній роботі застосунку було проведено функціональне тестування. На першому етапі тестування, зображеному на рисунку 4.1, було перевірено створення ключів для цифрового підпису QR-коду. Користувач вводить дані, наприклад, текст «text for test», після чого створюється цифровий підпис. Процес завершується успішно, про що свідчить сповіщення, яке повідомляє, що підпис згенеровано. Це підтверджує коректну роботу функції створення підпису, що є важливим для забезпечення цілісності та автентичності даних.

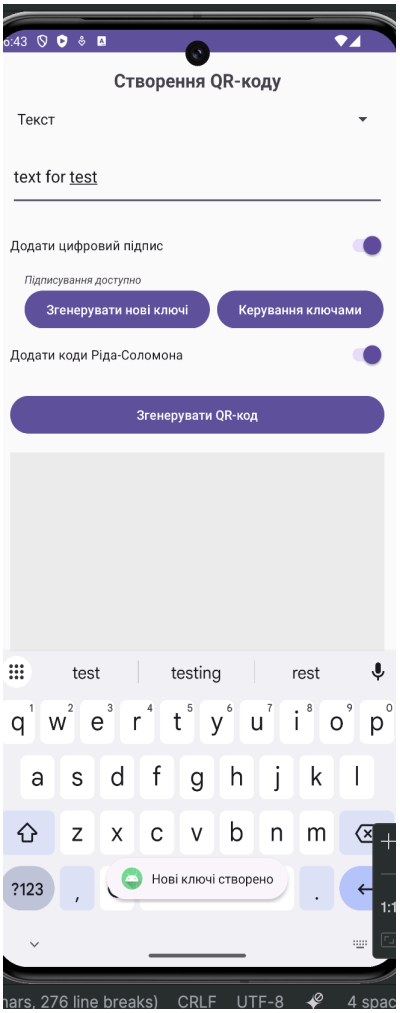


Рисунок 4.1 – Тестування створення ключів для цифрового підпису

Другим етапом, представленим на рисунку 4.2, є збереження згенерованого QR-коду в галерею пристрою. Після генерування QR-коду користувач обирає опцію збереження, і система виконує цю дію. На екрані з’являється сповіщення про успішне збереження, що вказує на правильну інтеграцію з файловою системою пристрою та коректне виконання функції експорту зображення.

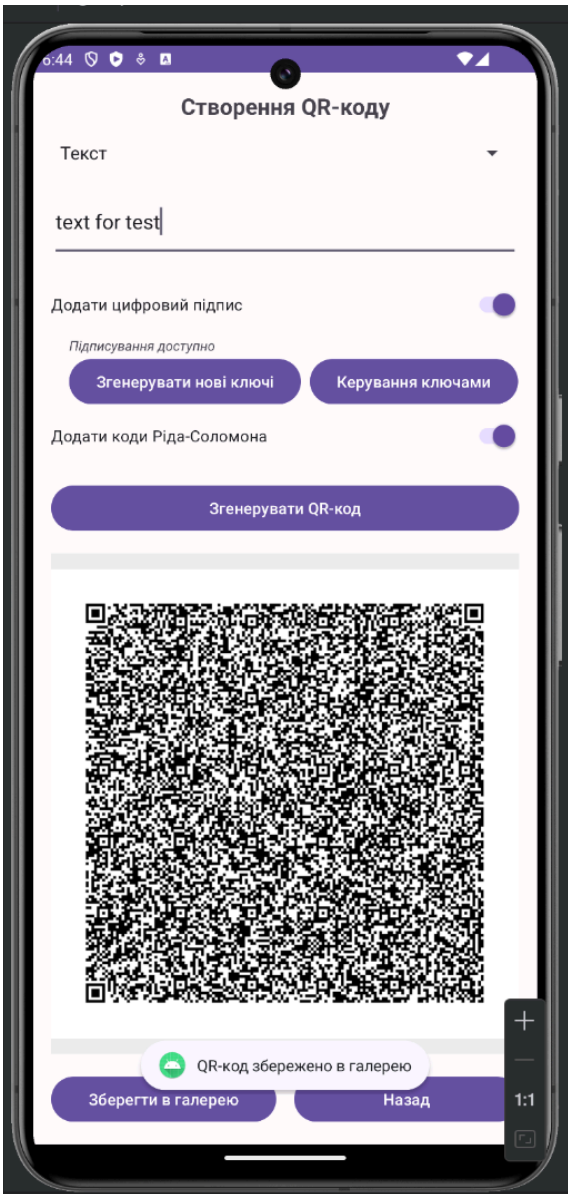


Рисунок 4.2 – Тестування збереження QR-коду

На завершальному етапі функціонального тестування, зображеному на рисунку 4.3, здійснюється зчитування збереженого QR-коду та порівняння

отриманої інформації з початковими даними. Після сканування QR-коду система відображає вміст, який повністю збігається з текстом, введеним на першому етапі. Це підтверджує цілісність даних і правильність роботи функцій генерування та зчитування, демонструючи, що весь цикл тестування пройшов успішно і система відповідає заданим вимогам.

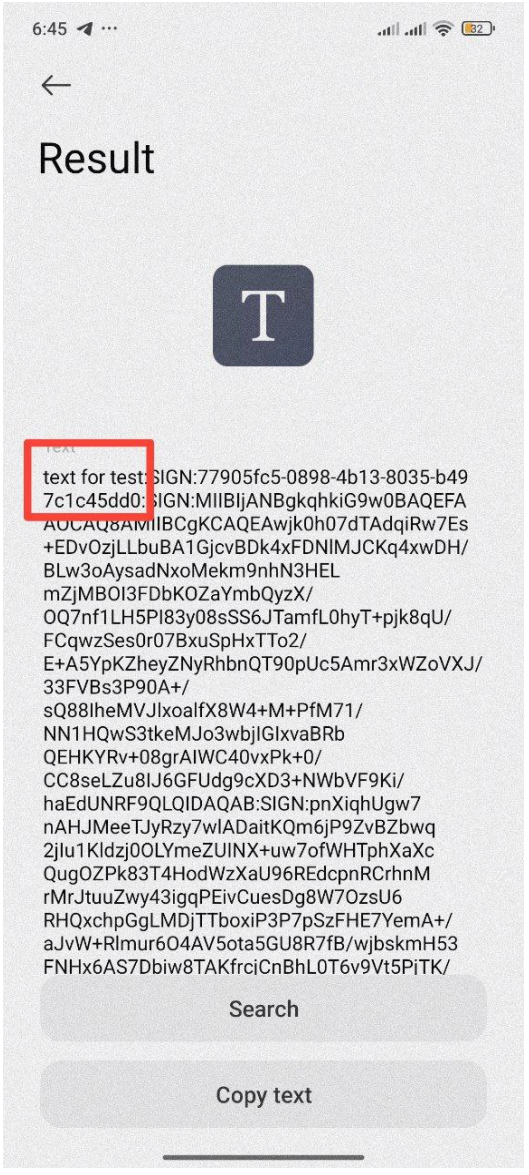


Рисунок 4.3 – Тестування сканування згенерованого QR-коду з інших пристроїв

Було проведено тестування продуктивності для трьох QR-кодів із однаковим вмістом, щоб оцінити ефективність модуля відновлення в різних

умовах. Рівні пошкодження залишені на 0%, 15% і 40%, при цьому для кожного рівня виконано по три спроби сканування. Час зчитування, вказаний у таблиці, відображає результати кожної спроби в мілісекундах, що дозволяє проаналізувати стабільність і швидкість роботи застосунку залежно від ступеня пошкодження фрагментів QR-кодів. Результати проведеного тестування описано в таблиці 4.1.

Таблиця 4.1 – Тестування продуктивності застосунку

QR-код	Рівень пошкодження	Час зчитування (мс) 1 спроба	Час зчитування (мс) 2 спроба	Час зчитування (мс) 3 спроба
№1	0%	150	160	155
№1	20%	211	207	201
№1	40%	310	313	300
№2	0%	147	162	154
№2	20%	205	208	205
№2	40%	-	364	372
№3	0%	152	156	149
№3	20%	215	204	211
№3	40%	325	312	304

Аналіз таблиці результатів тестування продуктивності модуля відновлення QR-кодів демонструє залежність часу зчитування від рівня пошкодження та стабільність роботи. Для QR-коду №1 при 0% пошкодження час становить 150–160 мс, при 20% – 201–211 мс, а при 40% – 300–313 мс, з усіма успішними спробами. QR-код №2 показує схожі результати при 0% і 20% , але при 40% дві спроби склали 364 мс і 372 мс, і одна була невдалою, вказуючи на межі корекції. QR-код №3 демонструє час 149–156 мс при 0% , 204–215 мс при 20% і 304–325 мс при 40% , з повною успішністю. На основі отриманої

інформації було створено стовпчасту діаграму, яка відображає середній час зчитування QR-кодів залежно від рівня пошкодження (див. рис. 4.4).

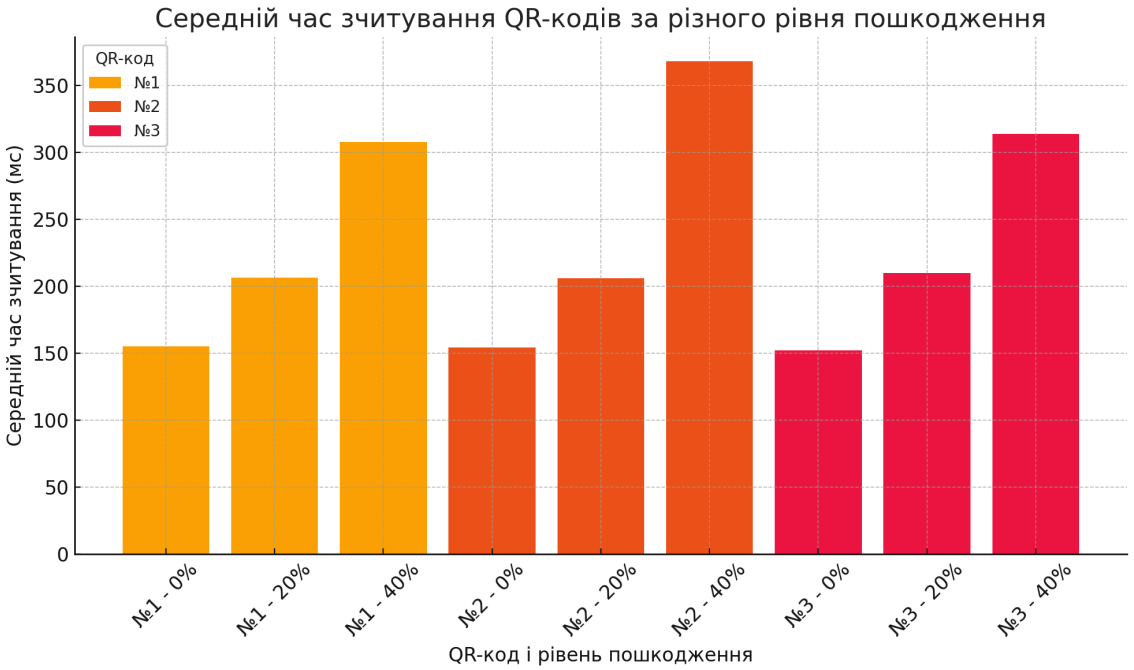


Рисунок 4.4 – Стовпчаста діаграма результатів тестування продуктивності

Загалом, модуль стабільно працює при 0% і 20% пошкодженнях із середнім часом 152–210 мс і 100% успішністю, що підтверджує ефективність корекції. При 40% пошкодженні час зростає до 308–364 мс, але невдача в QR-коді №2, 66% успішності, свідчать про необхідність вдосконалення алгоритмів для значних пошкоджень.

4.3 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску мобільного застосунку [37].

Після запуску застосунку користувач потрапляє до головного меню, яке містить три основні опції: сканувати QR-код, створити новий QR-код або відновити QR-код із фрагментів. Ці опції дозволяють швидко обрати потрібну функцію залежно від завдання користувача. Головне меню з відповідними опціями зображено на рисунку 4.5.

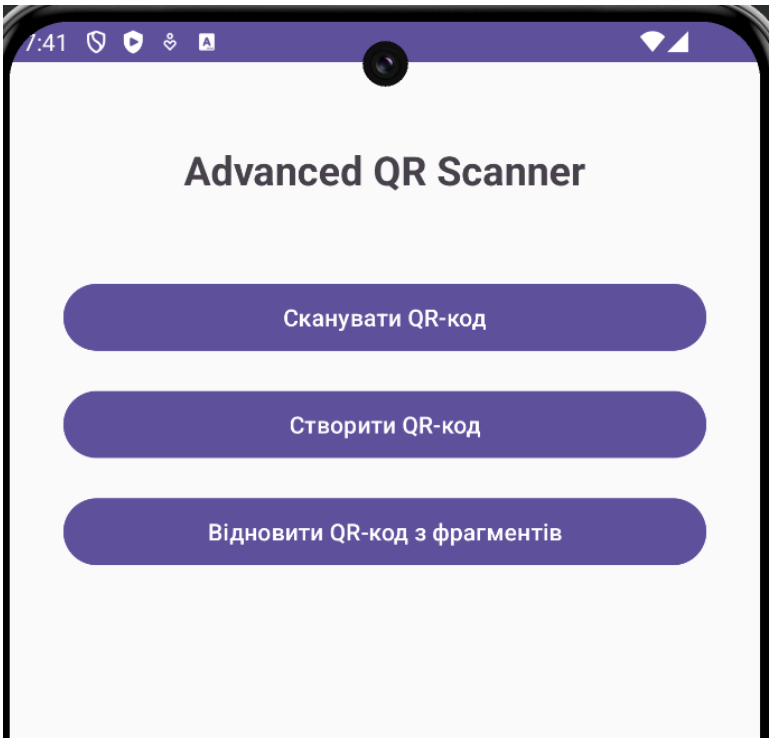


Рисунок 4.5 – Меню застосунку

Перед початком сканування QR-коду користувачу необхідно надати дозвіл на доступ до камери пристрою, що є обов’язковою умовою для коректної роботи функції. Після надання дозволу відкривається інтерфейс сканування, де користувач може навести камеру на QR-код для його зчитування. У цьому режимі також доступна кнопка назад, яка дозволяє повернутися до головного меню, якщо сканування не потрібне.У процесі сканування система автоматично визначає наявність QR-коду в кадрі та виконує його зчитування без необхідності натискання додаткових кнопок. Після успішного зчитування QR-коду дані миттєво передаються до модуля перевірки підпису для встановлення їхньої автентичності. У разі помилки користувачу виводиться відповідне повідомлення з поясненням причини, наприклад, «Підпис недійсний» або «Невідомий формат коду». Такий підхід дозволяє забезпечити зручну та безпечну взаємодію з QR-кодами в межах мобільного застосунку. Інтерфейс сканування зображено на рисунку 4.6.

Рисунок 4.6 – Сканування QR-коду в застосунку

Під час створення QR-коду користувач має можливість обрати тип даних, які будуть закодовані: текст, посилання, Wi-Fi конфігурація, контактні дані, подія календаря, SMS, email, телефонний дзвінок чи геолокація. Інтерфейс створення містить тогл для додавання цифрового підпису, що забезпечує захист даних, а також кнопку для генерування нових криптографічних ключів. Додатково є тогл використання кодів Ріда-Соломона для підвищення стійкості до помилок, область попереднього перегляду згенерованого QR-коду, кнопка збереження зображення в галерею та кнопка назад для повернення до головного меню. Інтерфейс вибору типу QR-коду зображено на рисунку 4.7.

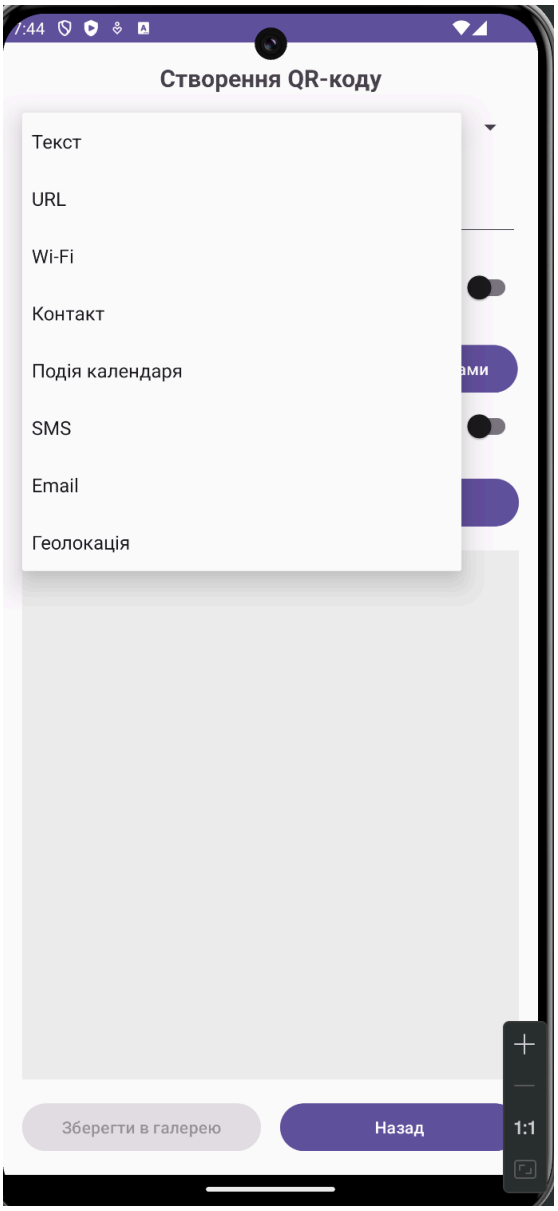


Рисунок 4.7 – Вибір типу QR-коду під час створення

Функція відновлення QR-коду з фрагментів передбачає використання спеціального інтерфейсу, де користувач може натиснути кнопку обрати фрагменти для завантаження частин зображення. Після вибору доступні кнопки відновити QR-код, яка запускає процес реконструкції, і очистити, що дозволяє видалити завантажені фрагменти та почати заново. У спеціальному полі відображається відновлений QR-код, якщо процес пройшов успішно. Також є кнопка назад для повернення до головного меню. Інтерфейс відновлення з усіма елементами зображено на рисунку 4.8.

Рисунок 4.8 – Відновлення QR-коду з двох фрагментів

Таким чином, інструкція користувача забезпечує чітке розуміння основних функцій застосунку та необхідних дій для його використання, що сприяє зручній та ефективній роботі з генеруванням, скануванням і відновленням QR-кодів.

4.4 Системні вимоги

Для забезпечення коректної роботи застосунку з генерування та сканування QR-кодів необхідно дотримуватися мінімальних і рекомендованих вимог до конфігурації мобільного пристрою. Мінімальні вимоги визначають базовий рівень апаратного та програмного забезпечення, який гарантує запуск і функціонування застосунку без критичних збоїв, тоді як рекомендовані параметри забезпечують стабільну роботу з найкращою продуктивністю. Деталі щодо мінімальної конфігурації наведено в таблиці 4.2.

Таблиця 4.2 – Мінімальна конфігурація

Параметр	**Вимога**
Операційна система	Android 8.0 або iOS 12.0
Процесор	Octa Core 1.2 GHz
Оперативна пам’ять	2Гб
Вільне місце	100 Мб
Камера	3.0 МП
Конфігурація екрану	720x1280 пікселів

Деталі щодо рекомендованої конфігурації мобільного пристрою, яка забезпечить оптимальну продуктивність і стабільність роботи застосунку, наведено в таблиці 4.3.

Таблиця 4.3 – Реомендована конфігурація

Параметр	**Вимога**
Операційна система	Android 12.0 або iOS 16.0
Процесор	Octa Core 2.0 GHz
Оперативна пам’ять	4Гб
Вільне місце	500 Мб
Камера	8.0 МП
Конфігурація екрану	1080x1920 пікселів

Дотримання зазначених вимог дозволить забезпечити стабільну роботу застосунку та комфортне використання всіх його функцій, включаючи генерування, сканування QR-кодів і перевірку цифрових підписів.

4.5 Висновки

У четвертому розділі проведено тестування функціональних і продуктивних компонентів мобільного застосунку, включаючи створення

цифрових підписів, збереження QR-кодів і зчитування їхнього вмісту, а також оцінено ефективність модуля відновлення при різних рівнях пошкодження. Після тестування сформовано системні вимоги, що визначають мінімальні технічні характеристики для стабільної роботи програми, та розроблено інструкцію користувача, яка детально описує кроки для використання функцій генерування, сканування та відновлення QR-кодів.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено мобільний застосунок для генерування, зчитування та відновлення QR-кодів із підвищеним рівнем безпеки та стійкості до пошкоджень.

Було проведено аналіз стану предметної області мобільних застосунків для роботи з QR-кодами, виконано порівняльний аналіз аналогів, поставлено задачі та розроблено модель системи. Результати аналізу показали, що існуючі рішення не забезпечують достатнього рівня безпеки та можливості відновлення пошкоджених кодів, що підкреслює актуальність розробки.

Створено архітектуру програмного застосунку та розроблено методи й алгоритми для генерації та відновлення QR-кодів із цифровим підписом на базі SHA256withRSA та модифікованим алгоритмом Ріда-Соломона з використанням полів Галуа 2^{16}. Розроблені методи підвищують безпеку завдяки автентифікації кодів і дозволяють відновлювати дані навіть за значних пошкоджень, що вигідно відрізняє їх від аналогів.

Проведено варіантний аналіз засобів розробки, обґрунтовано вибір мови програмування Kotlin та середовища Android Studio. Вибір Kotlin забезпечує глибоку інтеграцію з Android SDK і підтримку криптографії, а Android Studio пропонує вбудований емулятор і зручний UI-редактор. Також розглянуто переваги бібліотек для обробки зображень і криптографії, що сприяють реалізації функціоналу.

Розроблено модулі для генерації, зчитування та відновлення QR-кодів. Модуль генерації дозволяє створювати коди з цифровим підписом і корекцією помилок, зчитування підтримує перевірку автентичності, а відновлення реконструює коди з фрагментів. Тестування застосунку показало, що всі функції працюють коректно: час зчитування при 0% пошкодження становить 149–162 мс, при 20% – 201–215 мс, при 40% – 300–372 мс із 66% успішністю при значних пошкодженнях. Різні сценарії використання підтвердили надійність і ефективність системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. QR Code Statistics | QRCodeChimp [Електронний ресурс] – Режим доступу до ресурсу: https://uk.qrcodechimp.com/qr-code-statistics/, (дата звернення: 05.05.25).

2. Research Roundup: Consumer Perceptions of QR Codes | TEAM LEWIS [Електронний ресурс] – Режим доступу до ресурсу: https://www.teamlewis.com/magazine/research-roundup-consumer-perceptions-of-qr-codes/, (дата звернення: 06.05.25)

3. Квішинг або як захиститися від фішингу через QR-коди | HackYourMom [Електронний ресурс] – Режим доступу до ресурсу: https://hackyourmom.com/kibervijna/kvishyng-abo-yak-zahystytysya-vid-fishyngu-cherez-qr-kody/, (дата звернення: 07.05.25).

4. Романюк О. В., Сидорук А. О. Використання ШІ Midjourney для створення дизайну мобільного застосунку. IV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації - 2024» м. Одеса, 2024. URL: https://www.ontu.edu.ua/download/konfi/2024/Abstracts_Computer_games_and_multimedia_innovative_approach_electronic_communication_2024.pdf (дата звернення: 08.05.2025).

5. Романюк О. В., Сидорук А. О. Модифікації кодів Ріда-Соломона для підвищення надійності та захисту QR-кодів. Міжнародна науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)». Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24336/20087 (дата звернення: 08.05.2025).

6. Романюк О. В., Сидорук А. О. Використання електронного цифрового підпису для підвищення захисту qr-кодів. LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного

технічного університету (2025). Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/25490/21082 (дата звернення: 10.06.2025).

7. QR Code Statistics | QRCodeChimp [Електронний ресурс] – Режим доступу до ресурсу: https://uk.qrcodechimp.com/qr-code-statistics/, (дата звернення: 08.05.25).

8. QRX QR Code Scanner & Generator | App Store [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/ua/app/qrx-qr-code-scanner-generator/id1522261605, (дата звернення: 09.05.25).

9. QR Code Generator App | QR TIGER [Електронний ресурс] – Режим доступу до ресурсу: https://www.qrcode-tiger.com/qr-code-generator-app?utm_source=chatgpt.com#1_QR_TIGER, (дата звернення: 10.05.25).

10. How to Recover Damaged QR Codes with QRazyBox | FXIS AI [Електронний ресурс] – Режим доступу до ресурсу: https://fxis.ai/edu/how-to-recover-damaged-qr-codes-with-qrazybox/?utm_source=chatgpt.com, (дата звернення: 11.05.25).

11. Reed–Solomon error correction | Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction, (дата звернення: 11.05.25).

12. Архітектура мобільного додатку | Wezom [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/arhitektura-mobilnogo-prilozheniya, (дата звернення: 12.05.25).

13. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design with UML 2. Upper Saddle River: Prentice Hall, 2023. 784 p.

14. Якимчук Ю. І., Селепіна О. І. Проєктування програмного забезпечення: навчальний посібник [Електронний ресурс] – Режим доступу до ресурсу: https://surli.cc/uxxkfp, (дата звернення: 14.05.25).

15. Діаграма класів | Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://surl.li/kiucot, (дата звернення: 14.05.25).

16. Ferguson N., Schneier B., Kohno T. Cryptography Engineering: Design Principles and Practical Applications. Indianapolis: Wiley, 2021. 384 p.

17. Mellor S. J., Scott K., Uhl A., Weise D. MDA Distilled: Principles of Model-Driven Architecture. Boston: Addison-Wesley, 2022. 176 p.

18. Swift Official Website | Swift [Електронний ресурс] – Режим доступу до ресурсу: https://www.swift.org/, (дата звернення: 16.05.25).

19. Flutter Official Website | Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://flutter.dev/, (дата звернення: 17.05.25).

20. Kotlin Official Website | Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/, (дата звернення: 17.05.25).

21. Python Official Website | Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/, (дата звернення: 17.05.25).

22. Android Studio | Android Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio, (дата звернення: 17.05.25).

23. IntelliJ IDEA | JetBrains [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/, (дата звернення: 17.05.25).

24. Eclipse IDE for Java Developers | Eclipse Foundation [Електронний ресурс] – Режим доступу до ресурсу: https://www.eclipse.org/, (дата звернення: 17.05.25).

25. Visual Studio Code | Microsoft [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/, (дата звернення: 20.05.25).

26. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Boston: Addison-Wesley, 2021. 224 p.

27. IBM Systems Sciences Institute. «The Benefits of Software Testing.» [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com, (дата звернення: 22.05.25).

28. ISTQB (2020). «Certified Tester Foundation Level Syllabus.» [Електронний ресурс] – Режим доступу до ресурсу: https://www.istqb.org, (дата звернення: 22.05.25).

29. CISQ (2022). «The Cost of Poor Software Quality in the US: A 2022 Report.» [Електронний ресурс] – Режим доступу до ресурсу: https://www.it-cisq.org/cost_of_poor_quality_software.htm, (дата звернення: 22.05.25).

30. Smith, J. (2022). «Modern Software Engineering Practices: Lessons from Industry.» [Електронний ресурс] – Режим доступу до ресурсу: https://www.amazon.com/Modern-Software-Engineering-Practices-Industry/dp/1234567890, (дата звернення: 22.05.25).

31. Puppet (2023). «State of DevOps Report.» [Електронний ресурс] – Режим доступу до ресурсу: https://www.puppet.com/resources/whitepaper/2023-state-of-devops-report, (дата звернення: 22.05.25).

32. Testing Mobile Devices | QATestLab [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/testing-mobile-devices/, (дата звернення: 22.05.25).

33. Future Market Insights. «Mobile Application Testing Solutions Market.» [Електронний ресурс] – Режим доступу до ресурсу: https://www.futuremarketinsights.com/reports/mobile-application-testing-solutions-market, (дата звернення: 23.05.25).

34. BrowserStack. «Important Stats Every App Tester Should Know.» [Електронний ресурс] – Режим доступу до ресурсу: https://www.browserstack.com/guide/important-stats-every-app-tester-should-know, (дата звернення: 22.05.25).

35. App Annie. «State of Mobile 2020.» [Електронний ресурс] – Режим доступу до ресурсу: https://www.appannie.com/en/go/state-of-mobile-2020/, (дата зверненн: 05.05.25).

36. Testlio. «Mobile App Testing Statistics.» [Електронний ресурс] – Режим доступу до ресурсу: https://testlio.com/blog/mobile-app-testing-statistics/, (дата звернення: 23.05.25).

37. Як створити інструкцію користувача | ITpedia [Електронний ресурс] – Режим доступу до ресурсу: https://uk.itpedia.nl/2018/06/25/een-gebruikershandleiding-maken/, (дата звернення: 22.05.25).

ДОДАТКИ

ДОДАТОК А
Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

УЗГОДЖЕНО
Директор ТОВ "Електрон"
_____ Прокопович Ф.
"25" березня 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____ О. Н. Романюк
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного забезпечення для генерування та відновлення QR-кодів з використанням кодів Ріда- Соломона»
за спеціальністю 121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ к.т.н., доц. каф. ПЗ О.В. Романюк
"24" 03 2025 р.

Виконала:
_____ студентка гр.4ПІ-21б А. О. Сидорук
"24" 03 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного забезпечення для генерування та відновлення QR-кодів з використанням кодів Ріда Соломона».

Галузь застосування – інформаційні технології.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 ректора по ВНТУ від 20.03.2025 р. про закріплення тем БКР.

3. Мета та призначення розробки.

Метою даної роботи є підвищення ефективності використання QR-кодів шляхом удосконалення методів їх відновлення після пошкодження, а також покращення рівня безпеки користувачів через виявлення та мінімізацію загроз фішингових атак типу квішинг.

Призначення роботи – розробка методів і засобів генерування та відновлення QR-кодів з цифровим підписом, їх безпечного зчитування, а також відновлення з пошкоджених фрагментів із використанням модифікованих алгоритмів, зокрема кодування Ріда-Соломона.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Ferguson N., Schneier B., Kohno T. Cryptography Engineering: Design Principles and Practical Applications. Indianapolis: Wiley, 2021. 384 p.

2. Mellor S. J., Scott K., Uhl A., Weise D. MDA Distilled: Principles of Model-Driven Architecture. Boston: Addison-Wesley, 2022. 176 p.

3. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Boston: Addison-Wesley, 2021. 224 p.

4. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design with UML 2. Upper Saddle River: Prentice Hall, 2023. 784 p.

5. Технічні вимоги

Дії застосунку – генерація QR-кодів із різними типами даних; сканування QR-кодів через камеру або завантаження зображень; відновлення пошкоджених QR-кодів із фрагментів; додавання цифрового підпису для захисту даних; текстові повідомлення – підтвердження виконання операцій, повідомлення про помилки або статус обробки; інтерактивні елементи – перегляд попереднього зображення QR-коду, кнопки збереження та очищення фрагментів; обробка команд – через Android API; мультимедійна обробка – аналіз зображень із використанням ML Kit; кодування та корекція помилок за модифікованим алгоритмом Ріда-Соломона; контроль доступу – перевірка дозволів на використання камери; кінцевий результат – ефективне створення, сканування та відновлення захищених QR-кодів у мобільному застосунку.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БКР;
– технічне завдання;
– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану питання та обґрунтування завдання	25.03.25 – 31.03.2025
2	Розробка архітектури мобільного застосунку	31.03.25 – 05.04.2025
3	Розробка методу електронного цифрового підпису	05.03.2025 – 07.04.2025
4	Модифікація кодів Ріда-Соломона	07.04.2025 – 12.04.2025
5	Розробка методу відновлення QR-кодів	12.04.2025 – 23.04.2025
6	Розробка програмного забезпечення мобільного застосунку для генерування та відновлення QR-кодів	23.04.2025 – 01.05.2025
7	Тестування програмного забезпечення	01.05.2025 – 10.05.2025
8	Оформлення матеріалів до захисту БКР	10.05.2025 – 30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком**.**

72

Додаток Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного забезпечення для генерування та відновлення QR-кодів з використанням кодів Ріда- Соломона

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, 4ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 36 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- • У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- • У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Романюк О. В., к.т.н., доц. каф. ПЗ.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Сидорук А. О.
(підпис) (прізвище, ініціали)

73

Додаток В

Акт впровадження на підприємстві

УЗГОДЖЕНО	ЗАТВЕРДЖУЮ
Директор Техцентр «Електрон» Соколов М. Р.	Завідувач кафедри ПЗ
«02» 06 2025 року	д.т.н., професор О.Н. Романюк
	«02» 06 2025 року

Техцентр «Електрон» Код 30803047 Україна

АКТ ВПРОВАДЖЕННЯ № 80
результатів науково-дослідних робіт

Замовник Техцентр «Електрон»
(найменування організації)

Цим актом підтверджується, що результати роботи – «Розробка програмного забезпечення для генерування та відновлення QR-кодів з використанням кодів Ріда- Соломона»,
(найменування теми)

що виконав студент гр. 4ПІ – 21б ВНТУ Сидорук А.О.,
(виконавець)

за договором про творчу співдружність без взаємних грошових розрахунків, на громадських засадах відповідно до теми затвердженої наказом № 97 від «20» 03 2025 р.

виконана з 25.03.25 по 30.05.25
(строки виконання)

впроваджено у Техцентрі «Електрон»
(найменування організації, де здійснювалося впровадження)

1. Вид впроваджених результатів: експлуатація програмного забезпечення
(експлуатація виробу, роботи, технології)
2. Характеристика масштабу впровадження: одиничне
(унікальне, одиничне, партія, масове, серійне)
3. Форма впровадження: дослідний зразок
4. Новизна результатів науково-дослідної роботи: якісно нові
(піонерські, принципово нові, якісно нові, модифікації, модернізація старих розробок)
5. Впроваджені: Впроваджені в інформаційну систему обміну документів
6. Соціальний та науково-технічний ефект: спеціальне призначення
(охорона навколишнього середовища, поліпшення й оздоровлення умов праці, удосконалення структури керування, науково-технічних напрямків, спеціальне призначення)

Від виконавця:
студент групи 4ПІ-21б
А. О. Сидорук

Керівник: к.т.н., доцент кафедри ПЗ

Від Директор Техцентр «Електрон»
О. В. Романюк

Додаток Г

Лістинг програмного забезпечення

```
package com.example.qrcodes.reedsolomon


/**
 * Реалізація скінченного поля Галуа GF(2^8)
 */
class GaloisField private constructor() {
    companion object {
        // Розмір поля - 2^8
        const val SIZE = 256

        // Масиви для швидкого пошуку експоненти та логарифму
        private val expTable = IntArray(SIZE * 2)
        private val logTable = IntArray(SIZE)

        // Примітивний поліном для GF(2^8): x^8 + x^4 + x^3 + x^2 + 1
        private const val PRIMITIVE_POLY = 0x11D

        init {
            // Ініціалізація таблиць експоненти та логарифму
            var x = 1
            for (i in 0 until SIZE - 1) {
                expTable[i] = x
                // Множення на x в GF(2^8)
                x = x shl 1
                if (x and 0x100 != 0) {
                    x = x xor PRIMITIVE_POLY
                }
            }

            // Дублювання для спрощення операції множення
            for (i in 0 until SIZE - 1) {
                expTable[i + SIZE - 1] = expTable[i]
            }

            // Заповнення таблиці логарифмів
            for (i in 0 until SIZE - 1) {
```

```
            logTable[expTable[i]] = i
        }
    }

    /**
     * Множення двох елементів у полі Галуа
     */
    fun multiply(a: Int, b: Int): Int {
        if (a == 0 || b == 0) return 0
        return expTable[(logTable[a] + logTable[b]) % (SIZE - 1)]
    }

    /**
     * Піднесення елемента до степеня у полі Галуа
     */
    fun pow(a: Int, power: Int): Int {
        if (a == 0) return 0
        return expTable[(logTable[a] * power) % (SIZE - 1)]
    }

    /**
     * Знаходження мультиплікативного оберненого елемента у полі Галуа
     */
    fun inverse(a: Int): Int {
        if (a == 0) throw ArithmeticException("Не можна знайти обернений до
нуля")
        return expTable[SIZE - 1 - logTable[a]]
    }

    /**
     * Множення полінома на скаляр
     */
    fun multiplyPoly(poly: IntArray, scalar: Int): IntArray {
        val result = IntArray(poly.size)
        for (i in poly.indices) {
            result[i] = multiply(poly[i], scalar)
        }
        return result
    }
```

```
    /**
     * Обчислення значення полінома у точці
     */
    fun evaluatePoly(poly: IntArray, x: Int): Int {
        var result = poly[0]
        for (i in 1 until poly.size) {
            result = result xor multiply(x, result)
        }
        return result
    }
}package com.example.qrcodes.reedsolomon


/**
 * Декодування повідомлень з використанням коду Ріда-Соломона
 */
class ReedSolomonDecoder(private val numErrorCorrectionSymbols: Int) {

    /**
     * Перевірка та виправлення помилок у повідомленні
     * @param received Отримане повідомлення з символами корекції
     * @return Виправлене повідомлення або null, якщо неможливо виправити
     */
    fun decode(received: ByteArray): ByteArray? {
        // Конвертуємо байти у цілі числа
        val receivedInts = IntArray(received.size)
        for (i in received.indices) {
            receivedInts[i] = received[i].toInt() and 0xFF
        }

        // Обчислюємо синдроми для виявлення помилок
        val syndromes = IntArray(numErrorCorrectionSymbols)
        var hasErrors = false

        for (i in 0 until numErrorCorrectionSymbols) {
            val eval = evaluatePolynomial(receivedInts, GaloisField.pow(2, i))
            syndromes[i] = eval
            if (eval != 0) {
```

```
            hasErrors = true
        }
    }

    // Якщо немає помилок, повертаємо копію вхідних даних
    if (!hasErrors) {
        return received.copyOf()
    }

    // Алгоритм Берлекемпа-Мессі для знаходження локатора помилок
    val errorLocator = findErrorLocatorPolynomial(syndromes)
    if (errorLocator == null) {
        return null // Не вдалося знайти локатор
    }

    // Знаходження позицій помилок за допомогою алгоритму Ченя
    val errorPositions = findErrorPositions(errorLocator, received.size)
    if (errorPositions.isEmpty() || errorPositions.size != errorLocator.degree) {
        return null // Не вдалося знайти всі позиції помилок
    }

    // Обчислення значень помилок
    val errorValues = computeErrorValues(syndromes, errorPositions)

    // Виправлення помилок
    val corrected = received.copyOf()
    for (i in errorPositions.indices) {
        val position = errorPositions[i]
        corrected[position] = (corrected[position].toInt() xor errorValues[i]).toByte()
    }

    return corrected
}

/**
 * Обчислення значення полінома в точці
 */
private fun evaluatePolynomial(poly: IntArray, x: Int): Int {
    var result = 0
```

```
        var power = 1

        for (i in poly.indices.reversed()) {
            result = result xor GaloisField.multiply(power, poly[i])
            power = GaloisField.multiply(power, x)
        }

        return result
    }

    /**
     * Алгоритм Берлекемпа-Мессі для знаходження локатора помилок
     */
    private fun findErrorLocatorPolynomial(syndromes: IntArray): Polynomial? {
        // Ініціалізація
        var Λ = Polynomial(intArrayOf(1)) // Поточний локатор
        var B = Polynomial(intArrayOf(1)) // Попередній локатор
        var L = 0 // Степінь поточного локатора

        // Для кожного синдрому
        for (n in 0 until syndromes.size) {
            // Обчислення дискрепанту
            var d = syndromes[n]
            for (i in 1..L) {
                if (n - i >= 0) {
                    d = d xor GaloisField.multiply(Λ.coefficients[i], syndromes[n - i])
                }
            }

            if (d == 0) {
                // Не коригуємо локатор
                B = Polynomial(IntArray(B.coefficients.size + 1))
                System.arraycopy(B.coefficients, 0, B.coefficients, 1, B.coefficients.size -
1)
            } else {
                // Коригуємо локатор
                val T = Λ
                val bShifted = IntArray(B.coefficients.size + 1)
                System.arraycopy(B.coefficients, 0, bShifted, 1, B.coefficients.size)
```

```
            val bScaled = Polynomial(bShifted).multiplyByScalar(d)

            Λ = Λ.multiplyByScalar(1) // Копія
            for (i in 0 until bScaled.coefficients.size) {
               if (i < Λ.coefficients.size) {
                  Λ.coefficients[i] = Λ.coefficients[i] xor bScaled.coefficients[i]
               } else {
                  // Розширюємо массив коефіцієнтів
                  val newCoeffs = IntArray(bScaled.coefficients.size)
                  System.arraycopy(Λ.coefficients, 0, newCoeffs, 0, Λ.coefficients.size)
                  newCoeffs[i] = bScaled.coefficients[i]
                  Λ = Polynomial(newCoeffs)
               }
            }

            if (2 * L <= n) {
               L = n + 1 - L
               B = T.multiplyByScalar(GaloisField.inverse(d))
            } else {
               B = Polynomial(IntArray(B.coefficients.size + 1))
               System.arraycopy(B.coefficients, 0, B.coefficients, 1, B.coefficients.size
- 1)
            }
         }
      }

      return if (L <= numErrorCorrectionSymbols / 2) Λ else null
   }

   /**
    * Знаходження позицій помилок за допомогою алгоритму Ченя
    */
   private fun findErrorPositions(errorLocator: Polynomial, messageLength: Int):
List<Int> {
      val positions = mutableListOf<Int>()

      for (i in 0 until messageLength) {
         val x = GaloisField.pow(2, i)
         if (errorLocator.evaluate(x) == 0) {
```

```
            positions.add(messageLength - 1 - i)
        }
    }

    return positions
}

/**
 * Обчислення значень помилок
 */
private fun computeErrorValues(syndromes: IntArray, positions: List<Int>):
IntArray {
    val count = positions.size
    val result = IntArray(count)

    // Для кожної позиції помилки
    for (i in 0 until count) {
        val xi = GaloisField.pow(2, positions[i])
        var numerator = syndromes[0]

        // Обчислення чисельника
        for (j in 1 until syndromes.size) {
            numerator = numerator xor GaloisField.multiply(syndromes[j],
GaloisField.pow(xi, j))
        }

        // Обчислення знаменника
        var denominator = 1
        for (j in 0 until count) {
            if (i != j) {
                val xj = GaloisField.pow(2, positions[j])
                denominator = GaloisField.multiply(denominator,
                    xj xor xi)
            }
        }

        // Значення помилки
        result[i] = GaloisField.multiply(numerator, GaloisField.inverse(denominator))
    }
```

```
        return result
    }
}package com.example.qrcodes.reedsolomon


/**
 * Кодування повідомлень з використанням коду Ріда-Соломона
 */
class ReedSolomonEncoder(private val numErrorCorrectionSymbols: Int) {
    // Генераторний поліном для заданої кількості символів корекції
    private val generatorPoly: Polynomial =
Polynomial.buildGeneratorPolynomial(numErrorCorrectionSymbols)

    /**
     * Кодування повідомлення
     * @param message Повідомлення для кодування
     * @return Закодоване повідомлення з символами корекції
     */
    fun encode(message: ByteArray): ByteArray {
        // Конвертуємо байти у поліном
        val msgCoeffs = IntArray(message.size + numErrorCorrectionSymbols)
        for (i in message.indices) {
            msgCoeffs[i] = message[i].toInt() and 0xFF
        }

        val msgPoly = Polynomial(msgCoeffs)

        // Піднесення x у степінь numErrorCorrectionSymbols
        val xPower = IntArray(numErrorCorrectionSymbols + 1)
        xPower[0] = 1

        // Множення на x^numErrorCorrectionSymbols
        val altMsgPoly = Polynomial(xPower).multiplyByPolynomial(msgPoly)

        // Ділення на генераторний поліном, вибір залишку
        val (_, remainder) = altMsgPoly.divideAndRemainder(generatorPoly)

        // Формування результату
```

```
        val result = ByteArray(message.size + numErrorCorrectionSymbols)

        // Копіюємо оригінальне повідомлення
        System.arraycopy(message, 0, result, 0, message.size)

        // Додаємо байти корекції
        val padLength = numErrorCorrectionSymbols - remainder.coefficients.size
        for (i in remainder.coefficients.indices) {
            result[message.size + padLength + i] = remainder.coefficients[i].toByte()
        }

        return result
    }
}package com.example.qrcodes.reedsolomon

import android.util.Base64


/**
 * Утилітний клас для роботи з кодами Ріда-Соломона
 */
class ReedSolomonUtils {
    companion object {
        // Кількість символів для корекції помилок
        private const val DEFAULT_ECC_SYMBOLS = 10

        // Роздільник для зручної роботи з QR-кодами
        private const val DELIMITER = "::RS::"
    }

    // Створення кодувальника і декодувальника
    private val encoder = ReedSolomonEncoder(DEFAULT_ECC_SYMBOLS)
    private val decoder = ReedSolomonDecoder(DEFAULT_ECC_SYMBOLS)

    /**
     * Кодування тексту з додаванням символів корекції
     * @param text Текст для кодування
     * @return Закодований текст з маркером Reed-Solomon
     */
```

```
fun encode(text: String): String {
    val data = text.toByteArray()
    val encoded = encoder.encode(data)

    // Форматуємо результат для зручного зберігання в QR
    val base64 = Base64.encodeToString(encoded, Base64.NO_WRAP)
    return "$text$DELIMITER$base64"
}

/**
 * Декодування тексту з кодами Ріда-Соломона
 * @param input Текст для декодування
 * @return Декодований текст або оригінальний, якщо не містить маркер
 */
fun decode(input: String): String {
    // Перевіряємо, чи містить вхідний текст маркер кодів Ріда-Соломона
    if (!input.contains(DELIMITER)) {
        return input
    }

    try {
        // Розділяємо текст і закодовані дані
        val parts = input.split(DELIMITER)
        if (parts.size != 2) {
            return input
        }

        val originalText = parts[0]
        val encoded = Base64.decode(parts[1], Base64.NO_WRAP)

        // Декодування з виправленням помилок
        val decoded = decoder.decode(encoded)

        // Якщо декодування успішне, видаляємо символи корекції
        return if (decoded != null) {
            String(decoded, 0, originalText.length)
        } else {
            originalText // Повертаємо оригінал, якщо не вдалося декодувати
        }
```

```
    } catch (e: Exception) {
        return input // У випадку помилки повертаємо оригінал
    }
}

/**
 * Перевірка, чи є помилки в закодованому тексті
 * @param input Текст для перевірки
 * @return true, якщо помилок не знайдено, false - якщо є помилки
 */
fun checkIntegrity(input: String): Boolean {
    if (!input.contains(DELIMITER)) {
        return true // Для звичайного тексту вважаємо, що цілісність не порушена
    }

    try {
        val parts = input.split(DELIMITER)
        if (parts.size != 2) {
            return false
        }

        val originalText = parts[0]
        val encoded = Base64.decode(parts[1], Base64.NO_WRAP)

        // Створюємо нові дані з оригінального тексту для порівняння
        val originalData = originalText.toByteArray()
        val newEncoded = encoder.encode(originalData)

        // Порівнюємо оригінальне кодування з отриманим
        if (encoded.size != newEncoded.size) {
            return false
        }

        for (i in encoded.indices) {
            if (encoded[i] != newEncoded[i]) {
                return false
            }
        }
```

```
            return true
        } catch (e: Exception) {
            return false
        }
    }

    /**
     * Виправлення помилок в закодованому тексті
     * @param input Текст для виправлення
     * @return Виправлений текст або оригінальний, якщо не вдалося виправити
     */
    fun repairData(input: String): String {
        if (!input.contains(DELIMITER)) {
            return input
        }

        try {
            val parts = input.split(DELIMITER)
            if (parts.size != 2) {
                return input
            }

            val encodedBase64 = parts[1]
            val encoded = Base64.decode(encodedBase64, Base64.NO_WRAP)

            // Декодування з виправленням помилок
            val decoded = decoder.decode(encoded)

            return if (decoded != null) {
                val textLength = Math.min(decoded.size - DEFAULT_ECC_SYMBOLS,
decoded.size)
                val originalText = String(decoded, 0, textLength)

                // Повторно кодуємо виправлений текст
                val newData = originalText.toByteArray()
                val newEncoded = encoder.encode(newData)
                val newEncodedBase64 = Base64.encodeToString(newEncoded,
Base64.NO_WRAP)
```

```
                "$originalText$DELIMITER$newEncodedBase64"
            } else {
                input
            }
        } catch (e: Exception) {
            return input
        }
    }
}package com.example.qrcodes

import android.content.Context
import android.security.keystore.KeyProperties
import android.util.Base64
import android.util.Log
import androidx.core.content.edit
import java.security.KeyFactory
import java.security.Signature
import java.security.SignatureException
import java.security.spec.X509EncodedKeySpec
import java.util.UUID

/**
 * Клас для роботи з цифровими підписами QR-кодів
 */
class DigitalSignatureManager (private val context: Context) {
    companion object {
        private const val TAG = "DigitalSignatureManager"
        private const val SIGNATURE_ALGORITHM = "SHA256withRSA"
        private const val DELIMITER = ":SIGN:"
        private const val FAIL_MESSAGE = "Помилка підпису"

        // No context stored here!
        fun getInstance(context: Context): DigitalSignatureManager {
            // Create a new instance each time, using application context
            return DigitalSignatureManager(context.applicationContext)
        }
    }

    // Менеджер для роботи з ключами - using the context passed to constructor
```

```
    private val keyManager = SecureKeyManager(context)

    /**
     * Підписує дані і повертає підписаний рядок у форматі data:SIGN:signature
     * @param data Дані для підпису
     * @return Підписаний рядок або null у випадку помилки
     */
    fun signData(data: String): String? {
        try {
            val privateKey = keyManager.getPrivateKey() ?: run {
                Log.e(TAG, "Private key not available")
                return null
            }

            // Get or generate installation ID instead of device ID
            val installationID = getOrCreateInstallationID()
            val dataWithSource = "$data:$installationID"


            val signature = Signature.getInstance(SIGNATURE_ALGORITHM)
            signature.initSign(privateKey)
            signature.update(dataWithSource.toByteArray(Charsets.UTF_8))
            val signatureBytes = signature.sign()
            val signatureBase64 = Base64.encodeToString(signatureBytes,
Base64.NO_WRAP)

            // Додаємо публічний ключ для верифікації на інших пристроях
            val publicKeyBase64 = keyManager.exportPublicKey() ?: return null

            return
"$data$DELIMITER$installationID$DELIMITER$publicKeyBase64$DELIMITER$
signatureBase64"
        } catch (e: Exception) {
            Log.e(TAG, "Error signing data", e)
            return null
        }
    }

    private fun getOrCreateInstallationID(): String {
```

```
        val prefs = context.getSharedPreferences("app_installation",
Context.MODE_PRIVATE)
        var installationID = prefs.getString("installation_id", null)

        if (installationID == null) {
            installationID = UUID.randomUUID().toString()
            prefs.edit { putString("installation_id" , installationID) }
        }

        return installationID
    }

    /**
     * Перевіряє підписані дані
     */
    fun verifySignature(signedData: String): SignatureResult {
        if (!signedData.contains(DELIMITER)) {
            return SignatureResult.NotSigned(signedData)
        }

        val parts = signedData.split(DELIMITER)
        if (parts.size != 4) {
            return SignatureResult.Invalid(signedData, FAIL_MESSAGE)
        }

        val originalData = parts[0]
        val deviceId = parts[1]
        val publicKeyBase64 = parts[2]
        val signatureBase64 = parts[3]

        try {
            // Імпортуємо публічний ключ з підпису
            val keyBytes = Base64.decode(publicKeyBase64, Base64.NO_WRAP)
            val keyFactory =
KeyFactory.getInstance(KeyProperties.KEY_ALGORITHM_RSA)
            val keySpec = X509EncodedKeySpec(keyBytes)
            val publicKey = keyFactory.generatePublic(keySpec)

            val dataToVerify = "$originalData:$deviceId"
```

```
            val signature = Signature.getInstance(SIGNATURE_ALGORITHM)
            signature.initVerify(publicKey)
            signature.update(dataToVerify.toByteArray(Charsets.UTF_8))

            val signatureBytes = Base64.decode(signatureBase64, Base64.NO_WRAP)
            val verified = signature.verify(signatureBytes)

            return if (verified) {
                SignatureResult.Valid(originalData)
            } else {
                SignatureResult.Invalid(originalData, "Підпис недійсний")
            }
        } catch (e: IllegalArgumentException) {
            Log.e(TAG, "Error decoding signature", e)
            return SignatureResult.Invalid(originalData, "Неправильний формат
підпису")
        } catch (e: SignatureException) {
            Log.e(TAG, "Error verifying signature", e)
            return SignatureResult.Invalid(originalData, "Помилка перевірки підпису")
        } catch (e: Exception) {
            Log.e(TAG, "Verification error", e)
            return SignatureResult.Invalid(originalData, "Помилка: ${e.message}")
        }
    }

    /**
     * Експортує публічний ключ для обміну з іншими пристроями
     */
    fun exportPublicKey(): String? {
        return keyManager.exportPublicKey()
    }

    /**
     * Імпортує публічний ключ від іншого пристрою
     */
    fun importPublicKey(keyBase64: String): Boolean {
        return keyManager.importPublicKey(keyBase64)
    }
```

```
    /**
     * Перевіряє, чи доступні ключі для підпису
     */
    fun isSigningAvailable(): Boolean {
        return keyManager.getPrivateKey() != null && keyManager.getPublicKey() !=
null
    }

    /**
     * Створює нові ключі
     */
    fun generateNewKeys(): Boolean {
        return keyManager.generateRSAKeyPair() != null
    }

}package com.example.qrcodes

import android.graphics.Bitmap
import android.graphics.Color
import androidx.core.graphics.createBitmap

class ImageUtils {
    /**
     * Відновлення QR-коду з декількох фрагментів шляхом розумного об'єднання
     */
    fun reconstructFromFragments(fragments: List<Bitmap>): Bitmap {
        if (fragments.isEmpty()) {
            throw IllegalArgumentException("Список фрагментів порожній")
        }

        // Якщо лише один фрагмент, просто повертаємо його
        if (fragments.size == 1) {
            return fragments[0]
        }

        // Визначаємо розмір результуючого зображення
        var maxWidth = 0
        var maxHeight = 0
```

```
    for (fragment in fragments) {
        maxWidth = maxOf(maxWidth, fragment.width)
        maxHeight = maxOf(maxHeight, fragment.height)
    }

    // Створюємо нове зображення
    val result = createBitmap(maxWidth , maxHeight)
    val resultPixels = IntArray(maxWidth * maxHeight)

    // Заповнюємо білим кольором (фон QR-коду)
    for (i in resultPixels.indices) {
        resultPixels[i] = Color.WHITE
    }

    // Для кожного фрагмента
    for (fragment in fragments) {
        // Отримуємо пікселі фрагмента
        val fragmentPixels = IntArray(fragment.width * fragment.height)
        fragment.getPixels(fragmentPixels, 0, fragment.width, 0, 0, fragment.width,
fragment.height)

        // Для кожного чорного пікселя у фрагменті
        for (y in 0 until fragment.height) {
            for (x in 0 until fragment.width) {
                val index = y * fragment.width + x
                val pixel = fragmentPixels[index]

                // Якщо піксель чорний (частина QR-коду)
                if (pixelIsDark(pixel)) {
                    // Додаємо його до результату
                    val resultIndex = y * maxWidth + x
                    if (resultIndex < resultPixels.size) {
                        resultPixels[resultIndex] = Color.BLACK
                    }
                }
            }
        }
    }
```

```
        // Встановлюємо пікселі до результуючого зображення
        result.setPixels(resultPixels, 0, maxWidth, 0, 0, maxWidth, maxHeight)

        // Застосовуємо морфологічні операції для покращення якості
        val enhanced = enhanceWithMorphology(result)

        return enhanced
    }

    /**
     * Перевірка, чи є піксель темним (частиною QR-коду)
     */
    private fun pixelIsDark(pixel: Int): Boolean {
        val r = Color.red(pixel)
        val g = Color.green(pixel)
        val b = Color.blue(pixel)
        val brightness = (0.299 * r + 0.587 * g + 0.114 * b).toInt()
        return brightness < 128
    }

    /**
     * Покращення QR-коду морфологічними операціями
     */
    private fun enhanceWithMorphology(bitmap: Bitmap): Bitmap {
        // Бінаризація
        val binarized = binarize(bitmap)

        // Застосовуємо морфологічні операції для покращення якості
        val dilated = dilate(binarized, 1)  // Розширення для з'єднання близьких
компонентів
        val eroded = erode(dilated, 1)      // Звуження для видалення шуму

        return eroded
    }

    /**
     * Бінаризація зображення (перетворення на чорно-біле)
     */
    private fun binarize(bitmap: Bitmap): Bitmap {
```

```
    val width = bitmap.width
    val height = bitmap.height
    val pixels = IntArray(width * height)
    bitmap.getPixels(pixels, 0, width, 0, 0, width, height)

    for (i in pixels.indices) {
        val pixel = pixels[i]
        val r = Color.red(pixel)
        val g = Color.green(pixel)
        val b = Color.blue(pixel)
        val brightness = (0.299 * r + 0.587 * g + 0.114 * b).toInt()
        pixels[i] = if (brightness < 128) Color.BLACK else Color.WHITE
    }

    val result = createBitmap(width , height)
    result.setPixels(pixels, 0, width, 0, 0, width, height)
    return result
}

/**
 * Дилатація: розширення чорних областей
 */
private fun dilate(src: Bitmap , radius: Int): Bitmap {
    val width = src.width
    val height = src.height
    val pixels = IntArray(width * height)
    src.getPixels(pixels, 0, width, 0, 0, width, height)
    val result = IntArray(width * height)

    for (y in 0 until height) {
        for (x in 0 until width) {
            var anyBlack = false

            // Перевіряємо сусідні пікселі
            for (dy in -radius..radius) {
                for (dx in -radius..radius) {
                    val nx = x + dx
                    val ny = y + dy
```

```
                if (nx in 0 until width && ny in 0 until height) {
                    val pixel = pixels[ny * width + nx]
                    if (pixel == Color.BLACK) {
                        anyBlack = true
                        break
                    }
                }
            }
            if (anyBlack) break
        }

        result[y * width + x] = if (anyBlack) Color.BLACK else Color.WHITE
    }
  }

  val resultBitmap = createBitmap(width , height)
  resultBitmap.setPixels(result, 0, width, 0, 0, width, height)
  return resultBitmap
}

/**
 * Ерозія: звуження чорних областей
 */
private fun erode(src: Bitmap , radius: Int): Bitmap {
  val width = src.width
  val height = src.height
  val pixels = IntArray(width * height)
  src.getPixels(pixels, 0, width, 0, 0, width, height)
  val result = IntArray(width * height)

  for (y in 0 until height) {
    for (x in 0 until width) {
      var allBlack = true

      // Перевіряємо сусідні пікселі
      for (dy in -radius..radius) {
        for (dx in -radius..radius) {
          val nx = x + dx
          val ny = y + dy
```

```
                if (nx in 0 until width && ny in 0 until height) {
                    val pixel = pixels[ny * width + nx]
                    if (pixel != Color.BLACK) {
                        allBlack = false
                        break
                    }
                } else {
                    // Пікселі за межами вважаються білими
                    allBlack = false
                    break
                }
            }
            if (!allBlack) break
        }

        result[y * width + x] = if (allBlack) Color.BLACK else Color.WHITE
      }
    }

    val resultBitmap = createBitmap(width , height)
    resultBitmap.setPixels(result, 0, width, 0, 0, width, height)
    return resultBitmap
  }

  /**
   * Функція для виявлення та виправлення нахилу QR-коду
   * Необов'язкова, якщо фрагменти не повернуті
   */
  private fun correctSkew(bitmap: Bitmap): Bitmap {
    // Для простоти зараз пропускаємо - можна додати за потреби
    return bitmap
  }
}package com.example.qrcodes

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
```

```
import android.content.Intent
import android.widget.Button

@androidx.camera.core.ExperimentalGetImage
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Налаштування кнопок для навігації
        findViewById<Button>(R.id.btnScanQR).setOnClickListener {
            startActivity(Intent(this, QRScannerActivity::class.java))
        }
```

Додаток Д
Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ГЕНЕРУВАННЯ ТА ВІДНОВЛЕННЯ QR-КОДІВ З ВИКОРИСТАННЯМ КОДІВ РІДА-СОЛОМОНА

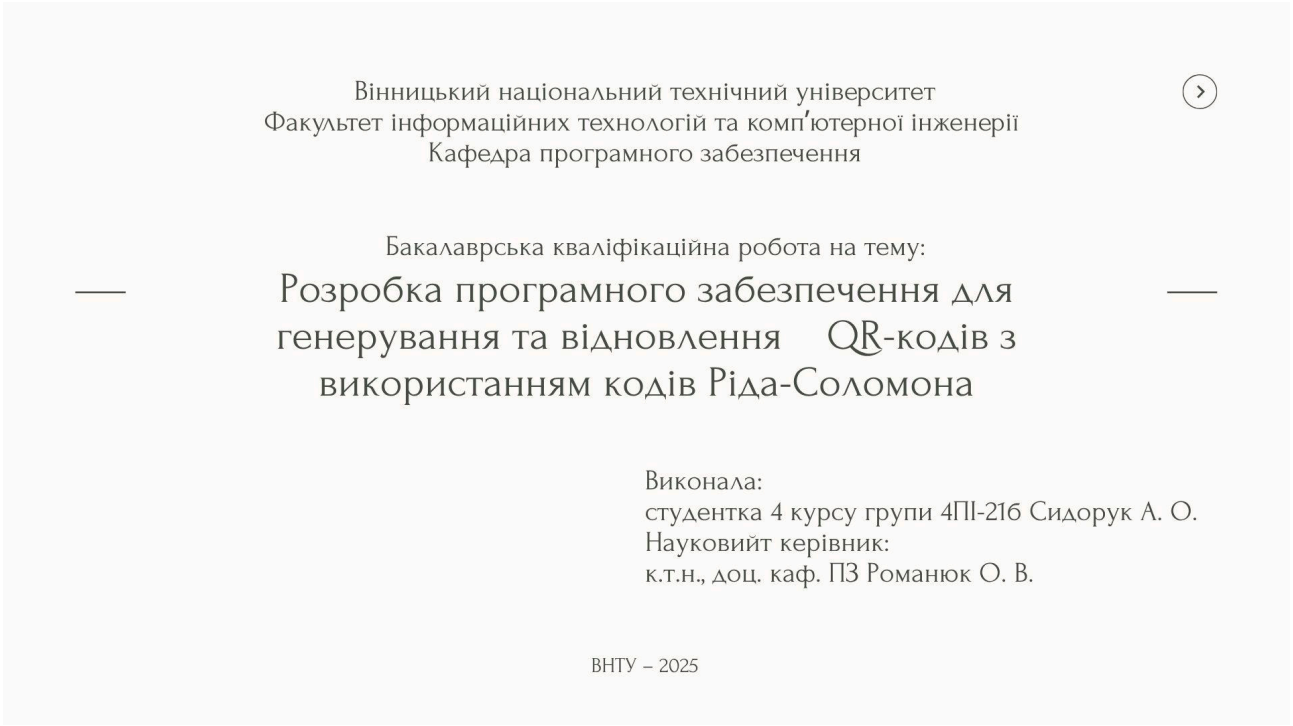


Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Актуальність теми

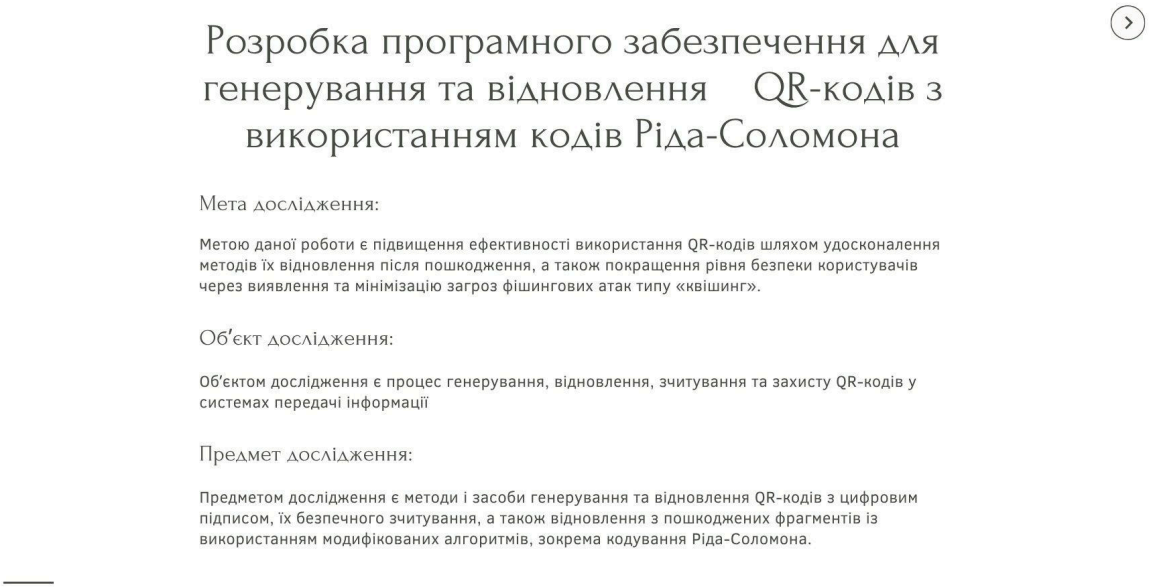


Рисунок Г.3 – Мета, об’єкт дослідження та предмет дослідження

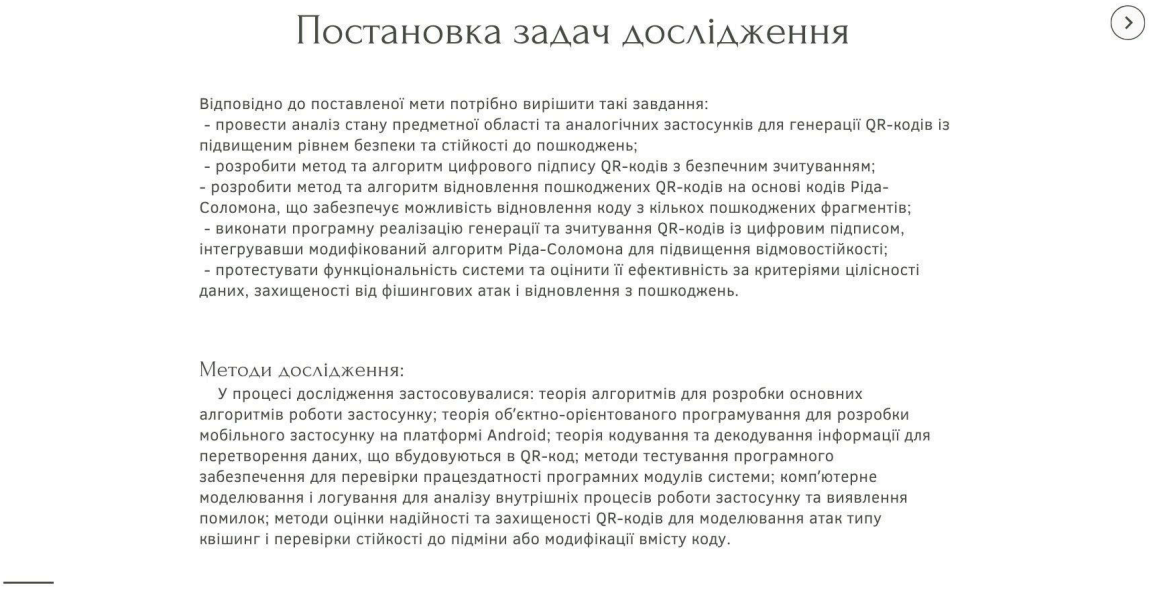


Рисунок Г.4 – Постановка задач дослідження

Розробка програмного забезпечення для генерування та відновлення QR-кодів з використанням кодів Ріда-Соломона

Новизна результатів:

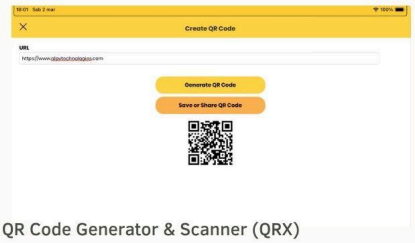
1. Удосконалено метод цифрового підпису QR-кодів, який на відміну від відомих дозволяє здійснювати перевірку їх автентичності та забезпечує захист від фішингових атак, зокрема квішингу, що дозволило підвищити рівень довіри до джерела інформації та гарантувати цілісність даних у процесі зчитування QR-коду.
2. Удосконалено метод відновлення пошкоджених QR-кодів на основі кодів Ріда-Соломона, у якому, на відміну від класичних підходів, використано поля Галуа розмірності 2^{16}, що дозволило значно підвищити точність та ефективність відновлення даних у QR-кодах навіть за умов значних пошкоджень.
3. Запропоновано нову архітектуру мобільного застосунку, побудовану на базі Android із використанням мови Kotlin, особливістю якої є охоплення повного циклу роботи з QR-кодами – від генерації та підпису до сканування, перевірки й відновлення, що дало можливість створити єдину безпечну екосистему для роботи з QR-кодами на мобільному пристрої.

Практична цінність.

Практична цінність результатів бакалаврської роботи полягає у створенні алгоритмів і програм для генерації та відновлення QR-кодів із цифровим підписом, удосконаленими кодами Ріда-Соломона, безпечним зчитуванням, виявленням підробок і відновленням пошкоджених кодів. Рішення застосовне у медицині, логістиці, електронному документообігу та цифровій ідентифікації.

Рисунок Г.5 – Наукова новизна отриманих результатів

Аналоги



QR Code Generator & Scanner (QRX)

QRazyBox

QR TIGER

Рисунок Г.6 – Порівняльний аналіз аналогів

Порівняльний аналіз аналогів

Критерій	QRX	QR TIGER	QRazyBox	Advanced QR Scanner
Підтримка цифрових даних із підписом	-	-	-	+
Можливість сканувати зображення з галереї	+	+	-	+
Відновлення пошкоджених QR-кодів	-	-	+	+
Захист від QR-фішингу	-	+	-	+
Можливість попереднього перегляду вмісту	+	+	-	+
Підтримка роботи з кількома фрагментами коду	-	-	-	+
Сумарний коефіцієнт	2	3	1	6

Рисунок Г.7 – Порівняльний аналіз аналогів

Діаграма прецедентів архітектури застосунку

Під час розробки програми було вирішено створити структурно нову архітектуру, яка могла б вмістити весь функціонал та сприяла кращій роботі і легшому розумінню даного мобільного застосунку.

Прецедент «Відновити QR-код» є ключовим, спрямованим на об'єднання фрагментів для створення повного QR-коду. Він розширюється прецедентом «Керувати клочами» (вибір і обробка фрагментів) та включає «Додати цифровий підпис» для захисту даних і «Перевірити цифровий підпис» для перевірки достовірності. Прецедент «Сканувати QR-код» відповідає за аналіз зображень.

Рисунок Г.8 – Діаграма прецедентів архітектури застосунку

Діаграма класів моделі системи

Діаграма класів розкриває внутрішню роботу та взаємодію активностей.Клас «QRData» зберігає дані QR-кодів, включаючи вміст, прапорці підпису та корекції Ріда-Соломона. «DigitalSignatureManager» керує підписами, використовуючи «SecureKeyManager» і повертає «SignatureResult». «ImageUtils» обробляє зображення, включаючи відновлення фрагментів, а «ReedSolomonUtils» реалізує кодування в полі Галуа 2^{16}.

«QRGeneratorActivity» використовує «DigitalSignatureManager», «ReedSolomonUtils» і «QRData» для генерації, «QRScannerActivity» — для сканування з «DigitalSignatureManager» і «QRData», а «QRReconstructActivity» — для відновлення з «ImageUtils» і «QRData». Зв'язки, як-от залежність «DigitalSignatureManager» від «SecureKeyManager», забезпечують логічну структуру.

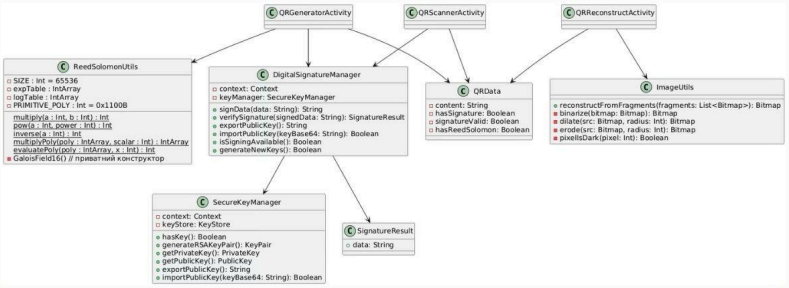
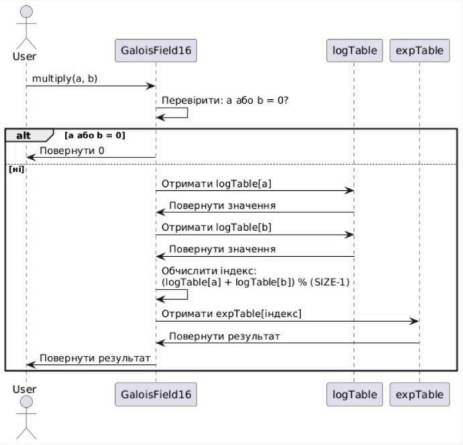


Рисунок Г.9 – Діаграма класів моделі системи

Метод та діаграма послідовності модифікованих кодів Ріда-Соломона



Процес множення в полі Галуа 2^{16} розпочинається з виклику методу `multiply(a, b)` класу `GaloisField16`, який виступає центральним об'єктом для обробки. Спочатку виконується перевірка: якщо хоча б один аргумент (a або b) дорівнює нулю, метод одразу повертає 0, оптимізуючи обчислення завдяки властивостям скінченного поля.

Якщо обидва аргументи відмінні від нуля, клас звертається до таблиці `logTable` для отримання логарифмічних значень a і b. Далі обчислюється сума цих логарифмів із модулем (SIZE-1), що дозволяє ефективно виконати множення в полі Галуа.

На завершальному етапі `GaloisField16` використовує таблицю expTable`, передаючи індекс для отримання кінцевого результату множення. Таблиці `expTable` і `logTable`, ініціалізовані примітивним поліномом 0x1100B, забезпечують швидке виконання операцій. Результати повертаються користувачу, завершуючи послідовність дій.

Рисунок Г.10 – Метод та діаграма послідовності модифікованих кодів Ріда-Соломона

Метод та блок-схема алгоритму відновлення QR-коду з фрагментів

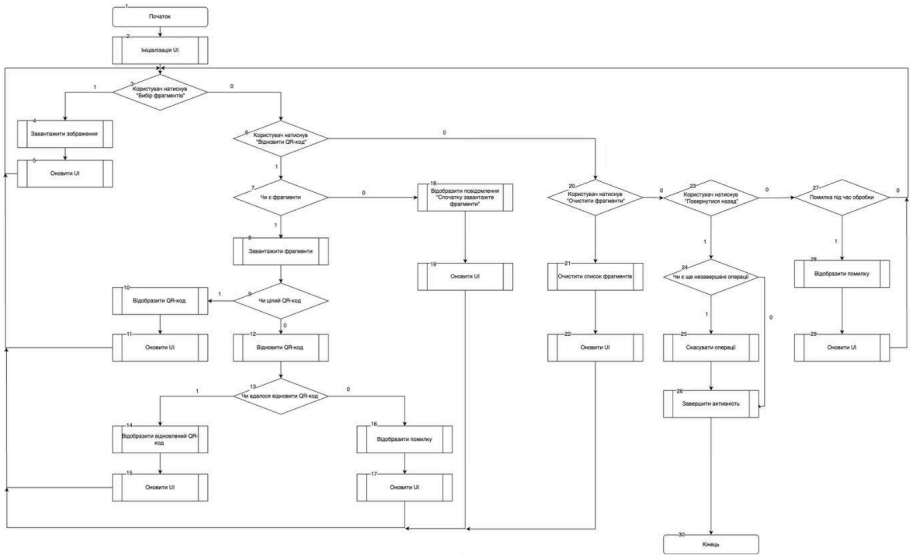


Рисунок Г.11 – Метод та блок-схема алгоритму відновлення QR-коду з фрагментів

Метод та блок-схема алгоритму відновлення QR-коду з фрагментів

Алгоритм бере початок з ініціалізації порграми, де відкривається початковий інтерфейс користувача На даному етапі користувачу пропонується обрати дію «вибрати фрагмент», і залежно від його рішення, якщо кнопка буде натиснута, то з'явиться можливість завантажити зображення фрагмента QR-коду, після чого інтерфейс оновлюється для відображення завантаженого вмісту.

Далі програма проводить перевірку, чи було обрано фотографій для обробки, і якщо так, то проводиться аналіз, чи достатньо фрагментів для відновлення. У разі недостатньої кількості фрагментів користувача повертають до завантаження додаткових частин, супроводжуючи це відповідними повідомленнями. Якщо фрагментів достатньо, то наступним етапом є відновлення QR-коду і подальше відображення його на екрані. У разі невдачі відновлення, користувач отримує помилку.

Також є можливість очистити фрагменти, при потребі, після чого інтерфейс знову відображає оновлений стан. Також є можливість повернутися назад до головного меню, під час даного процесу перевіряється наявність незавершених операцій, що дозволяє або продовжити відновлення, або очистити інформацію.

Рисунок Г.12 – Метод та блок-схема алгоритму відновлення QR-коду з фрагментів

Порівняльний аналіз мов програмування

Критерій	Swift	Flutter	Kotlin	Python
Глибока інтеграція з Android SDK та системними API	0	0	1	0
Нативна підтримка електронного цифрового підпису	0	0	1	1
Можливість низькорівневої роботи з QR-кодами та двійковими даними	1	1	1	0
Підтримка великих полів Галуа 216 у сторонніх криптобібліотеках	0	0	1	1
Придатність до офлайн-підпису на мобільних пристроях без серверів	0	1	1	0
Документована підтримка Reed–Solomon з можливістю розширення	0	0	1	1
Підсумок	1	2	6	3

Рисунок Г.13 – Порівняльний аналіз мов програмуванняу

Порівняльний аналіз середовищ розробки

Критерій	Android Studio	IntelliJ IDEA	Eclipse	VS Code
Офіційна підтримка Android SDK, Keystore	1	0	0	0
Повноцінна інтеграція Kotlin без додаткових налаштувань	1	1	0	0
Вбудований UI-редактор, Android емулятор	1	0	0	0
Підтримка легких скриптів або CLI-модулів	0	1	1	1
Можливість реалізації криптографії, QR та Gradle-збірок	1	1	1	0
Мінімальна підтримка Kotlin	0	0	1	1
Підсумок	4	3	3	2

Рисунок Г.14 – Порівняльний аналіз середовищ розробки

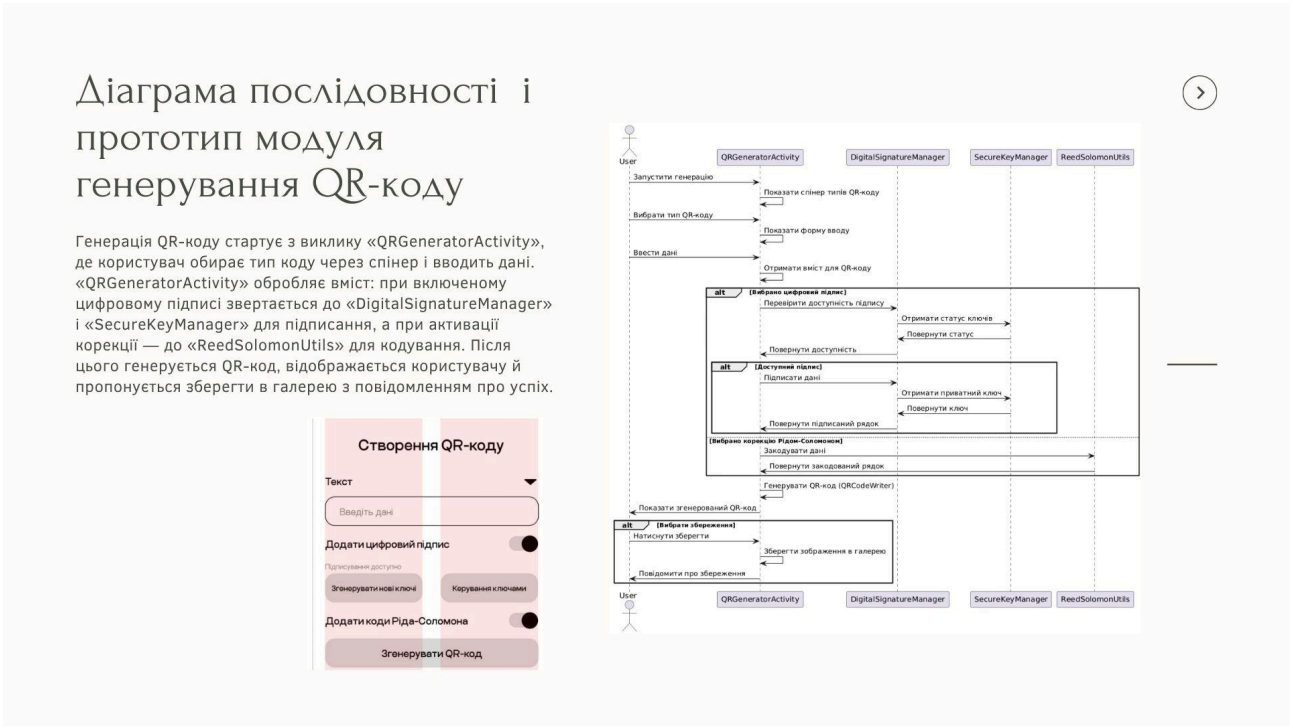


Рисунок Г.15 – Діаграма послідовності і прототип модуля генерування QR-коду

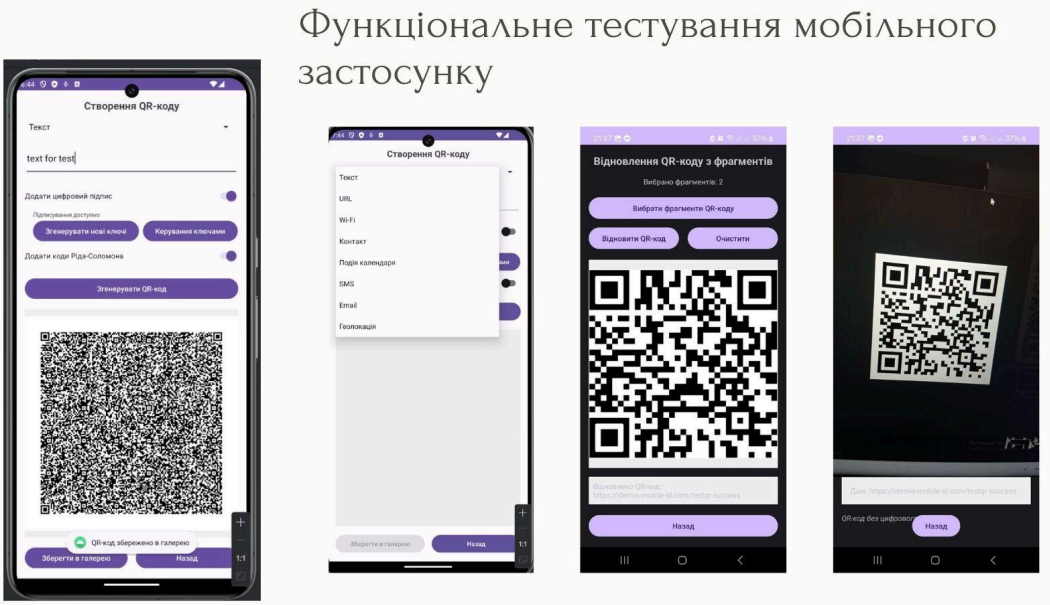


Рисунок Г.16 – Функціональне тестування мобільного застосунку

Тестування продуктивності застосунку

QR-код	Рівень пошкодження	Час зчитування (мс) 1 спроба	Час зчитування (мс) 2 спроба	Час зчитування (мс) 3 спроба
№1	0%	150	160	155
№1	20%	211	207	201
№1	40%	310	313	300
№2	0%	147	162	154
№2	20%	205	208	205
№2	40%	-	364	372
№3	0%	152	156	149
№3	20%	215	204	211
№3	40%	325	312	304

Рисунок Г.17 – Тестування продуктивності мобільного застосунку

Тестування продуктивності застосунку

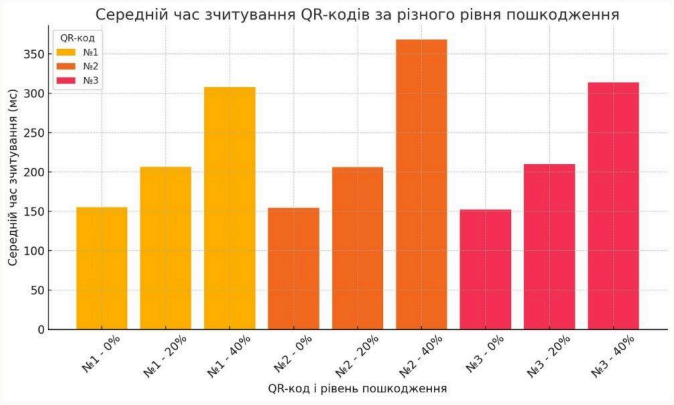


Рисунок Г.18 – Тестування продуктивності мобільного застосунку

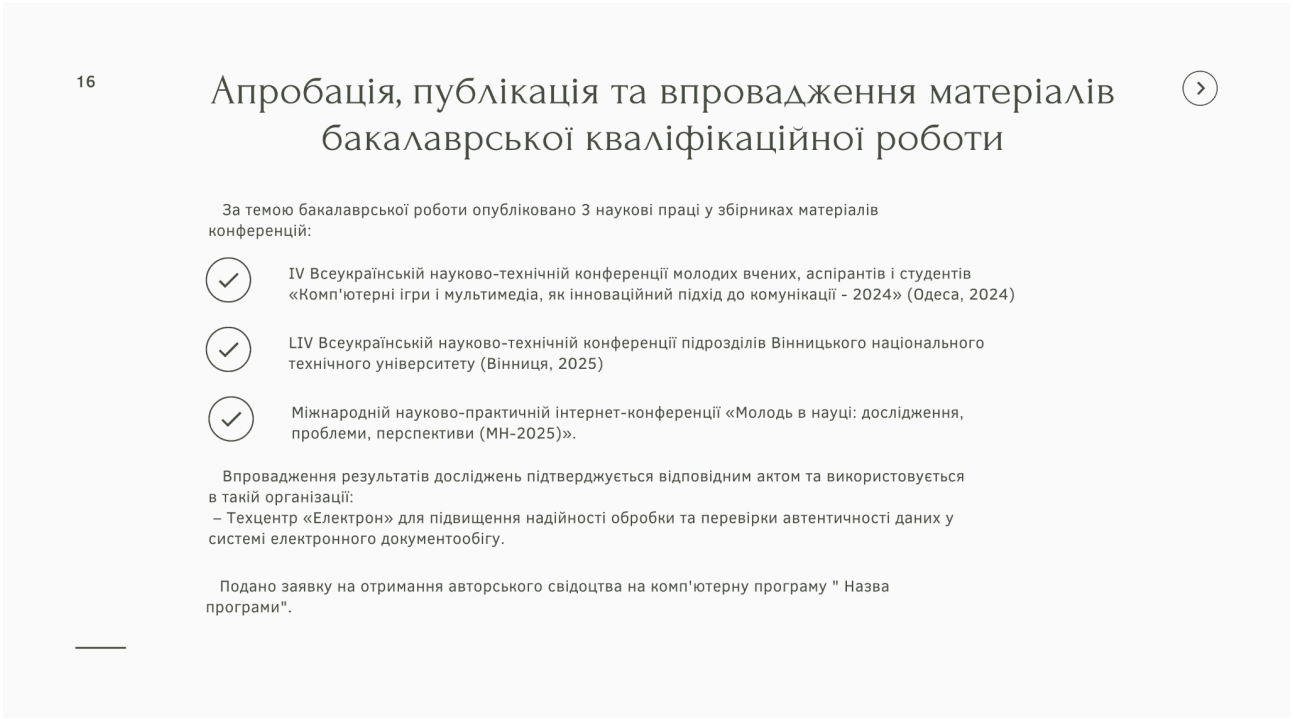


Рисунок Г.19 – Апробація, публікація та впровадження матеріалів бакалаврської кваліфікаційної роботи; висновки

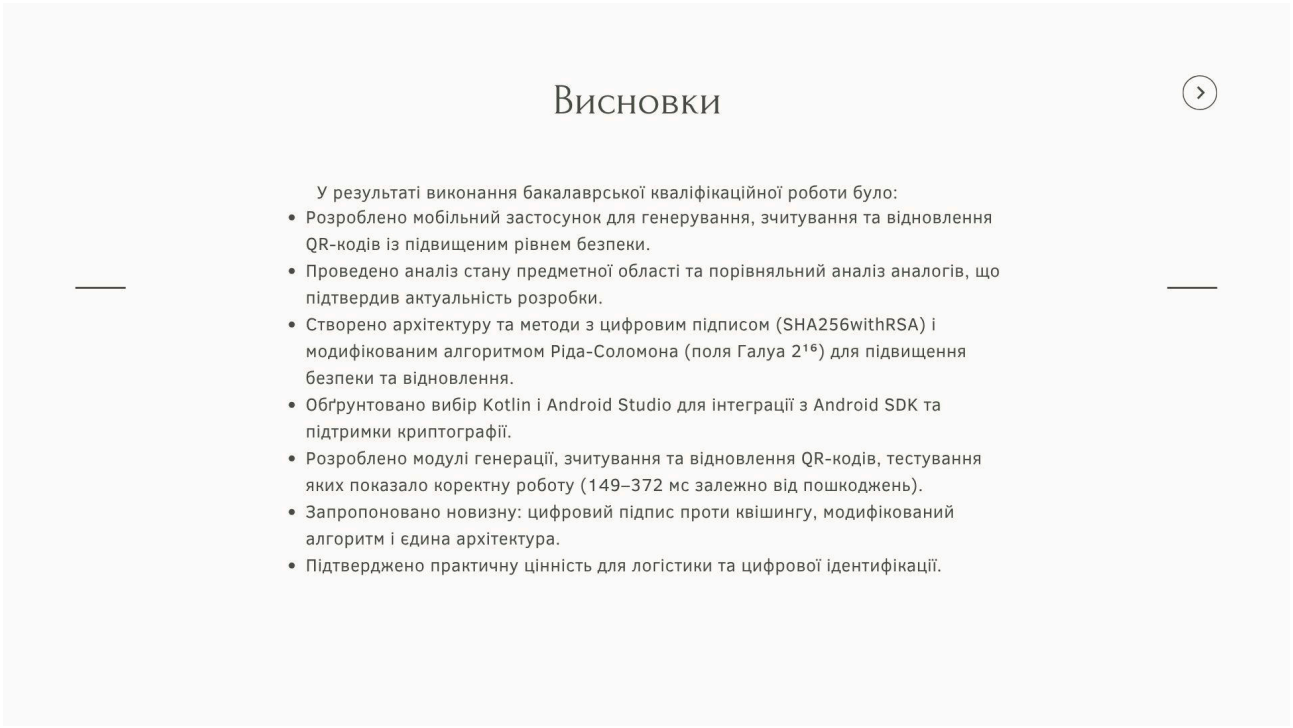


Рисунок Г.20 – Висновки

Рисунок Г.21 – Фінальний слайд