

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗАХИСТУ ДАНИХ НАТИВНИХ

(назва теми)

АНДРОІД ЗАСТОСУВАНЬ

SECURITY SOFTWARE FOR NATIVE ANDROID APPLICATIONS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-219

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ЖИР Є.С.

(ПРІЗВИЩЕ та ініціали)

Керівник ТАБУНЩИК Г.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н., проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЖИР Єгора Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для захисту даних нативних андроїд застосувань Security Software for Native Android Applications

керівник проєкту (роботи) к.т.н., професор, ТАБУНЦІК Галина Володимирівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «4» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи)) 1 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір програмних засобів.
3. Розробка програми. 4. Експлуатація, тестування та результати.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1–4 Основна частина	ТАБУНЩИК Г.В., професор		
Нормоконтроль	БСЛОВА А.В., асистент		

7. Дата видачі завдання « 4 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи та підбір літератури.	1 тиждень	Розділ 1
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Вибір програмних засобів	4 тиждень	Розділ 2
4	Розробка додатка	5 тиждень	Розділ 3
5	Тестування	6 тиждень	Розділ 4
6	Формування висновків та написання пояснювальної записки	7 тиждень	Додатки
7	Нормоконтроль та рецензування.	8 тиждень	
8	Захист роботи	9 тиждень	

Студент(ка)

Підпис
(підпис)

Єгор ЖИР

(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Галина ТАБУНЩИК

(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
254 с., 12 табл., 72 рис., 2 дод., 17 джерел.

ЗАХИСТ ДАНИХ, ANDROID ДОДАТОК, AES, ШИФРУВАННЯ,
КОНФІДЕНЦІЙНІСТЬ, ANDROID KEYSTORE, ОБФУСКАЦІЯ,
AUTHENTICATION, БАНКІВСЬКА КАРТКА.

Об'єкт дослідження – безпека даних.

Предмет дослідження – програмне забезпечення для захисту даних нативних андроїд застосунків.

Мета роботи полягає у підвищенні захищеності даних користувача в андроїд додатку.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, нативна розробка андроїд застосунків, мова програмування Kotlin, база даних SQLite, архітектура Model-View-Intent, інтерфейс був розроблений за допомогою Jetpack Compose, пристрій Xiaomi Redmi 9.

Результати: розроблено андроїд додаток-гаманець для збереження даних банківських карток для андроїд застосунків з використанням Kotlin, що реалізує шифрування, а тим самим захист даних за допомогою алгоритму AES.

Висновки: проаналізовано та розроблено андроїд додаток, який може безпечно зберігати дані.

Галузь використання – користувачі мобільних телефонів на основі операційної системи Android.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 254 pages, 12 tables, 72 figures, 2 appendices, 17 sources.

DATA PROTECTION, ANDROID APP, AES, ENCRYPTION, PRIVACY, ANDROID KEYSTORE, OBFUSCATION, AUTHENTICATION, BANK CARD.

Research object – data security.

Research subject – Security software for native Android applications.

The purpose of the thesis is to increase the security of user data in the Android application.

Materials, methods and technical means: structured and object-oriented programming, native development of Android applications, Kotlin programming language, SQLite database, Model-View-Intent architecture, interface was developed using Jetpack Compose, device Xiaomi Redmi 9.

Results: An Android application-wallet was developed for storing bank card data for Android applications using Kotlin, which implements encryption and thus data protection using the AES algorithm.

Conclusions: analyzed and developed an android application that can store data securely.

Field of use – users of mobile phones based on the Android operating system.

ЗМІСТ

	С.
Перелік скорочень.....	8
Вступ	9
1 Аналіз предметної області	10
1.1 Аналіз існуючих додатків, що вирішують аналогічні проблеми.....	10
1.1.1 Додаток «SecureCard».....	10
1.1.2 Додаток «Google Wallet».....	10
1.2 Методи розробки мобільних додатків	11
1.3 Захист даних	14
1.3.1 Захист від інтерфейсного втручання	14
1.3.2 Шифрування даних.....	14
1.3.3 Захист вихідного коду від зворотної розробки.....	17
2 Вибір програмних засобів	18
2.1 Вибір мови програмування Kotlin чи Java	18
2.2 Вибір інтегрованого середовища розробки.....	19
2.3 Архітектура та патерни проєктування програмного забезпечення	22
2.4 Вибір сховища для даних	24
2.5 Шифрування даних	24
2.6 Вибір системи впровадження залежності (DI)	26
2.7 Вибір інструменту для створення інтерфейсу користувача.....	28
2.8 Оптимізація та безпека APK.....	29
3 Розробка програми.....	30
3.1 Аналіз вимог до застосування	30
3.1.1 Концепція.....	30
3.1.2 Функціональні вимоги до системи.....	30
3.1.3 Специфікація прецеденту «Розблокування додатка»	31
3.1.4 Специфікація прецеденту «Додавання картки у додаток».....	32
3.2 Архітектура і структура додатка	33

3.3	Проектування інтерфейсу	35
3.3.1	Дизайн система і локалізація	35
3.3.2	Розбір екранів	37
3.3.3	Захист від шпигунського програмного забезпечення	54
3.4.	Проектування даних	55
3.4.1	Огляд шару data.....	55
3.4.2	Огляд шару domain	61
4	Експлуатація, тестування та результати.....	63
4.1	Створення захищеного APK	63
4.2	Експлуатація додатка	67
4.3	Тестування додатка.....	68
4.3.1	Unit тести додатка.....	68
4.3.2	Тестування на реальних пристроях.....	70
	Висновки	72
	Перелік джерел і посилань	73
	ДОДАТОК А Код програми	75
	ДОДАТОК Б Слайди презентації.....	246

ПЕРЕЛІК СКОРОЧЕНЬ

ADB	– Android Debug Bridge;
AES	– Advanced Encryption Standard;
API	– Application Programming Interface;
APK	– Android Application Package;
CBC	– Cipher Block Chaining;
CIV	– Synthetic Initialization Vector;
CVV	– Card Verification Code;
DES	– Data Encryption Standard;
DI	– Dependency Injection;
DSL	– Domain-Specific Language;
GCM	– Galois/Counter Mode;
IDE	– Integrated Development Environment;
JVM	– Java Virtual Machine;
KMM	– Kotlin Multiplatform Mobile;
MVC	– Model-View-Controller;
MVI	– Model-View-Intent;
MVP	– Model-View- Presenter;
MVVM	– Model-View-ViewModel;
NFC	– Near Field Communication;
NPE	– Null Pointer Exception;
PIN	– Personal Identification Number;
PKCS	– Public Key Cryptography Standards;
RSA	– Rivest-Shamir-Adleman;
UI	– User Interface;
URI	– Uniform Resource Identifier;
XML	– Extensible Markup Language;
ООП	– Об’єктно Орієнтоване Програмування.

ВСТУП

У сучасному світі андроїд пристрої займають дуже важливе місце у житті людини. Ми звикли тримати свої конфіденційні дані на пристроях, то може бути фотографії, якісь замітки чи повідомлення у месенджері. І звісно при такій популярності буде зростати загроза від шахраїв які хочуть викрасти ці дані. Хоч за останні роки Google і став робити більший акцент на безпеку даних користувачів, але загроза все ж залишається, Android як і раніше залишається більш схильним до загроз шкідливого програмного забезпечення, а тому ця тема є актуальною для розробників Android додатків, бо їм треба брати захист даних у свої руки.

Об'єктом дослідження є безпека даних.

Предмет дослідження – програмне забезпечення для захисту даних нативних андроїд застосувань.

Мета роботи полягає у підвищенні захищеності даних користувача в андроїд додатку.

Для досягнення поставленої мети у роботі вирішувалися такі завдання:

- дослідити якими способами можна здобути дані додатка;
- вивчити як протидіяти спробам здобути дані додатка;
- розробити нативний Android додаток який спроможний захистити внутрішні дані;
- оформити пояснювальну записку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз існуючих додатків, що вирішують аналогічні проблеми

1.1.1 Додаток «SecureCard»

SecureCard [1] – це застосунок для зберігання дисконтних та банківських карток. Застосунок має можливість додавати дисконтні та банківські картки за допомогою ручного вводу, робити пошук серед них, змінювати тему, вмикати та вимикати засоби захисту як ПІН та сканування відбитка пальця, також має захист від запису екрана (рис. 1.1).

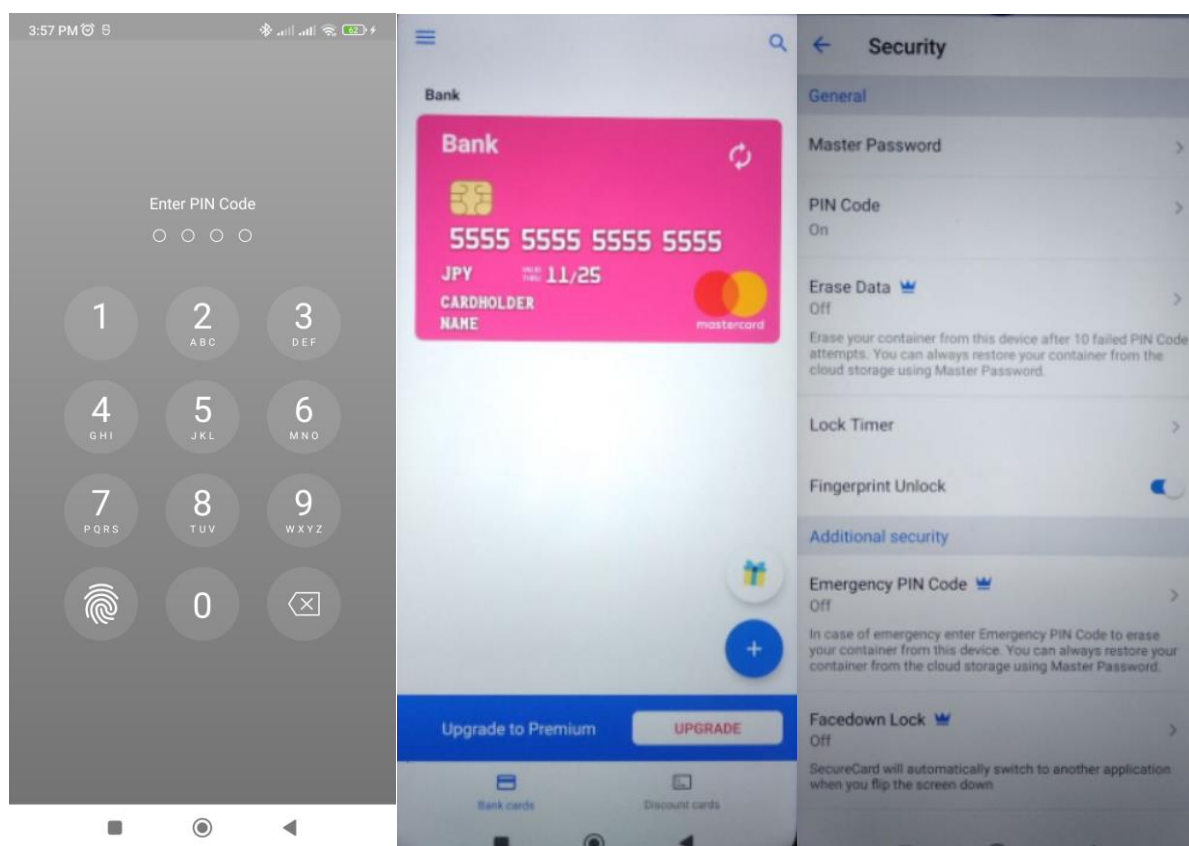


Рисунок 1.1 – Знімки екрана з функціональністю додатка SecureCard

1.1.2 Додаток «Google Wallet»

Google Wallet [2] – це застосунок спрямований на зберігання карток та розрахування ними за допомогою NFC. Додаток використовує дизайн Material 3.

Зі сторони функціонала, це NFC розрахунок, додавання карток ручним способом чи скануванням, є захист від запису екрана (рис. 1.2).

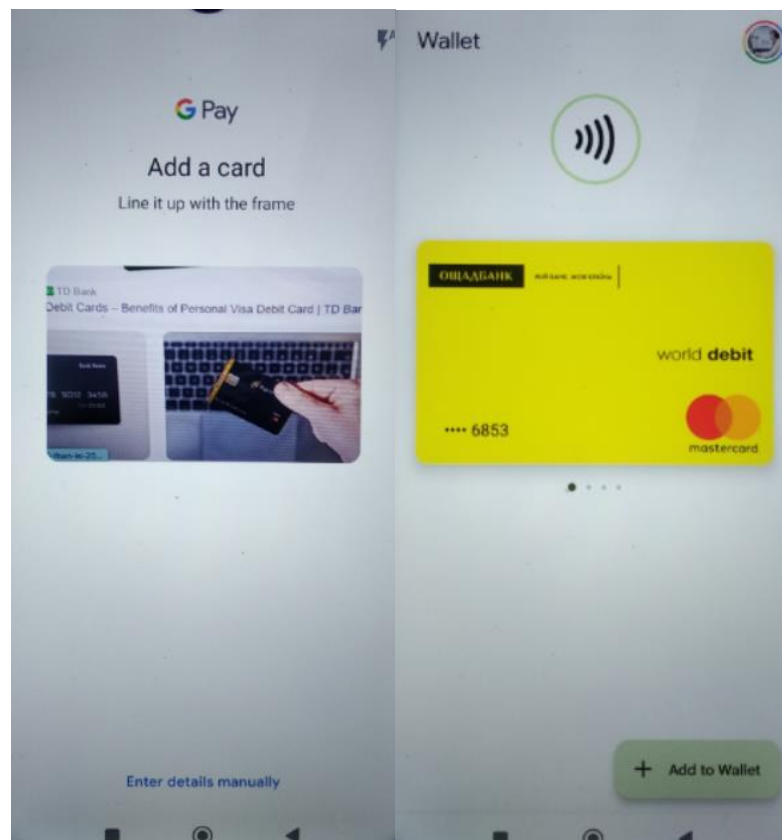


Рисунок 1.2 – Знімки екрана з функціональністю додатка Google Wallet

1.2 Методи розробки мобільних додатків

Якщо вийти на вулицю і почати запитувати людей, чи мають вони смартфон, то буде складно знайти людину, яка не мала б мобільного девайсу, і це робить галузь мобільної розробки дуже привабливою для бізнесу.

Добре, дуже приваблива для бізнесу, але з чого почати? Почати слід з вибору методу розробки мобільних застосунків. Мабуть, ви колись чули суперечки, що краще нативна або мультиплатформна розробка або ще як її називають кросплатформова?

Що таке нативна мобільна розробка – це розробка застосунків для певної операційної системи – в більшості випадків це буде Android чи iOS. Ці застосунки

розробляються з певним набором інструментів, для прикладу візьмемо Android, щоб створити нативний додаток ми повинні писати мовою програмування Java чи Kotlin, у випадку з iOS – це Objective-C, чи новіша мова як Swift.

Що таке мультиплатформна розробка – це розробка застосувань яка буде сумісна між декількома операційними системами, в більшості випадках це Android та iOS. Для розробки використовують такі поширені фреймворки з відкритим вихідним кодом як: React Native, Flutter, Cordova чи Kotlin Multiplatform Mobile також відомий як KMM [3]. Проведено порівняння нативного і мультиплатформного додатка (див. табл. 1.1).

Таблиця 1.1 – Порівняння додатків

	Нативний додаток	Мультиплатформний додаток
Продуктивність додатка	Має перевагу, через пряме спілкування з API функціональністю та має базову програмну мову	Поступається, тому що потрібно спілкуватися через посередника і зазвичай використовує посередницьку мову програмування
Вартість розробки	Поступається якщо потрібно зробити два додатки на різні платформи. Має перевагу, якщо потрібно зробити складний додаток	Має перевагу, якщо потрібно зробити не дуже складний додаток.
Швидкість розробки	Має перевагу для складних додатків.	Має перевагу, якщо потрібно зробити не дуже великий та складний додаток
Ризики різної логіки на платформах	Поступається, тому що має різний вихідний код до кожної платформи	Має перевагу, тому що має єдиний вихідний код
Функціональність додатка	Має перевагу, тому що має прямий доступ до забезпечення пристрою	Поступається, тому що не має прямого доступу до забезпечення пристрою, тому якийсь функціонал може бути втраченим

Кінець таблиці 1.1

	Нативний додаток	Мультиплатформний додаток
Інтуїтивний інтерфейс	Має перевагу, тому що кожна операційна система має свій інтуїтивний інтерфейс до якого користувачі звикли	Поступається, тому що буде єдиний інтерфейс, а тим самим менш інтуїтивний для користувача який звик до нативного інтерфейсу
Підтримка	Має перевагу, тому що підтримкою займаються дистриб'ютори операційної системи	Поступається, тому що підтримкою займаються менш довірені компанії

Кожен метод має свої недоліки та переваги, тому вибір метода повинен виходити із мети, якщо бюджет обмежений і потрібно швидко зробити не дуже великий та складний проєкт, краще розробляти мультиплатформний додаток. У випадку коли ви маєте чіткий план, достатньо ресурсів і бачите що нативна функціональність має головну роль у застосунку, то у цьому випадку слід вибрати розробку нативного додатка.

Також буває коли мультиплатформний додаток стає успішним, і приймається рішення розвитку та додавання нового функціоналу, то може стати так, що подальша розробка може стати складнішою та дорогою ніж нативна розробка, в цьому випадку має сенс мігрувати додаток на нативний додаток.

Я вибрав нативну розробку, тому що головною метою моєї дипломної роботи є розробка захисту даних в андроїд додатках, а оскільки мені треба розробити додаток тільки на одну платформу, то найкращим вибором буде нативна розробка.

1.3 Захист даних

Дуже критичним критерієм додатку є захист цілісності і конфіденційності даних, а тому важливо забезпечити захист даних від загроз природного чи штучного характеру.

Природна загроза – це загроза, яка походить від фізичного втручання, наприклад крадіжка мобільного пристрою та безпосередньо зовнішнє (інтерфейсне) втручання у дані додатка.

Штучна загроза – це навпаки загроза, яка походить від внутрішнього втручання у програму, наприклад віруси чи злом додатку.

1.3.1 Захист від інтерфейсного втручання

Інтерфейсне втручання найпростіше та найчастіша форма отримання доступу до даних. Щоб забезпечити захист даних на цьому рівні, ми повинні додати екран авторизації. Це може бути пароль, чи біометричний метод авторизації як сканування відбитка пальця чи розпізнавання обличчя.

1.3.2 Шифрування даних

Шифрування має найважливішу роль у захисту від внутрішнього втручання, якби шифрування не існувало, то будь-який вірус який здатний витягувати дані з файлів додатка, мав би успіх. А тому треба вибрати правильний алгоритм.

Вибираючи шифрування, треба звернути увагу ні кілька факторів:

- рівень захисту;
- мета захисту;
- доступні ресурси.

Рівень захисту залежить від того, що ми хочемо захистити, наприклад, для імені, прізвища або віку, захист буде нижчим, ніж для збережених паролів або даних банківських карток.

Шифрування під час передачі даних, або під час завантаження у базу даних – це мета захисту з якою необхідно визначитися.

Алгоритми шифрування вимагають ресурсів для шифрування та дешифрування даних. Якщо потрібна швидкість, щоб дешифрувати багато даних, треба обирати алгоритм спрямований на оптимізацію ефективності.

Сьогодні найпоширеніші види шифрування – це симетричні та асиметричні.

Симетричне шифрування використовує один ключ, для шифрування та дешифрування. Приклади алгоритмів з симетричним шифруванням: AES, DES і Blowfish. Перевагою є швидкість, що може бути вдалим вибором для шифрування та дешифрування великих обсягів даних. Однак є і недолік – це обов’язкова передача ключа між відправником і одержувачем, що робить ключ вразливим до атак, але для цього існують асиметричне шифрування. Порівняння алгоритмів (див. табл. 1.2) [4].

Асиметричне шифрування – це шифрування якої головна мета є безпечний обмін даними між відправником та отримувачем. На відміну симетричному підходу, асиметричне має два ключа, один публічний та приватний. Публічними ключами, як можливо здогадатися з назви, обмінюються між двома сторонами. Приватний навпаки тримається в секреті. Щоб зрозуміти як це працює, можна взяти за приклад поштовий ящик, поштовий ящик – це публічний ключ, у який можуть кинути лист будь-хто, а приватний ключ, це ваш ключ з яким ви можете відкрити поштовий ящик, та достати листи, на відміну цього прикладу, у програмуванні публічний та приватний ключ математично пов’язані. Але цей зв’язок є дуже затратним і повільним, що може не підійти для деяких випадків. Це шифрування реалізують RSA та Діффі-Геллмана. Порівняння алгоритмів (див. табл. 1.3).

Таблиця 1.2 – Порівняння симетричних алгоритмів

	DES	AES	Blowfish
Розмір блока (bits)	64	128	64
Розмір ключа (bits)	56	128, 192, 256	32-448
Безпека	Слабий рівень захисту	Сильніший рівень захисту ніж у Blowfish	Сильний рівень захисту
Швидкість	Повільний	Швидкий	Швидше ніж AES
Стандартизація	Вважається застарілим	Добре тестований та стандартизований	Менш популярний ніж AES

Вирвалися переможцями Blowfish та AES це два хороші алгоритми, але якщо вибирати з цих двох, то вибір впаде на AES, тому що він більш перевірений та стандартизований.

Таблиця 1.3 – Порівняння асиметричних алгоритмів

	RSA	Діффі-Геллмана
Розмір ключа (bits)	1024-4096	1024-4096
Генерація ключа	На основі складності розкладання чисел	На основі складності обчислення дискретних логарифмів
Безпека	Сильний рівень захисту	Сильний рівень захисту
Швидкість	Повільний	Швидший ніж RSA
Стандартизація	Використовується для цифрових підписів	Використовується переважно для обміну ключами

У цьому порівнянні можна сказати нічия, тому що RSA та Діффі-Геллмана служать різним цілям.

1.3.3 Захист вихідного коду від зворотної розробки

Шифрування даних, це добре, але це не зможе захистити від фактора людини – робити помилки. Можна врахувати всі методи захисту даних, але програма все одно буде схильна до злому, якщо зловмисник зможе дістати вихідний код додатка за допомогою зворотної розробки та знайти вразливі місця програми.

Зворотна розробка – це процес відтворення бізнес-логіки програми із кінцевого продукту. Це може використовуватися в законних цілях, наприклад для навчання, але також може використовуватися в незаконних цілях, наприклад для викрадення інтелектуальної власності або пошук вразливих місць з метою злому.

Щоб зменшити ризики від викрадення вихідного коду, потрібно додати обфускацію у процес будування проєкту. Тому що отримати доступ до вихідного коду дуже легко, просто, відкрити APK з майже з будь-яким архіватором.

Обфускація – це технологія яка робить код складним для розуміння. Це досить базова технологія під час проєктування програми, вона налаштовується досить легко і швидко, та має приємний плюс як зменшення розміру програми, що також має величезну роль для деяких користувачів, але є і мінуси, ускладнює тестування і змушує стежити за винятками для обфускації, тому що без обфускації програма може нормально працювати, а з нею можуть бути помилки.

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ

2.1 Вибір мови програмування Kotlin чи Java

Нативні андроїд додатки пишуться за допомогою Java чи Kotlin, це можливо за допомогою компіляції кода у ByteCode який запускається на JVM [5]. Порівняння мов (див. табл. 2.1) [6].

Таблиця 2.1 – Порівняння Kotlin та Java

Функція	Kotlin	Java
Типізованість	Сильно типізований з неявним оголошенням типів	Сильно типізований з явним оголошенням типів
Безпека від NPE	Має захист	Не має захисту
ООП	Має підтримку	Має підтримку
Функціональне програмування	Має підтримку	Має підтримку
Ефективність	Теоретично менша ефективність, але статистично більша	Теоретично висока ефективність, але статистично менша
Лаконічність	Має перевагу	Поступається
Швидкість написання коду	Має перевагу	Поступається
Кількість помилок	Має менше помилок, тому що захищає від найпоширенішої помилки NPE, а тому має перевагу	Немає захисту від NPE, складніший синтаксис, а тому поступається
Функціональність	Має перевагу	Поступається
Складність навчання	Синтаксис легший, але має багато ключових слів, які можуть бути незрозумілі спочатку	Синтаксис складніший, але якщо є досвід роботи з сі-подібними мовами, то проблем з вивченням не буде

Як бачите, Kotlin має перевагу для розробки, і це не дивно, адже Java писалася математиками, а Kotlin розробниками для розробників, а тому мій вибір падає на Kotlin.

2.2 Вибір інтегрованого середовища розробки

Існує багато середовищ розробки, доступних для написання android застосувань. Ось декілька прикладів:

Eclipse – це середовище розробки з відкритим вихідним кодом. Перша версія IDE була випущена 29 листопада 2001 року. Eclipse підтримує багато мов програмування, включаючи Java. Щоб почати займатися андроїд розробкою треба зробити деякі налаштування. Ця середа, надає багато різноманітних плагінів, одним з них є плагін Android Development Tools, який забезпечує створення проєкту, редактор макету та інші інструменти.

Visual Studio – популярна середа розробки від Microsoft, її головна ціль це розробка на таких мовах програмування як C, C++, C#. Для розробки мобільних додатків, IDE пропонує інструмент для мультиплатформної розробки Xamarin.

Android Studio – середовище розробки від Google на основі IntelliJ IDEA. IDE була анонсована 16 травня 2013 року на Google I/O конференції. Головна ціль середовища є розробка нативних андроїд додатків, але також має велику базу плагінів, серед яких є плагіни націлені на мультиплатформну розробку. Ось деякий функціонал який пропонує IDE:

- конвертація Java коду у Kotlin (рис. 2.1);
- налагодження (рис. 2.2);
- пропонує великий вибір емуляторів (рис. 2.3);
- профайлер (рис. 2.4);
- трансляція фізичного пристрою в IDE (рис. 2.5).

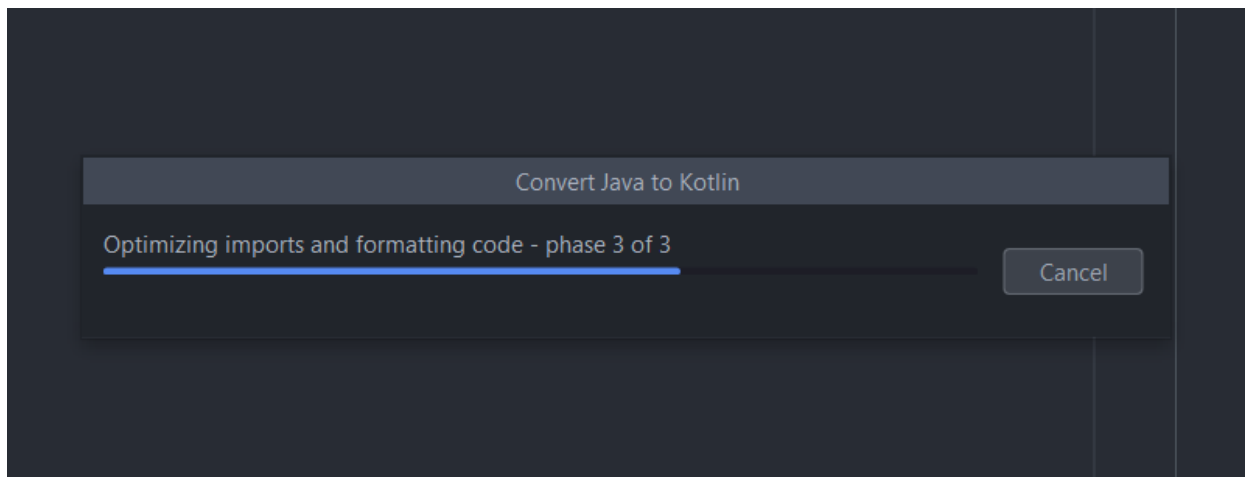


Рисунок 2.1 – Процес конвертація Java коду у Kotlin

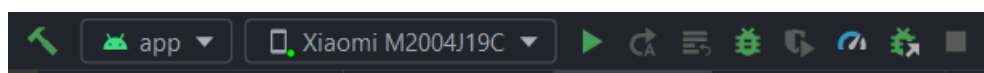


Рисунок 2.2 – Верхня панель налагодження

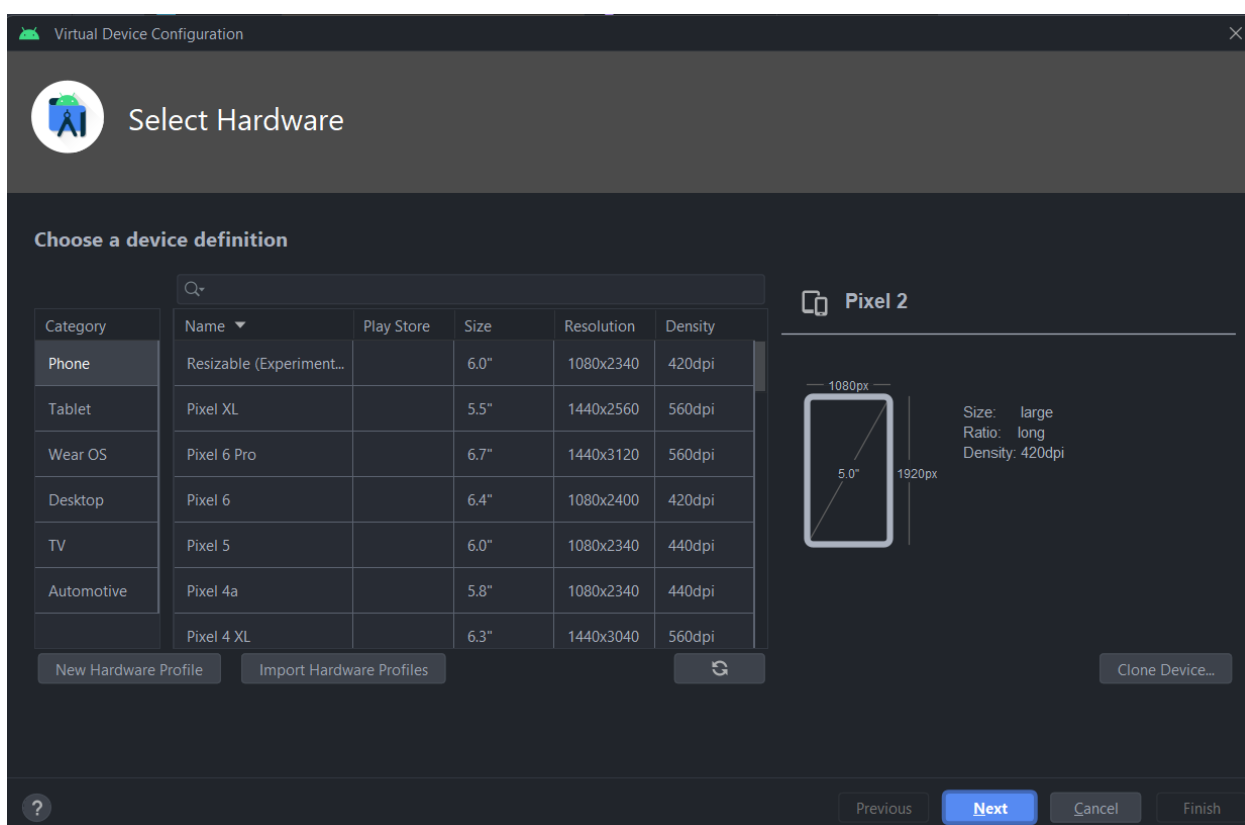


Рисунок 2.3 – Вікно вибору емуляторів

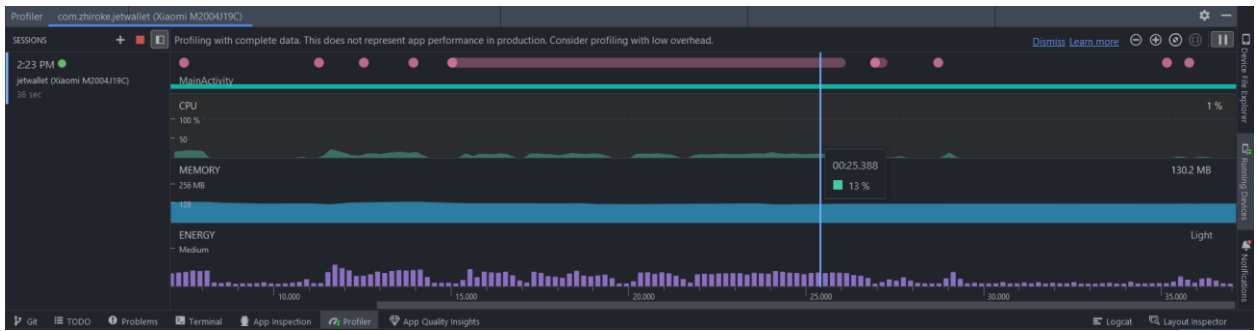


Рисунок 2.4 – Вікно профайлеру

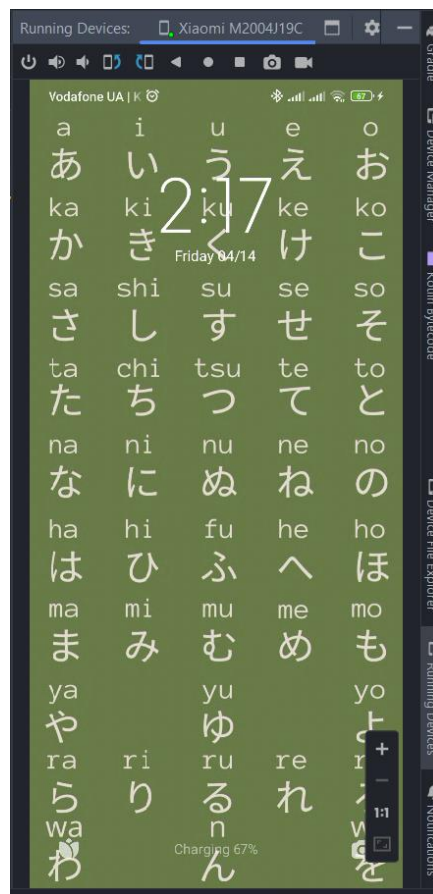


Рисунок 2.5 – Вікно з трансляцією екрана фізичного пристрою

Як можна помітити Android Studio має значну перевагу над іншими інтегрованими середовищами розробки, тому будемо її використовувати.

2.3 Архітектура та патерни проєктування програмного забезпечення

Архітектура та патерни проєктування – це структурні рішення, які використовуються для подолання поширених проблем при розробці додатку.

Поширені патерни проєктування: MVC, MVP, MVVM та MVI.

MVC (Model-View-Controller) – мабуть, найпростіший та найстаріший патерн проєктування. Головна ідея полягає у прямому спілкуванні між екраном та моделлю даних (джерело даних, репозиторій) (рис. 2.6).

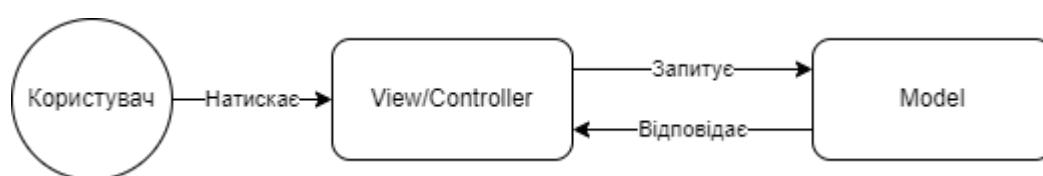


Рисунок 2.6 – Зображення функціонування MVC

MVP (Model-View-Presenter) – це вдосконалений MVC, різниця між патернами у додаванні між інтерфейсом та моделлю, таким проміжним шаром, як presenter. Presenter має посилання на view, та view має посилання на presenter і таким чином відбувається комунікація (рис. 2.7).



Рисунок 2.7 – Зображення функціонування MVP

MVVM (Model-View-ViewModel) – може бути схожим на MVP, тому що теж має щось схоже на presenter як viewModel, але зв'язок між view та viewModel відрізняється. На відміну presenter, viewModel не зберігає посилання на view. Комунікація відбувається через змінну state у класі viewModel на яку підписується view, а viewModel своєю чергою тільки оновлює цю зміну, про що

одразу ж view дізнається та може дістати потрібні дані. Цей підхід є найпоширенішим на цей час (рис. 2.8).

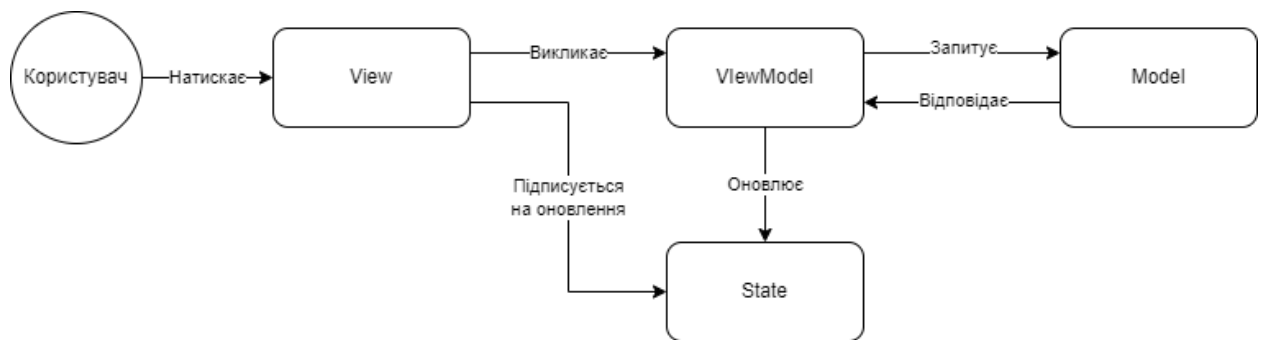


Рисунок 2.8 – Зображення функціонування MVVM

MVI (Model-View-Intent) – цей патерн вдосконалює патерн MVVM. З часом клас MVVM може сильно розростись, що зробить його важким для розуміння чи дебагінгу, на допомогу прийде патерн MVI, головною перевагою є односпрямований потік даних, який дозволяє дуже легко розуміти де і що відбувається. Але є і недоліки: по-перше, це складність розуміння патерну, по-друге, вимушує писати багато шаблонного коду, що вимагає часу. Але не зважаючи на ці мінуси, цей патерн є кращим вибором для масштабування та підтримки проєкту (рис. 2.9) [7].

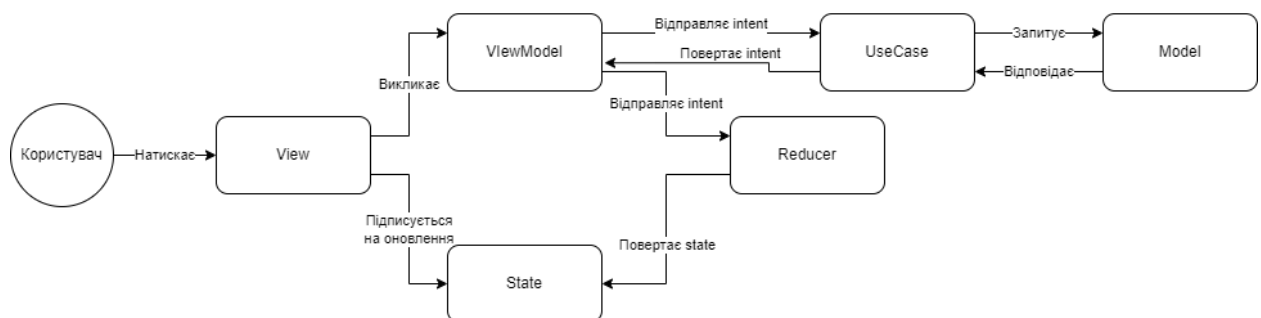


Рисунок 2.9 – Зображення функціонування MVC

Як бачите всі ці патерни еволюціонували один в одного, і тому наступний паттерн вирішував проблеми попереднього.

Для програми я вибрав MVI паттерн, тому що з ним легше відстежувати логіку, яку пишеш і легше додавати новий функціонал.

2.4 Вибір сховища для даних

У андроїду є вбудована імплементація бази даних SQLite, тому ми будемо використовувати її для зберігання даних з картки. Наявне спілкування з базою даних досить не зручне, тому нам треба вибрати зручніше для нас API. Я вибрав бібліотеку Room розробленої Гуглом, це API для роботи з базою даних SQLite. Перевага цього рішення, це можливість підписатися на оновлення даних у базі даних, зручне API щоб робити запити до бази даних, можливість міграції з різних версій баз даних, а також дуже зручно шифрувати і дешифрувати дані.

Для зберігання PIN коду користувача база даних SQLite буде не зручною, потрібно щось легше для зберігання пари ключ-значення. У цьому випадку є вбудоване рішення від Гугла як SharedPreferences якого достатньо, щоб покрити всі потреби.

2.5 Шифрування даних

Було вирішено не розробляти свою реалізацію алгоритмів, а використовувати вже наявні рішення з різних бібліотек.

2.5.1 Шифрування бази даних

Щоб зашифрувати дані в SQLite базі даних, нам потрібно буде додати популярну та перевірену бібліотеку SqlCipher.

SqlCipher – це криптографічна бібліотека розроблена компанією Zetetic LLC, ще у 2008 році й з того часу вона стала доволі популярною в Android розробці. Бібліотека використовує найпоширеніший алгоритм шифрування 256 бітовий AES алгоритм з CBC режимом, що відповідає переможцю з раніше

порівнянних алгоритмів шифрування. Бібліотека не має своєї реалізації шифрування, для цього використовуються сторонні бібліотеки. Під час налаштування шифрування, ми маємо передати байтовий пароль. Після передачі пароля, система SqlCipher генерує ключ за допомогою стандарту формування ключа PBKDF2-HMAC-SHA512. Під капотом кожна сторінка в базі даних шифрується окремо, а розмір сторінки відповідає 4096 байтів, але розмір можна задати інший, 4096 байтів – значення за замовченням.

Байтовий пароль, який був згаданий вище, треба також десь зберігати, інакше зломисник зможе дешифрувати дані бази даних. І щоб безпечно зберігати пароль, нам потрібно буде використовувати Android KeyStore і бібліотеку Java Cipher.

Java Cipher – надає криптографічний функціонал шифрування і дешифрування (див. табл. 2.2).

Таблиця 2.2 – Підтримуваний функціонал Cipher

Алгоритми	Режими	Доповнення
AES	CBC	ISO10126
AESWrap	CCM	OAEP
ARCFOUR	CFB, CFB _x	PKCS1
Blowfish	CTR	PKCS5
DES	CTS	SSL3
DESede	ECB	-
DESedeWrap	GCM	-
ECIES	OFB, OFB _x	-
PBE	PCBC	-
RC2	-	-
RC4	-	-
RC5	-	-
RSA	-	-

Як можна побачити Java Cipher підтримує алгоритм AES, який нам був і потрібен.

Щоб змусити працювати Java Cipher та Android KeyStore разом, потрібно буде написати свою імплементацію для збереження паролів (KeyStoreHelper і PasswordProvider).

2.5.2 Шифрування SharedPreferences

Для шифрування PIN користувача ми будемо використовувати перевірену часом бібліотеку як EncryptedSharedPreferences, вона за нас шифрує дані. Більш того EncryptedSharedPreferences успадковується від вбудованого класу SharedPreferences, що зробить дуже зручним у разі якщо нам треба буде мігрувати зі звичайних SharedPreferences на EncryptedSharedPreferences.

З вибору алгоритмів EncryptedSharedPreferences підтримує тільки 256 бітовий AES алгоритм з режимами CTR для ключів і GCM для значень.

На відміну від вручну написаних класів для зберігання пароля, як це робиться для бази даних, тут все легше. Як і в першому випадку, пароль зберігається за допомогою Android KeyStore, але створення пароля та збереження в Android KeyStore робиться в бібліотеці під капотом.

І цих якостей більш ніж достатньо, щоб обрати EncryptedSharedPreferences для зберігання PIN коду та інших ключ-значення даних.

2.6 Вибір системи впровадження залежності (DI)

При розробці додатка, з'являється дуже багато залежностей і ініціалізація класу стає важкою працею, а тому є безперечно суттєвим використовувати щось, що змогли б полегшити ініціалізацію класів, і на це приходить такий патерн як DI.

DI – це патерн проектування націлений на спрощення ініціалізації класу, це досягається за допомогою написання інструкції ініціалізації компонентів. Існують різні варіації реалізації впровадження залежності:

- впровадження конструктора – у цій реалізації впровадження впроваджуються через конструктор класу;
- впровадження інтерфейсу – цей спосіб визначає реалізацію об’єкта залежно від інтерфейсу, тобто для різних інтерфейсів може бути різна ініціалізація;
- впровадження сетера – спосіб, де впровадження реалізується за допомогою сетерів об’єкта.

У андроїд розробці, є кілька поширених варіацій бібліотек що реалізують DI:

- Dagger 2;
- Dagger Hilt;
- Koin.

Dagger 2 – це один із найпоширеніших DI для Java, який добре інтегрується в андроїд розробці. Його відмінна риса, це генерація Java коду який заснований на анотаціях, своєю чергою цей згенерований код дає можливість побудувати граф залежностей, який формується під час компіляції. Наявність графа залежностей забезпечує безпеку на рівні компіляції. Ще серед плюсів варто зазначити такі: ефективність і оптимізація коду, можливість використання з Java та Kotlin кодом, модульність (робить код легко підтримуваним). З мінусів це: велика кількість шаблонного коду який треба писати, менша швидкість компілювання проєкту та складність вивчення.

Dagger Hilt – це надбудова над Dagger 2, вона має схожі недоліки та переваги, але є що відрізняється: краща інтеграція з андроїдом, менша кількість шаблонного коду та легше вивчити ніж Dagger 2. Також треба зазначити, що Google рекомендує цей підхід [7].

Koin – цей підхід значно відрізняється від Dagger 2 і Dagger Hilt, він не використовує кодогенерацію та не генерує граф залежностей, але він дуже

зручний, тому що використовує DSL для визначення залежностей. Хоч головна мова для бібліотеки, це Kotlin, він усе одно може підтримує Java. З плюсів це: мала кількість шаблонного коду (швидкість розробки), добре інтегрується з Kotlin (використовує переваги мови), легкий для вивчення, має низький рівень впливу на швидкість компілювання та модульність. З мінусів це: відсутність безпеки на рівні компіляції (при неправильній конфігурації, буде кидати помилки під час виконання програми), має гіршу ефективність коду ніж Dagger 2 [8].

Для мого додатка я обрав Koin як DI, тому що це найзручніше рішення з вищеперелічених систем впровадження залежностей. Ефективність яку пропонує Koin є достатньою для потреб застосунку.

2.7 Вибір інструменту для створення інтерфейсу користувача.

Андроїд використовує за замовченням вбудований інструмент XML Layouts, але нещодавно з'явився новий підхід у розробці UI як Jetpack Compose. Він був анонсований ще у 2019 році на Google I/O, а вже у 28 липня 2021 року була випущена перша стабільна версія бібліотеки [9].

Розробка інтерфейсу користувача з XML Layouts здійснюється на мові XML, де ми створюємо view (базовий клас всіх візуальних елементів). Існує два типу елементів контейнер і контент, комбінація вкладень цих елементів створює UI. XML код компілюється у Java під час компіляції, який використовується андроїд системою для відображення UI. Цей підхід є легким для вивчення і використовувався довгий час, але він має свої недоліки: по-перше, це складність розробки складних view, по-друге, потребує багато шаблонного коду – що зрештою споживає час.

Jetpack Compose – дозволяє розробляти UI за допомогою декларативного синтаксису на основі Kotlin. Бібліотека використовує реактивну модель програмування для оновлення composable (компонент в Jetpack Compose). Оновлення composable елементів здійснюється оновленням тільки потрібних нам

елементів, на відміну XML Layouts де ми оновлюємо цілий екран заради однієї view.

Переваги Jetpack Compose над XML Layouts:

- Kotlin синтаксис;
- реактивна модель програмування;
- ефективність та оптимізація;
- швидкість розробки;
- зрозумілість коду;
- легше розробляти складні UI елементи;
- Google популяризує цей метод розробки.

Переваги XML Layouts над Jetpack Compose:

- легко вивчити;
- менш критичні помилки в коді;
- стабільніше на даний час.

Я обираю Jetpack Compose, тому що у моєму додатку я маю складний UI елемент, також додаток має Google дизайн Material 3 і з цією бібліотекою легше імплементувати його, і не можна не зазначити, що Jetpack Compose найкраще працює з MVI.

2.8 Оптимізація та безпека APK

Під час створення реліз APK ми будемо використовувати R8. R8 – це розроблений Гуглом оптимізатор і обфускатор коду. Вперше він був за замовченням встановлений в Android Studio 3.4, ще 17 квітня 2019 року, прийшовши на заміну старому обфускатору ProGuard. R8 працює швидше ніж ProGuard і виходить APK меншим розміром.

Головні функції R8:

- обфускація коду;
- видалення класів і функцій, що не використовуються;
- оптимізація коду;
- видалення ресурсів, що не використовуються.

3 РОЗРОБКА ПРОГРАМИ

3.1 Аналіз вимог до застосування

3.1.1 Концепція

Необхідно розробити додаток, що дозволяє користувачам зберігати дані карток.

Додаток допоможе мати користувачеві швидкий доступ до його карток, при цьому зберігання карток є повністю безпечним.

3.1.2 Функціональні вимоги до системи

Користувач повинен мати можливість додати банківську картку вручну.

Користувач повинен мати можливість заповнити дані скануванням картки.

Користувач повинен мати можливість редагувати картку.

Користувач повинен мати можливість видалити картку.

Користувач повинен мати можливість налаштувати зовнішній вигляд картки.

Користувач повинен мати можливість встановити PIN код.

Користувач повинен мати можливість розблокувати додаток за допомогою відбитка пальця.

Користувач повинен мати можливість скинути PIN код.

Система повинна блокувати розблокування на 30 секунд, якщо користувач зробив 5 невдалих спроб авторизації за допомогою вводу PIN коду.

Система повинна блокувати розблокування за допомогою відбитка пальця на 30 секунд, якщо користувач зробив 5 невдалих спроб авторизації за допомогою відбитка пальця.

Система повинна мати валідацію даних картки, що вводяться.

Користувач повинен мати можливість дати скопіювати номер картки.

Система повинна шифрувати дані, що зберігаються.

Вихідний код системи повинен бути обфускованим.

Система повинна забороняти можливості перегляду внутрішньої пам'яті додатка.

3.1.3 Специфікація прецеденту «Розблокування додатка»

У таблиці 3.1 наводиться специфікація прецеденту «Розблокування додатка».

Таблиця 3.1 – Специфікація прецеденту «Розблокування додатка»

Основний виконавець	Користувач
Зацікавлені особи	Додаток
Передумова	Користувач задав PIN код у додатку, у девайса є сканер відбитка пальця
Постумова	Користувач успішно розблокував додаток, та отримав доступ до збережених карток
Основний сценарій	A1. Користувач відкриває додаток. A2. Користувач: а. вводить PIN код; б. сканує відбиток пальця. A3. Система розблоковує додаток.

Кінець таблиці 3.1

Основний виконавець	Користувач
Альтернативні сценарії	В1. Якщо користувач не задавав PIN код: а. користувач вводить новий PIN код і натискає «Далі»; б. користувач підтверджує PIN код повторним введенням і натискає «Підтвердити»; с. користувач розблоковує додаток. В2. Якщо після кроку А2.а, користувач ввів невірний PIN код: а. користувачеві надається 4 додаткові спроби; б. після 5 невдалих спроб, система блокує ввід PIN коду на 30 секунд. В3. Якщо після кроку А2.б, користувач мав невдалу спробу розблокувати додаток за допомогою сканування відбитка пальця: а. користувачеві надається 4 додаткові спроби; б. після 5 невдалих спроб, система повертає помилку і блокує сканування відбитка пальця на 30 секунд. В4. Якщо користувач не пам'ятає PIN код, то він може його скинути: а. користувач натискає «Не пам'ятаю PIN код»; б. система відображає діалог із попередженням про видалення даних; с. користувач натискає «Так, видалити всі дані»; д. далі повторити дії у В1.

3.1.4 Специфікація прецеденту «Додавання картки у додаток»

Таблиця 3.2 – Специфікація прецеденту «Додавання картки у додаток»

Основний виконавець	Користувач
Зацікавлені особи	Додаток
Передумова	Користувач розблокував додаток
Постумова	Користувач успішно додав картку, і може її бачити в каруселі карток

Кінець таблиці 3.2

Основний сценарій	A1. Користувач знаходиться на головному екрані додатка. A2. Користувач натискає на кнопку «+». A3. Система показує діалогове вікно додавання карток. A4. Користувач вводить дані картки. A5. Користувач натискає на кнопку «Зберегти». A6. Система шифрує дані картки та завантажує в базу даних. A7. Система ховає діалогове вікно. A8. Система автоматично витягує і дешифрує дані. A9. Користувач бачить додану нову картку в каруселі карток.
Альтернативні сценарії	B1. Після кроку A4, якщо користувач ввів некоректні дані, то кнопка «Зберегти» буде заблокована, до моменту коли дані будуть введені коректно.

3.2 Архітектура і структура додатка

Додаток дотримується Чистої Архітектури (Clean Architecture), а тому розбитий на модулі по шарах [10].

Переваги розбиття на модулі:

- поділ логіки за шарами;
- пере-використання коду;
- обмеження доступу до файлів;
- поліпшує тестування;
- незалежність.

Модулі додатка:

- а) додаток (app) – головний модуль, який зв'язує всі інші модулі;
- б) презентація (presentation) – група UI модулів, з якими взаємодіє

користувач:

- 1) блокування (auth) – модуль, який відповідає за надавання доступу до додатка користувачеві;

2) мій гаманець (my wallet) – у цьому випадку головний UI модуль, де ми можемо додавати, редагувати, видаляти та переглядати картки;

в) домен (domain) – модуль, що відповідає за бізнес-логіку додатка, він зв'язує UI модуль з модулем для даних;

г) дані (data) – модуль для взаємодії з даними. Модуль, як правило, має відповідальність за локальні (бази даних) чи віддаленні (сервер) запити, у цьому випадку, це тільки робота з базами даних;

г) ядро (core) – група модулів, що об'єднує модулі, які можуть бути використані в інших модулях:

1) компоненти (components) – модуль для загальних UI елементів;

2) загальне (common) – модуль для якихось корисних класів, функцій чи констант, які можуть бути перевикористані у різних модулях, які не пов'язані між собою за логікою;

3) теми (theme) – модуль, що відповідає за тему додатка, він містить в собі кольори та константи для розмірів;

4) навігація (navigation) – модуль, що містить навігацію між екранами та логіку допоміжних класів для навігації;

5) розпізнавання карток (card recognition) – модуль, що відповідає за логіку розпізнавання карток за допомогою камери.

Додаток має модулі, де кожний з них може мати залежності до інших модулів (рис. 3.1).

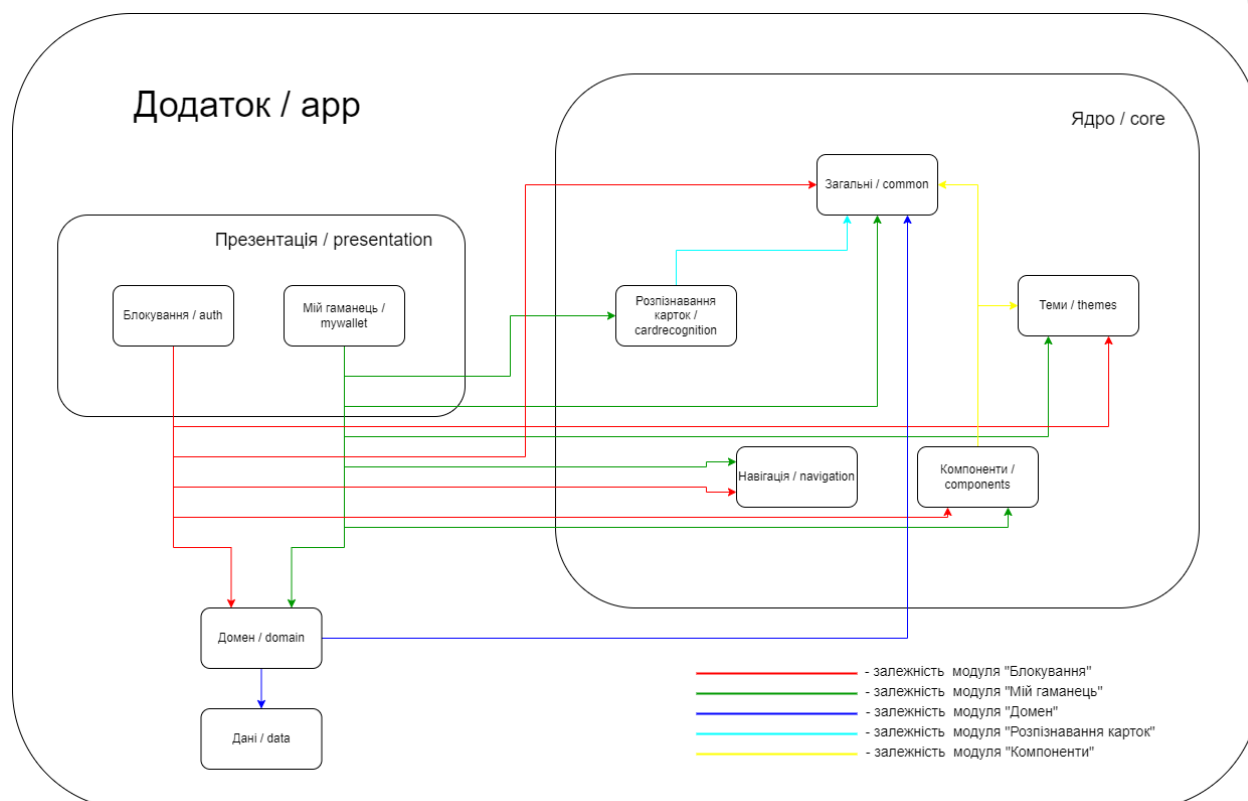


Рисунок 3.1 – Зображення залежностей модулів додатка

3.3 Проєктування інтерфейсу

3.3.1 Дизайн система і локалізація

Інтерфейс буде написаний за допомогою Jetpack Compose, як вже було зазначено у пункті 2.7, також там було зазначено, що системою дизайну буде Material 3.

Material 3 – це дизайн система створена Гуглом з відкритим вихідним кодом. 22 жовтня 2022 року Jetpack Compose бібліотека, що підтримує Material 3, була вперше оголошена стабільною [11]. Відмінна риса дизайн системи для андроїд додатка, полягає у динамічній палітрі кольорів теми відповідно до системних кольорів, але треба зазначити, що цей функціонал доступний тільки починаючи з Android 12 (рис. 3.2).

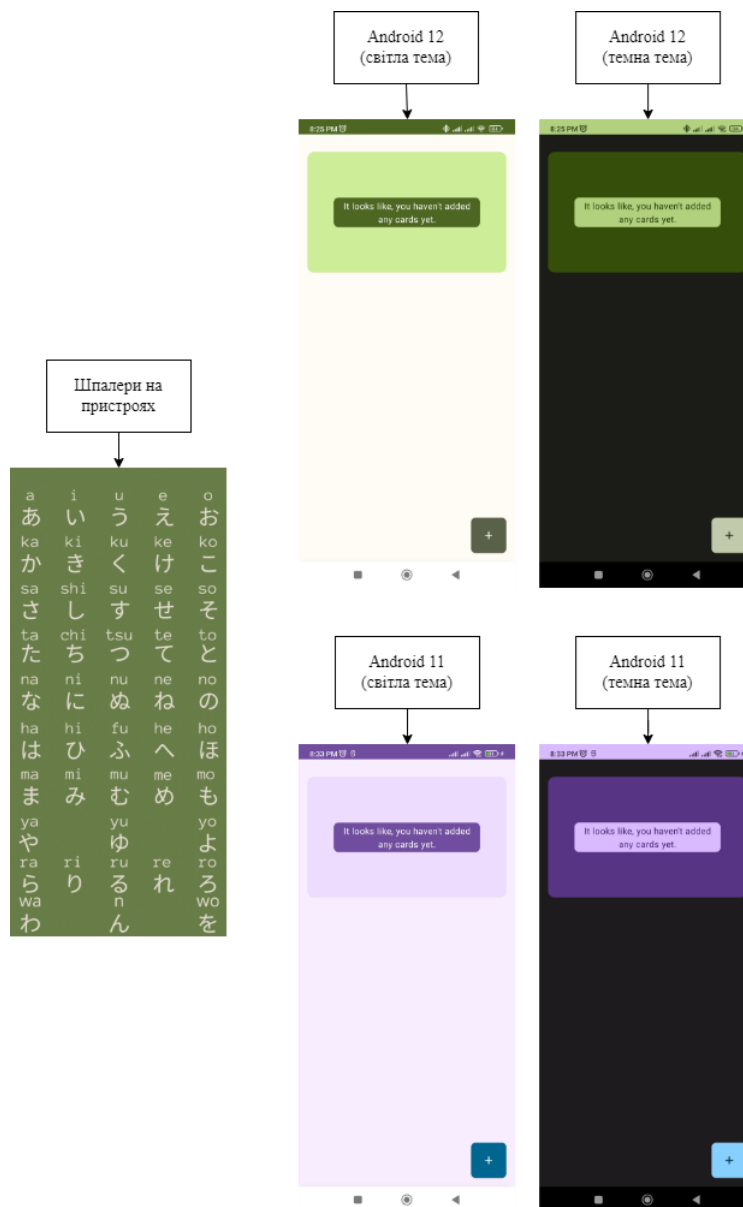


Рисунок 3.2 – Material 3 на різних версіях андроїда і з різними темами

Як можна побачити, на Android з версією 12, палітра динамічна – тема додатка збігається з кольорами шпалер. На Android 11, інша ситуація – ми використовуємо прописані попередньо кольори (див. табл. 3.3).

Таблиця 3.3 – Палітра кольорів для Android 11 і нижче

Android 11 >= (світла)	Android 11 >= (темна)
<pre> 5 /** Material 3 light colors palette */ 6 val LightPrimary = Color(color: 0xFF714D9F) 7 val White = Color(color: 0xFFFFFFFF) 8 val LightPrimaryContainer = Color(color: 0xFFEEDCFF) 9 val LightOnPrimaryContainer = Color(color: 0xFF2A0054) 10 val LightSecondary = Color(color: 0xFF00658F) 11 val LightSecondaryContainer = Color(color: 0xFFC7E7FF) 12 val LightOnSecondaryContainer = Color(color: 0xFF001E2E) 13 val LightTertiary = Color(color: 0xFF006A64) 14 val LightTertiaryContainer = Color(color: 0xFFFFDAD6) 15 val LightOnTertiaryContainer = Color(color: 0xFF00201D) 16 val LightError = Color(color: 0xFFB1A1A1) 17 val LightErrorContainer = Color(color: 0xFFFFDAD6) 18 val LightOnErrorContainer = Color(color: 0xFF410002) 19 val LightBackground = Color(color: 0xFFFFE0FF) 20 val LightOnBackground = Color(color: 0xFF24005A) 21 val LightSurface = Color(color: 0xFFFFE0FF) 22 val LightOnSurface = Color(color: 0xFF24005A) 23 val LightOutline = Color(color: 0xFF7B757F) 24 val LightSurfaceVariant = Color(color: 0xFFE8E0EB) 25 val LightOnSurfaceVariant = Color(color: 0xFF4A454E) </pre>	<pre> 27 /** Material 3 dark colors palette */ 28 val DarkPrimary = Color(color: 0xFFD9B9FF) 29 val DarkOnPrimary = Color(color: 0xFF401B6D) 30 val DarkPrimaryContainer = Color(color: 0xFF583485) 31 val DarkOnPrimaryContainer = Color(color: 0xFFEEDCFF) 32 val DarkSecondary = Color(color: 0xFF86CFFF) 33 val DarkOnSecondary = Color(color: 0xFF00344C) 34 val DarkSecondaryContainer = Color(color: 0xFF004C6D) 35 val DarkOnSecondaryContainer = Color(color: 0xFFC7E7FF) 36 val DarkTertiary = Color(color: 0xFF500BCF) 37 val DarkOnTertiary = Color(color: 0xFF003733) 38 val DarkTertiaryContainer = Color(color: 0xFF00504B) 39 val DarkOnTertiaryContainer = Color(color: 0xFF71F7EB) 40 val DarkError = Color(color: 0xFFFFB4AB) 41 val DarkOnError = Color(color: 0xFF690005) 42 val DarkErrorContainer = Color(color: 0xFF93000A) 43 val DarkOnErrorContainer = Color(color: 0xFFFFDAD6) 44 val DarkBackground = Color(color: 0xFF1D1B1E) 45 val DarkOnBackground = Color(color: 0xFFE7E1E5) 46 val DarkSurface = Color(color: 0xFF1D1B1E) 47 val DarkOnSurface = Color(color: 0xFFEADDFF) 48 val DarkOutline = Color(color: 0xFF958E99) 49 val DarkSurfaceVariant = Color(color: 0xFF4A454E) 50 val DarkOnSurfaceVariant = Color(color: 0xFFCC4CF) </pre>

Стосовно локалізації, то додаток підтримує дві мови, це українська та англійська. Вибір мови відбувається у системних налаштуваннях.

3.3.2 Розбір екранів

Якщо подивитися у код, то ми маємо тільки два екрани: «Блокування» (рис. 3.3) і «Каруселька карток» (рис. 3.4), але вони мають різні стани та можливість відкривати спливаючі вікна.

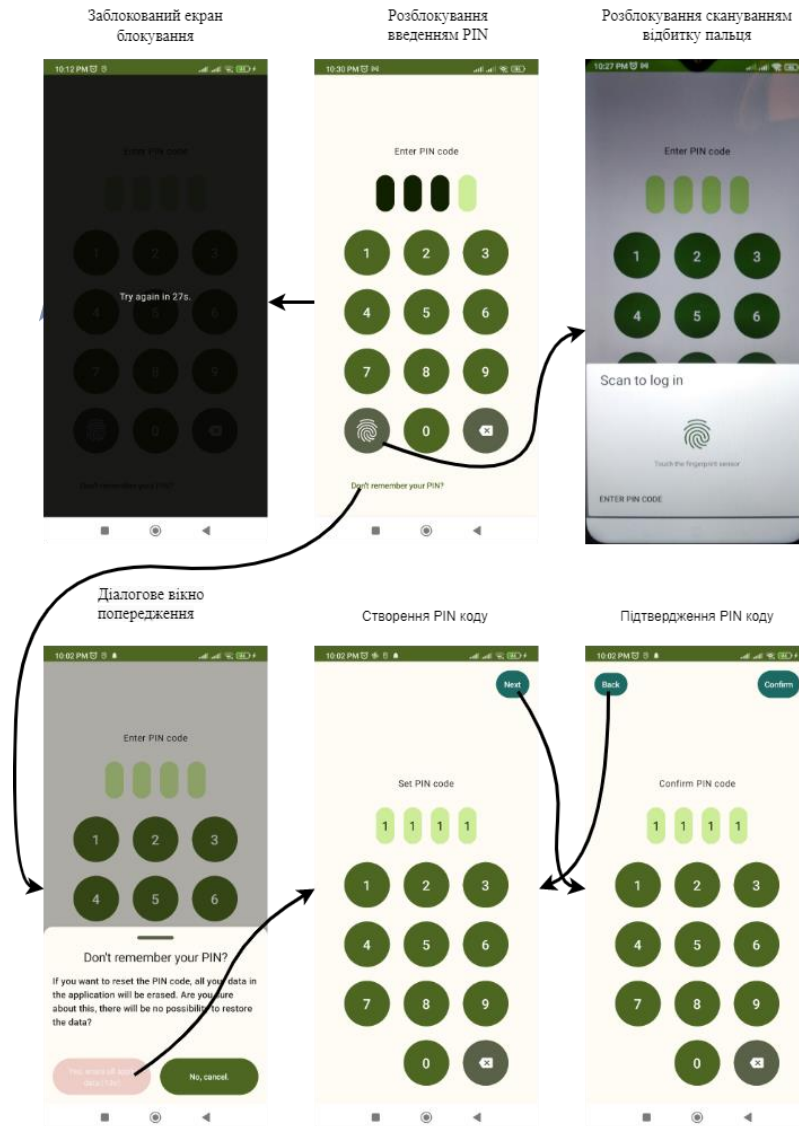


Рисунок 3.3 – Зображення навігації екрану «Блокування»

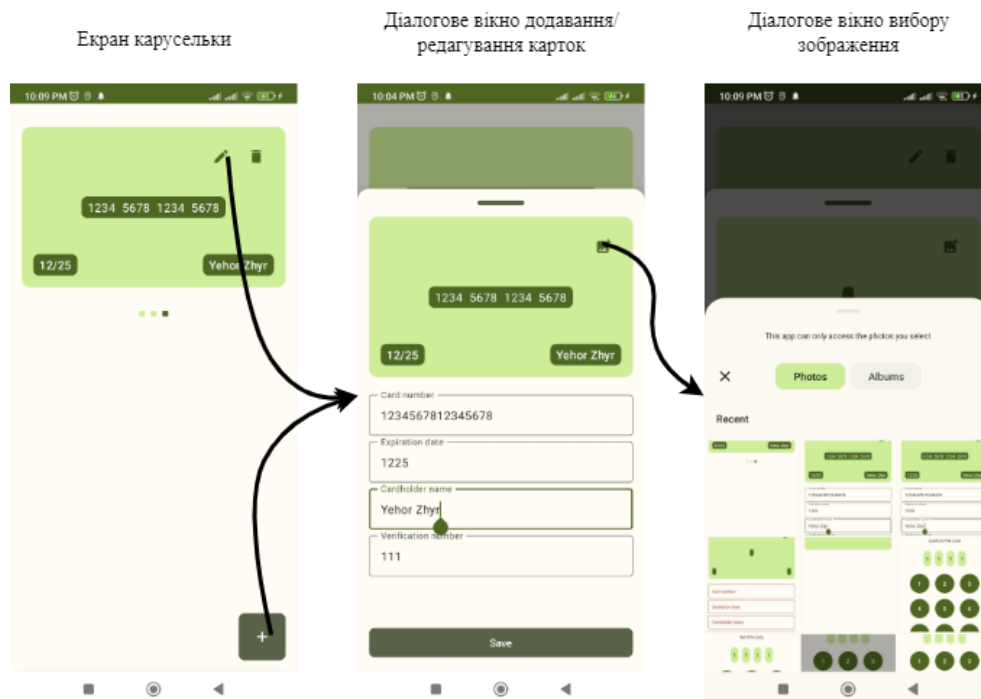


Рисунок 3.4 – Зображення навігації екрану «Каруселька карток»

У наступних частинах буде розбір кожного екрана та їх компонентів.

3.3.2.1 Розбір екрана «Блокування»

Екран «Блокування» представляє перший екран який ви побачите при відкритті додатка. Він відповідає за захист доступу до додатка.

На екрані «Блокування» у нас є два потоки: «розблокування» і «створення PIN коду». Кожен потік має свою функціональність і свої стани.

3.3.2.2 Потік «Розблокування»

Потік «Розблокування» має стан зображений на рис. 3.5.

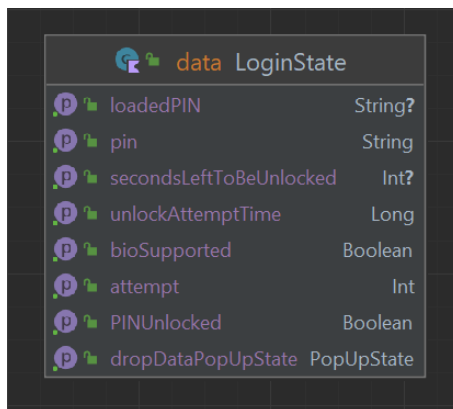


Рисунок 3.5 – Стан потоку «Розблокування»

Потік «розблокування» має таку функціональність:

- розблокування за допомогою відбитка пальця. Для цього я розробив обгортку для гуглівського API, який допомагає працювати з бібліотекою. Після виклику функції на відображення спливаючого вікна, ми перевіряємо на перебільшення помилкових спроб, і якщо ми перебільшили їх, то ми відображаємо помилку (рис. 3.6), якщо ні, то бібліотека відображає спливаюче вікно (рис. 3.7);

- розблокування за допомогою введення PIN коду. Ця функціональність містить в собі циферблат, який у лівому нижньому куту відображає кнопку для вибору методу входу через сканування відбитка пальця (якщо пристрій не підтримує цей метод входу, ми ховаємо цю кнопку), у правому куту має кнопку «стерти», усі інші кнопки – це цифри. Щоб підраховувати кількість введених символів, було написано спеціальне поле для вводу чотиризначного значного PIN коду, при вводі коду, система не відображає введені цифри, тільки їх кількість. Якщо користувач ввів 5 раз невірний код, його екран блокується на 30 секунд (рис. 3.8);

- скидання PIN коду. Якщо користувач не може розблокувати додаток і все одно хоче продовжувати користуватися додатком, то він має можливість скинути PIN код, для цього у самому низу є кнопка «Не пам’ятаєте ваш PIN код?», потрібно натиснути на неї. Після цього відкриється спливаюче вікно, яке попереджає, що всі дані будуть видалені, якщо продовжити. Вікно попередження

має дві кнопки: «Ні, скасувати» та «Так, я хочу видалити дані додатка». Якщо натиснути на першу кнопку, то вікно попередження ховається. Щоб натиснути на другу кнопку слід вчекати 15 секунд (це було зроблено, щоб запобігти випадкового і необдуманого натискання), в разі натискання, то всі дані видаляються, вікно попередження ховається та система змінює «розблокування» на «створення PIN коду».

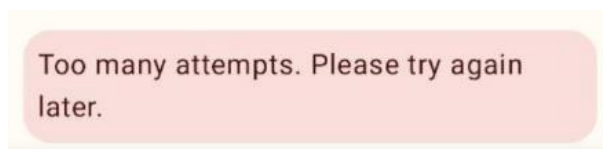


Рисунок 3.6 – Повідомлення перевищенням спроб

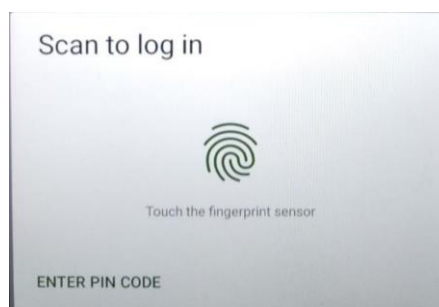


Рисунок 3.7 – Діалогове вікно для сканування відбитка пальця

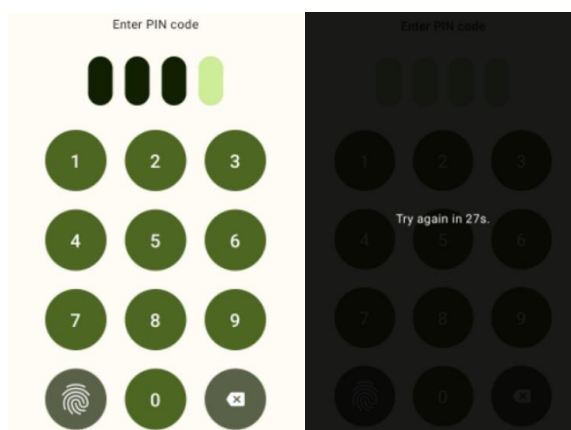


Рисунок 3.8 – Розблокування з PIN і блокування екрана

3.3.2.3 Потік «Створення PIN коду»

Потік «Створення PIN коду» має стан зображений на рис. 3.9

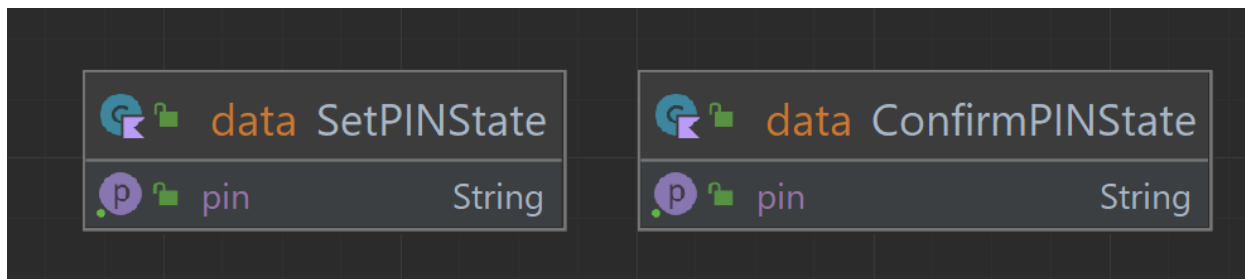


Рисунок 3.9 – Стан потоку «Створення PIN коду»

Потік «створення PIN коду» має таку функціональність:

- вигадування PIN коду. У цьому кроці користувач має циферблат, на якому додатково розташована у правому нижньому куті кнопка «стирання». За для відображення введеного PIN коду, система пере-використовує поле вводу зі стану «розблокування», але у цьому випадку система передає параметром дозвіл на відображення введених цифр. У самому правому верхньому куті ми маємо кнопку «далі», яка за замовченням заблокована, щоб її розблокувати, треба, щоб поле відображення цифр, було заповнене (рис. 3.10);
- підтвердження PIN коду. Інтерфейс трохи відрізняється від останнього кроку. У лівому верхньому куту додалась нова кнопка «назад», яка повертає користувач на попередній крок. У правому верхньому куту змінилася кнопка «далі» на «підтвердити», але правило розблокування її не змінилось (рис. 3.11). Якщо введений код не відповідає коду з минулого кроку, то користувач отримує помилку (рис. 3.12). У випадку успішного підтвердження, користувача переміщає до екрана «Каруселька карток».

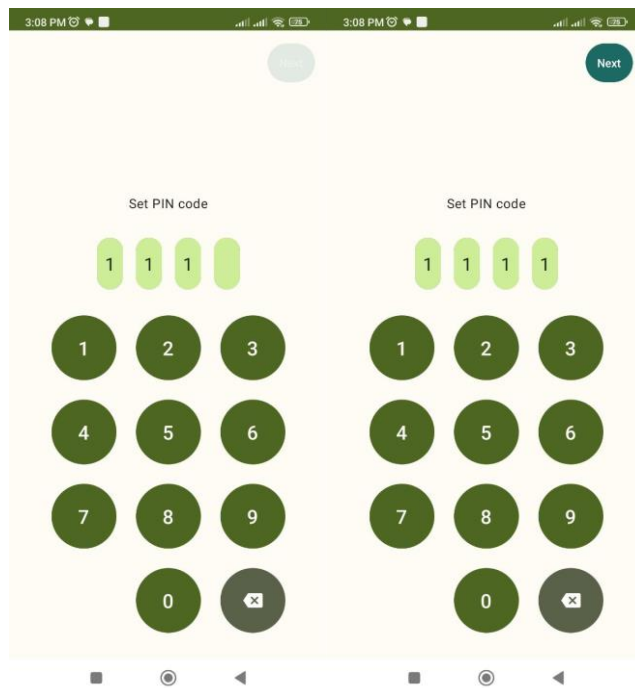


Рисунок 3.10 – Крок створення PIN коду

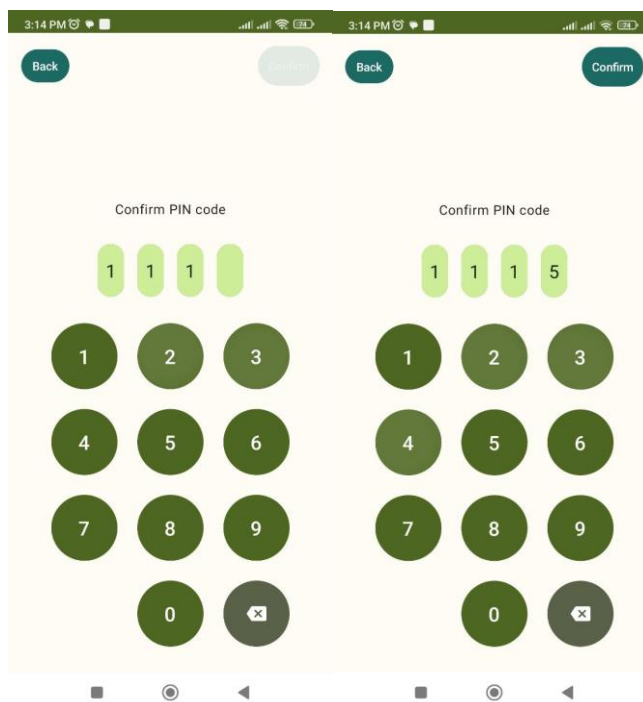


Рисунок 3.11 – Крок підтвердження PIN коду

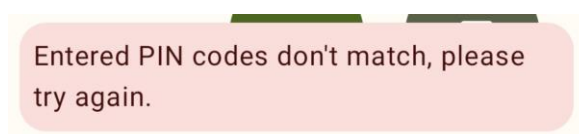


Рисунок 3.12 – Повідомлення про незбіжність PIN кодів

3.3.2.4 Розбір екрана «Каруселька карток»

Екран «Каруселька карток» – це головний екран де ми відображаємо картки користувача. Екран має лише один потік, але він має багато UI елементів. Стан екрану зображений на рис. 3.13.

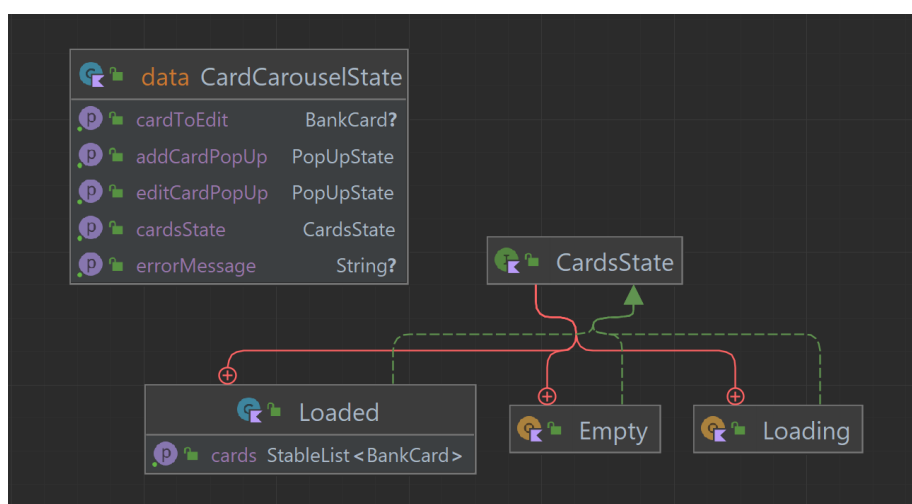


Рисунок 3.13 – Стан екрану «Каруселька карток»

3.3.2.5 UI елемент «Каруселька»

Ця каруселька має декілька станів: перший, це завантаження, другий – пустий і третій – користувач завантажив картки.

У першому випадку користувач перейшов до екрана і відбувається завантаження карток. Система відображає макет картки з анімації у вигляді переливання полів (рис. 3.14).

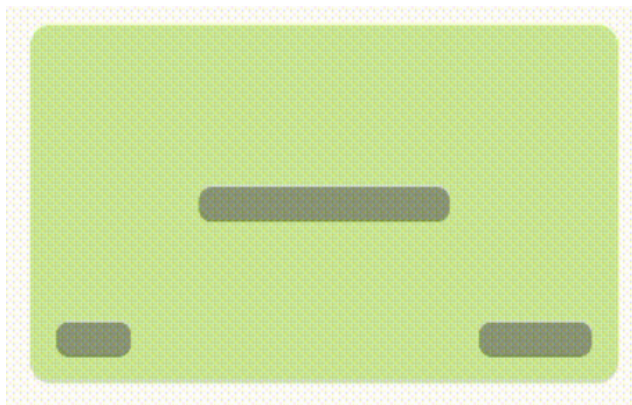


Рисунок 3.14 – Стан під час завантаження даних «Карусельки»

У другому випадку система відображає пустий макет картки, на якому написано, що карток нема (рис. 3.15).

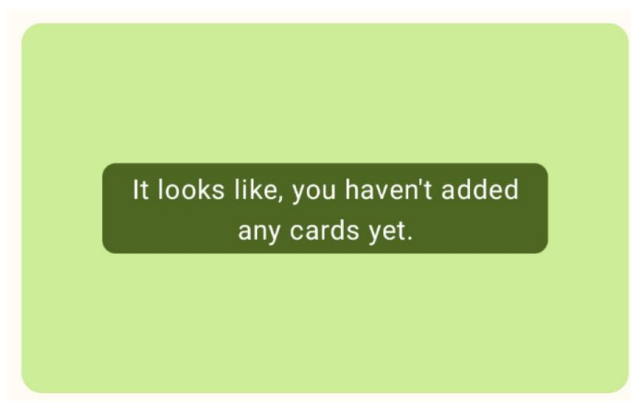


Рисунок 3.15 – Пустий стан «Карусельки»

У третьому випадку ми відображаємо горизонтальний список карток (рис. 3.16). Для того, щоб відобразити кількість і поточну позицію, система використовує спеціальний індикатор, який знаходиться під списком карток. Щоб гортати список, треба зробити горизонтальний свайп (провести пальцем по екрану). Також присутня анімація під час гортання списку (рис. 3.17). Якщо затиснути картку, то можна скопіювати у буфер обміну номер картки, після копіювання пройде вібрація, яка є індикатором успішного копіювання.

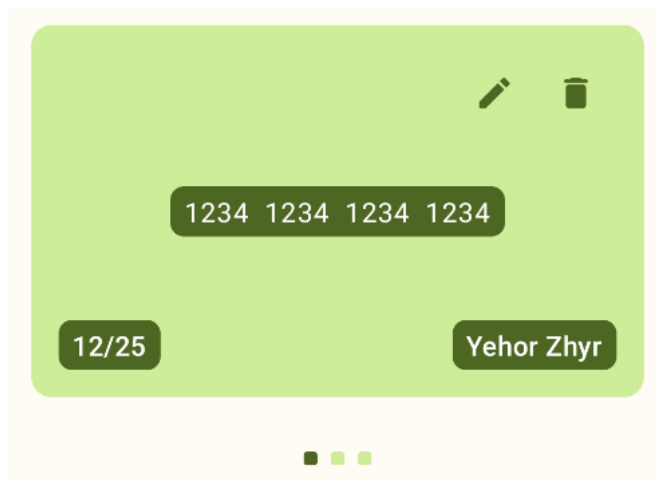


Рисунок 3.16 – Стан завантажених даних «Карусельки»

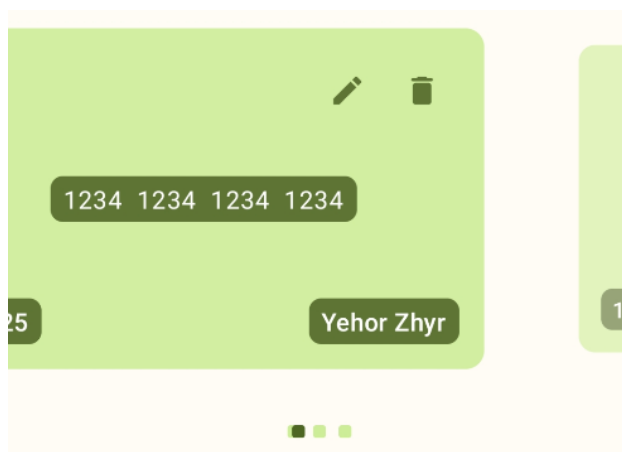


Рисунок 3.17 – Анімація гортання «Карусельки»

3.3.2.6 UI елемент «Картка»

«Картка» є доволі складним UI елементом, який буде пере-використовуватися в декількох UI елементах. А тому розробка цього елемента зайняла найбільший час, якщо порівнювати з іншими елементами. Структуру файлів елемента «Картка» можна побачити на рис. 3.18.

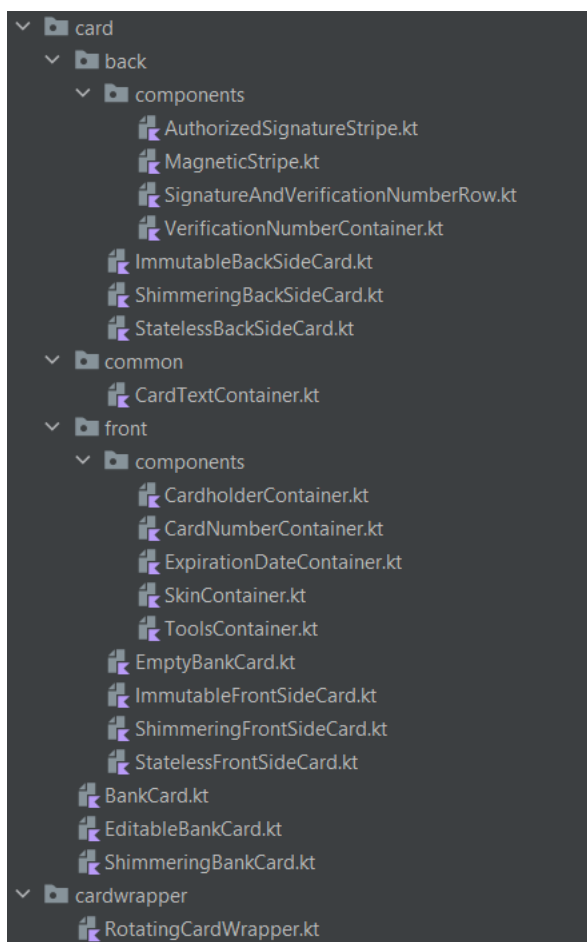


Рисунок 3.18 – Файлова структури елемента «Картка»

Картка має дві сторони: фронтальну та задню. Розміри й округлення картки відповідають європейським стандартам.

На фронтальній стороні, система відображає такі поля: номер картки, ім'я та прізвище власника, і термін придатності (рис. 3.19). Серед можливих кнопок, це: значок «Видалити» і «Редагувати» (рис. 3.20), також є випадок коли ми відображаємо значок «Додати малюнок» чи «Видалити малюнок».



Рисунок 3.19 – Фронтальна сторона елемента «Картка»

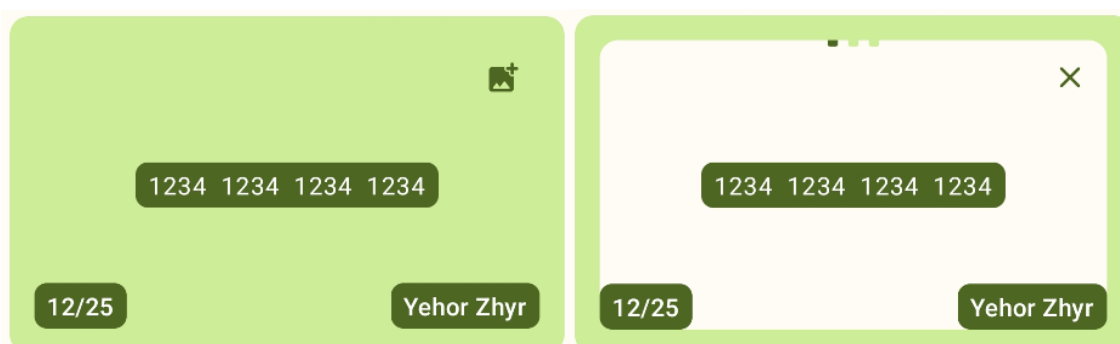


Рисунок 3.20 – Варіацією кнопок елемента «Картка»

Для того, щоб побачити іншу сторону, треба зробити два швидких натискання на картку, після цього почнеться анімація перевертання картки (рис. 3.21).



Рисунок 3.21 – Анімація перевертання елемента «Картка»

Задня сторона має поле для CVV, інші елементи це муляжі які повторюють текстуру реальної банківської картки, але ніякої функціональності не несуть (рис. 3.22).



Рисунок 3.22 – Задня сторона елемента «Картка»

3.3.2.7 UI елемент кнопка «Додавання картки»

Кнопка «Додавання картки» розташована у правому лівому куті (рис. 3.23).



Рисунок 3.23 – Кнопка «Додавання картки»

3.3.2.8 UI елемент спливаюче вікно для додавання картки

Спливаюче вікно для додавання має два стану: перше, це ручне додавання карток і друге – сканування картки. Перемикання між станами здійснюється за допомогою горизонтального свайпа.

У ручному додаванні, система відображає UI елементи вертикально, тож перший елемент, це «картка». Система пере-використовує вже розроблений елемент. Другий, це поля вводу: номер картки, термін придатності, ім'я власника та номер підтвердження (CVV). Важливо зазначити, що кожне поле має додатковий стан для помилки, поле переходить до помилкового стану у разі не

відповідності до правил валідації поля. При введенні даних у кожне поле дані відображаються не тільки в полях введення, а й в спеціально зазначених полях першого елемента «картка», якщо поле на картці знаходиться на зворотній стороні, то буде застосовано автоматичну анімацію перевертання. Третій елемент – кнопка для збереження даних, вона стає активною, якщо всі правила валідації були дотриманні (рис. 3.24).

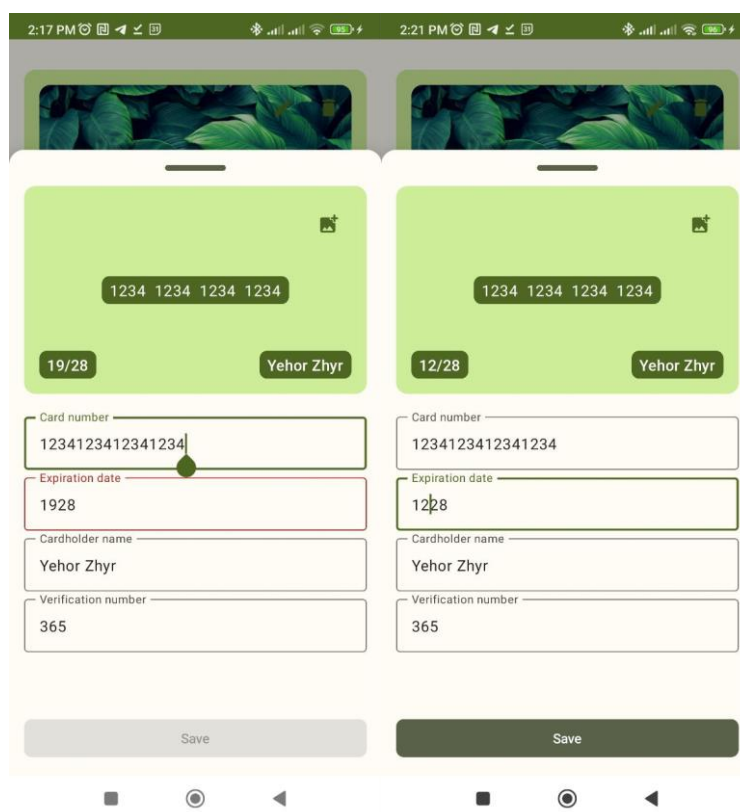


Рисунок 3.24 – Спливаюче вікно для додавання карток

Щоб редагувати вигляд картки (додати малюнок), користувач повинен натиснути на кнопку «Додати малюнок» (рис. 3.25), яка розташована у верхньому правому куті елемента «Картка», після натискання ми викликаємо спеціальне, рекомендоване Гуглом API, яке відкриє спливаюче вікно для вибору фотографії з галереї (рис. 3.26).

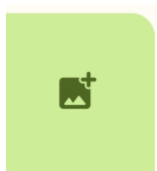


Рисунок 3.25 – Кнопка «Додати малюнок»

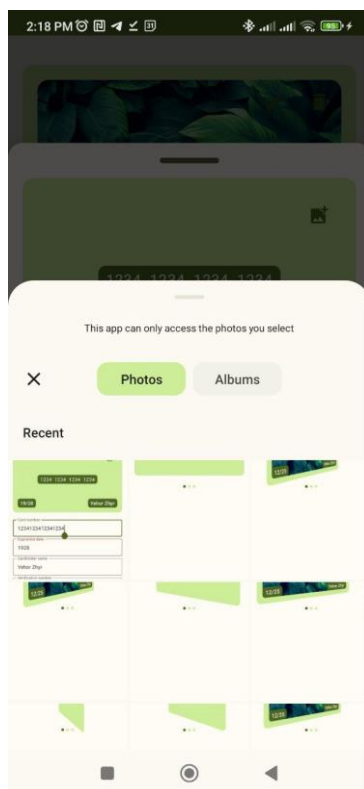


Рисунок 3.26 – Спливаюче вікно для вибору фотографії

Після вибору фотографії, вона буде застосована до картки (рис. 3.27).

Переваги способу з Google API, у тому, що нам не треба запитувати у користувача дозвіл на доступ до його фотографій, що є дуже чуткою темою для деяких користувачів. Більш того, цей спосіб є більш безпечним, тому що система має доступ тільки до тих фотографій, які обрав користувач [12].

Прибрати малюнок з картки, можна за допомогою кнопки «Видалити малюнок» яка прийде на заміну кнопці «Додати малюнок» (рис. 3.28).

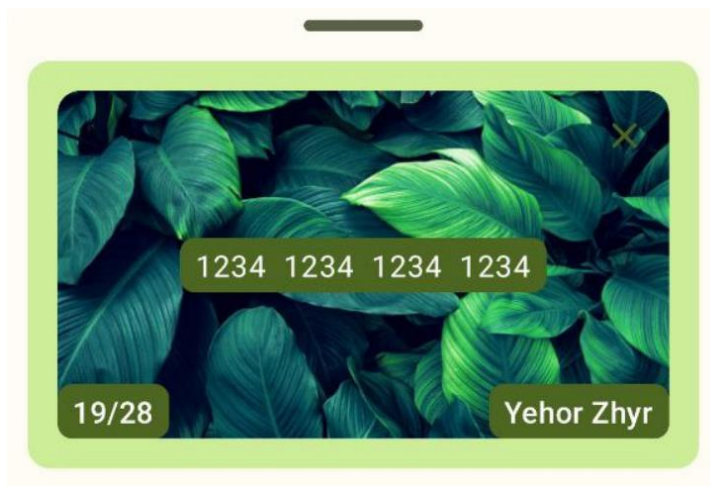


Рисунок 3.27 – Відображення застосованого малюнка до картки



Рисунок 3.28 – Кнопка «Видалити малюнок»

У стані додавання картки скануванням, система може заповнити тільки номер картки й термін придатності. Інші дані, користувач повинен заповнювати ручним способом.

При свайпі до цього стану, якщо користувач не дав дозвіл на використання картки, йому буде показано діалогове вікно, де розповідається навіщо додатку дозвіл на використання камери – це вважається гарним тоном, при запиті дозволів у користувача [13]. Якщо користувач не погодився, то система повертає його до ручного вводу. У разі погоди, Android показую йому вікно з запитом на дозвіл використання камери. Якщо користувач не дав дозвіл, йому знову показують вікно з описом причини, для чого це треба. Після того як система має дозвіл на використання камери, користувач може побачити виріз для камери у вигляді банківської картки по центру вікна (рис. 3.29). Після того як сканування завершиться, користувача буде повернуто до ручного вводу, поля номер карти і термін придатності будуть вже заповнені зісканованими даними.



Рисунок 3.29 – Додавання картки скануванням

3.3.2.9 UI елемент спливаюче вікно для редагування картки

Для того, щоб відредагувати карту, треба натиснути на кнопку «редагувати» на картці у правому верхньому куту.

Спливаюче вікно редагування має такий же інтерфейс як для додавання карток, крім того, що вікно має один стан – ручний, сканування карток не було додано, тому що при редагуванні не потрібна ця функція. Щоб почати редагувати треба натиснути на кнопку «Редагувати» в елементі «Картка»(рис. 3.30).

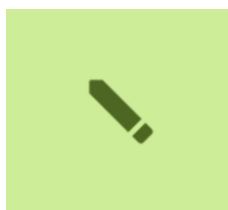


Рисунок 3.30 – Кнопка «Редагувати»

3.3.3 Захист від шпигунського програмного забезпечення

Може так статися, що додаток заражений вірусом, чи користувач випадково відкрив додаток під час транслювання екрана, і ці моменти загрожують безпеці даних користувача. Щоб уникнути цього, треба прописати одну команду в класі MainActivity, в onCreate() методі перед викликом setContentView():

```
window.setFlags(WindowManager.LayoutParams.FLAG_SECURE,  
WindowManager.LayoutParams.FLAG_SECURE)
```

Після цього можливість зробити скріншот чи відеозапис екрана будуть блокуватися (рис. 3.31 – 3.32).

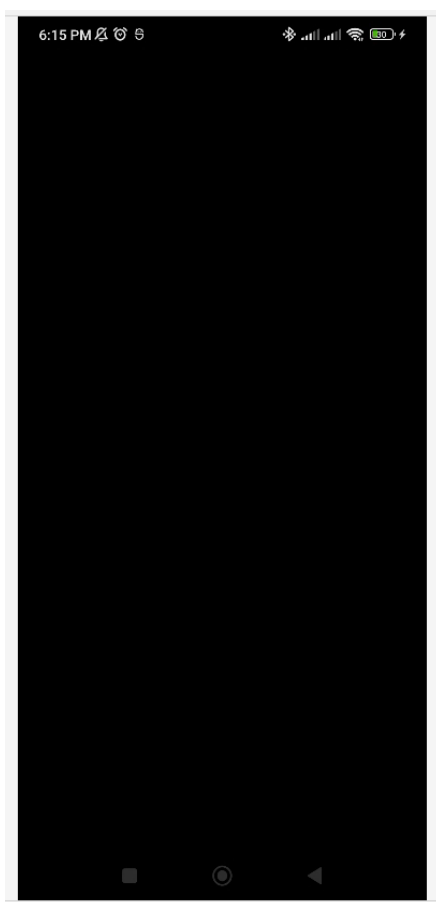


Рисунок 3.31 – Блокування запису екрану

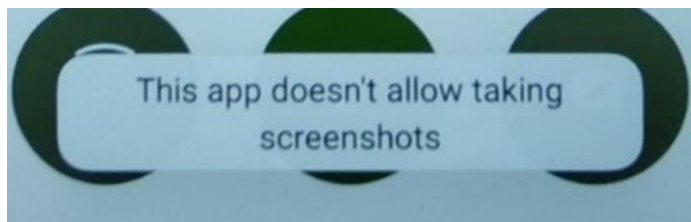


Рисунок 3.32 – Блокування знімків екрану

3.4. Проєктування даних

Як було затверджено у розділі 3.2, ми використовуємо Чисту Архітектуру, а тому додаток розділений за шарами, у цьому розділі будуть оглядатися шар дані і домен.

3.4.1 Огляд шару data

У додатку зберігається об'єкт `BankCardLocal`, це головна сутність додатка. Об'єкт має такі поля:

- термін придатності (`expirationDate`) – термін придатності картки, що має тип даних `string`, записується у 4 цифри: місяць і рік (12/24);
- номер (`number`) – номер картки, що має тип даних `string`, записується у 16 цифр (1234567812345678);
- посилання фотографії (`skinUri`) – має тип даних `string`, але також він може бути пустим (`null`), фотографія відображається на фронтальній стороні картки;
- ім'я власника (`cardHolderName`) – ім'я власника, має тип даних `string`
- ідентифікатор (`id`) – ідентифікатор картки, має тип даних `string`, створюється автоматично, під час додавання картки;
- CVV (`verificationNumber`) – має тип даних `string`.

Крім об'єкта `BankCardLocal`, у додатку зберігаються примітиви:

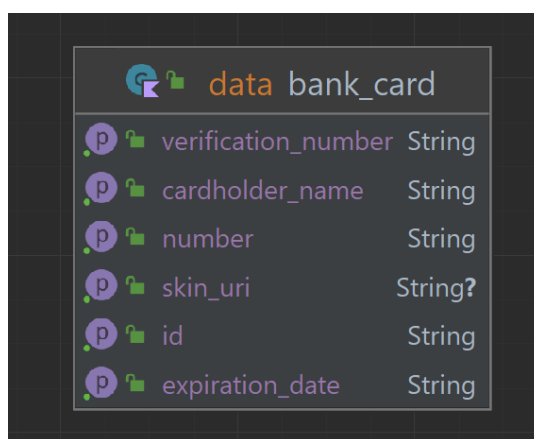
- PIN код (`pin`) – чотирьох значний код, який потрібен для розблокування додатка, зберігається у вигляді `string`;

- час розблокування (unlock_time) – час, коли можливість; розблокування буде знову доступна. Це поле потрібно, щоб уникнути можливості пропуску часу очікування. Зберігається як UNIX час у вигляді long;
- пароль від бази даних (password) – є приватним ключем для шифрування і дешифрування даних у базі даних, зберігається у вигляді масиву байтів.

3.4.1.1 База даних WalletDatabase

Структура таблиці bank_card

Структура таблиці повністю відповідає класу BankCardLocal (рис. 3.33).



data bank_card	
verification_number	String
cardholder_name	String
number	String
skin_uri	String?
id	String
expiration_date	String

Рисунок 3.33 – Структура таблиці bank_card

API таблиці bank_card

API таблиці відображенні на рис. 3.34.

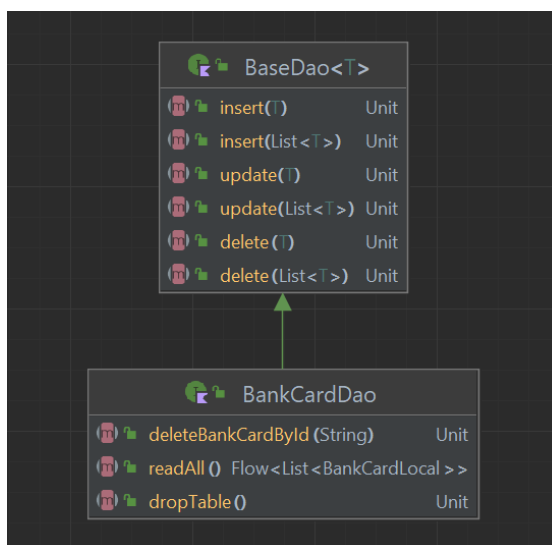


Рисунок 3.34 – Структура таблиці bank_card

Але з усіх цих методів, використовуються лише:

- insert(bankCard);
- update(bankCard);
- deleteBankCardById(bankCard.id);
- readAll(): Flow<List<BankCardLocal>>;
- dropTable().

Ініціалізація бази даних

Сутність BankCardLocal, зберігається у SQLite бази даних. Щоб ліпше було працювати з базою даних, ми використовуємо бібліотеку Room. Room сам по собі не шифрує дані, але при ініціалізації бази даних, бібліотека має необов'язковий метод для задання фабрики. Бібліотека SQLCipher, має фабрику для шифрування даних, і цю фабрику ми як раз передаємо в той метод. Метод про який йде мова, це openHelperFactory(), а фабрика – SupportFactory. Нижче наведено код ініціалізації бази даних.

```
private fun provideWalletRoomDatabase(context: Context, passwordProvider:
PasswordProvider): WalletDatabase {
    val password = passwordProvider.getPassword(fileName =
```

```

"wallet_db_password.bin")
    return Room.databaseBuilder(context, WalletDatabase::class.java,
WALLET_DATABASE_NAME).run {
        openHelperFactory(SupportFactory(password))
        build()
    }
}

```

У конструктор `SupportFactory`, передається ключ `password`, за яким буде шифруватися і дешифруватися дані. Цей ключ отримується з методу `getPassword` класу `PasswordProvider`.

Клас `PasswordProvider`

`PasswordProvider` має два методи, публічний, котрий система викликає, щоб отримати пароль (рис. 3.35). Пароль зберігається у зашифрованому вигляді у бінарному файлі `wallet_db_password.bin`, якщо цього файлу не існує, то система викликає другий метод, який генерує новий пароль, шифрує його та зберігає у новоствореному файлі. Шифрування і дешифрування виконується за допомогою класу `KeyStoreHelper` – він є залежністю для цього класу.

Код розробленого класу можна знайти у додатку А клас `PasswordProvider`.

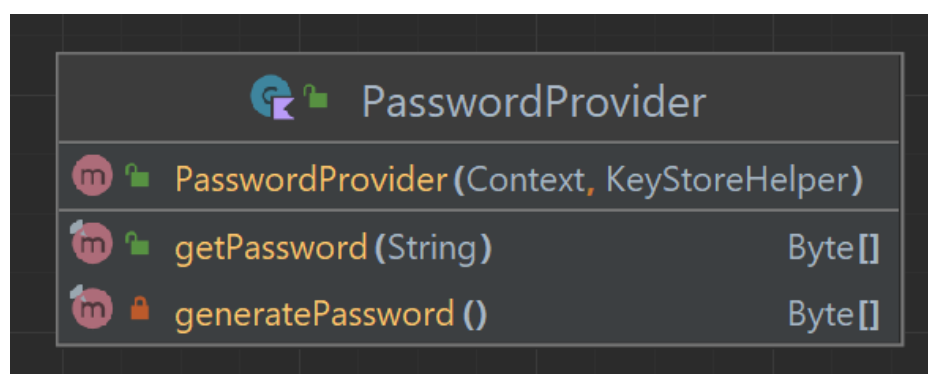


Рисунок 3.35 – API класу `PasswordProvider`

Клас KeyStoreHelper

Клас відповідає за шифрування і дешифрування даних з файлу (рис. 3.36). Шифрування потребує секретного ключа, але зберігати його у додатку не є безпечним варіантом, щоб розв'язати цю проблему існує нативний метод для збереження ключів Android Keystore System.

Android Keystore System – рішення для зберігання криптографічних ключів. Ключі зберігаються у відносно безпечній апаратній зоні на пристрої. Ця зона була спроектована таким чином, щоб бути стійкою до атак. Хоч це і вважається безпечним методом, але як і будь-яка система, вона має вразливості, якщо зломисник має root-права на пристрої, то він все ж таки зможе отримати доступ до даних.

Шифрування і дешифрування виконується за допомогою Java бібліотеки Cipher, вона найпоширеніша, перевірена часом та підтримує багато алгоритмів шифрування. Алгоритмом для шифрування був обраний AES, режим шифрування – CBC (ланцюгування шифроблоків), доповненням – PKCS7.

Код розробленого класу можна знайти у додатку А клас KeyStoreHelper.

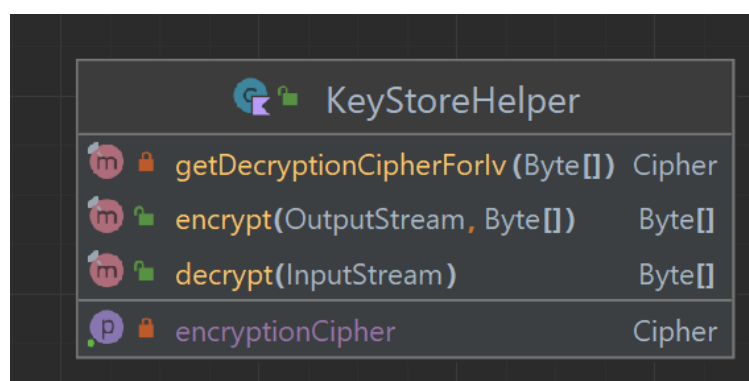


Рисунок 3.36 – API класу KeyStoreHelper

Сховище SecuredPrefs

SecuredPrefs – клас обгортка для роботи з EncryptedSharedPreferences.

API SecuredPrefs

API SecuredPrefs зображено на рис. 3.37.

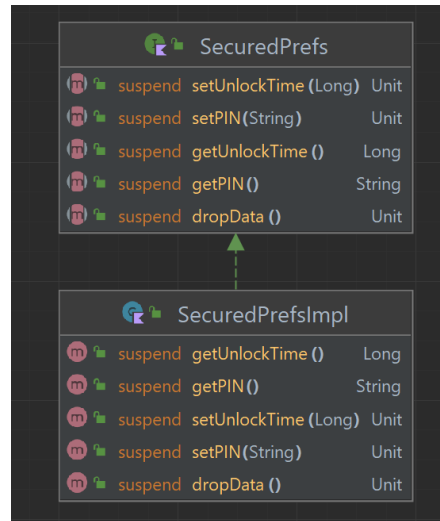


Рисунок 3.37 – API класу SecuredPrefs

Ініціалізація SecuredPrefs

Клас SecuredPrefs має лише одну залежність і це EncryptedSharedPreferences. Ініціалізував EncryptedSharedPreferences за рекомендацією Гугла, в основі якої лежить алгоритм шифрування AES [14]:

```

private fun provideEncryptedSharedPreferences(context: Context):
SharedPreferences {
    val masterKey = MasterKey.Builder(context)
        .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
        .build()

    return EncryptedSharedPreferences.create(
        context,
        "secured_user_prefs",
        masterKey,
        EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
        EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
    )
}
  
```

```

    )
}

```

3.4.2 Огляд шару domain

У цьому модулі маємо два репозиторії `UserDataRepository` і `BankCardRepository`.

Перший репозиторій відповідає за стан даних користувача, це його PIN код, час розблокування спроб чи видалення всіх даних (рис. 3.38). Реалізація інтерфейсу наведена у додатку А, клас `UserDataRepositoryImpl`.

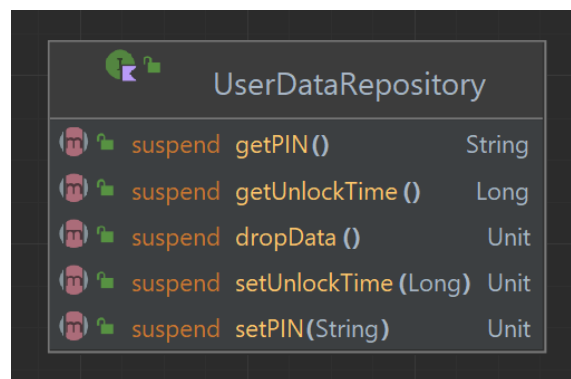


Рисунок 3.38 – API класу `UserDataRepository`

Другий відповідає за управління картками користувача (рис. 3.39). Реалізація інтерфейсу наведена у додатку А, клас `BankCardRepositoryImpl`.

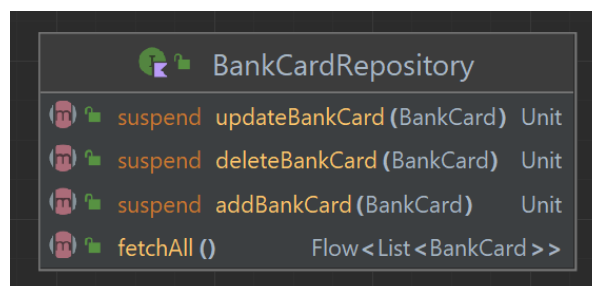


Рисунок 3.39 – API класу `BankCardRepository`

Як можна побачити у перевизначеному методі `addBankCard()` чи `fetchAll()`, система використовує функції розширення для перетворення `bankCard.toLocal()` чи `bankCard.toUi()`. Це також є одною з практик Чистої Архітектури для розмежувань дії класів даних за шарами. Якщо у presenter потрібен клас, який існує тільки в data модулі, то потрібно зробити такий же клас, але для presenter модуля. Особливо це корисно і зручно, коли presenter модуль маємо складний клас, з яким буде важко працювати у data модулі. Приклад з проєкту: у нас є presenter клас `BankCard`, який ми хочемо зберігати в таблиці, але SQLite не підтримує зберігання класу `Uri`, що є типом поля `skinUri`, і в цьому моменті дуже зручно зробити ще один клас спеціально для data модуля (рис. 3.40).

Property	Type
id	String
skinUri	Uri?
expirationDate	String
number	String
cardholderName	String
verificationNumber	String

Property	Type
expirationDate	String
number	String
skinUri	String?
cardholderName	String
id	String
verificationNumber	String

Рисунок 3.40 – Клас картки у domain і data модулі

І звісно треба додати функції перетворення між класами. Код функцій перетворень можна знайти у додатку А, клас `BankCard`, функція `BankCard.toLocal()` і функція `BankCardLocal.toUi()`.

Як вже було зазначено, додаток зберігає також примітиви, але для них нам не потрібно робити поділ по шарах.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ

4.1 Створення захищеного APK

Для створення захищеного APK треба додати R8 та вимкнути режим налагодження.

Щоб вимкнути режим налагодження, треба змінити переміну `isDebuggable` на `false` у `build.gradle.kts` (app) у секції тип збірки «випуск».

```
android {  
    ...  
    buildTypes {  
        getByName("release") {  
            isDebuggable = false  
        }  
    }  
    ...  
}
```

Щоб додати R8, треба в усіх модулях у файлу `build.gradle.kts` додати ці рядки.

```
android {  
    ...  
    buildTypes {  
        getByName("release") {  
            ...  
            isMinifyEnabled = true  
            proguardFiles(  
                getDefaultProguardFile("proguard-android-optimize.txt"),  
                "proguard-rules.pro"  
            )  
        }  
    }  
}
```

```
...
}
```

У модулі `build.gradle.kts (app)`, ще додати цей рядок:

```
android {
    ...
    buildTypes {
        getByName("release") {
            ...

            isShrinkResources = true

            ...
        }
    }
    ...
}
```

Після цього треба змінити тип збірки на «випуск» (рис. 4.1).

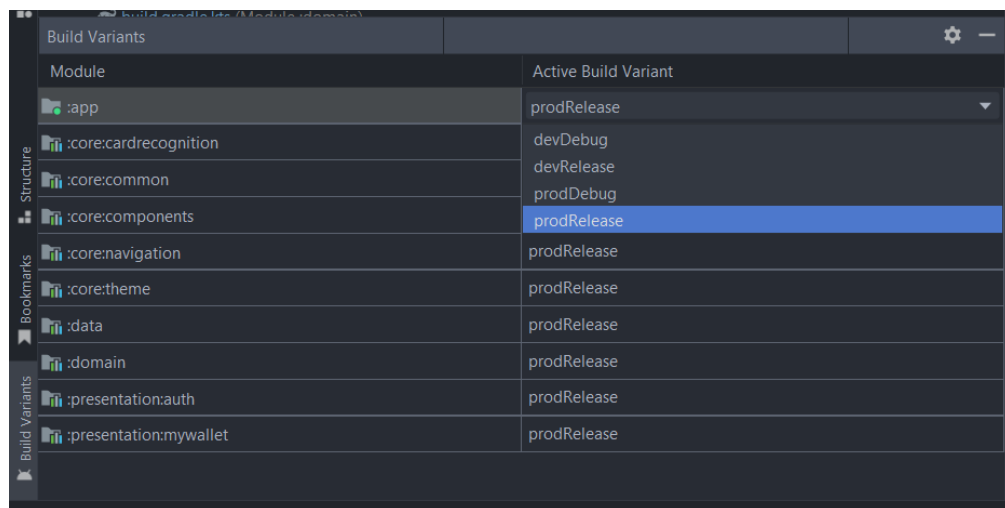


Рисунок 4.1 – Зміна типу збірки на «випуск»

Після того як проєкт синхронізується, треба зібрати проєкт і останнім кроком буде збірка APK (рис. 4.2).

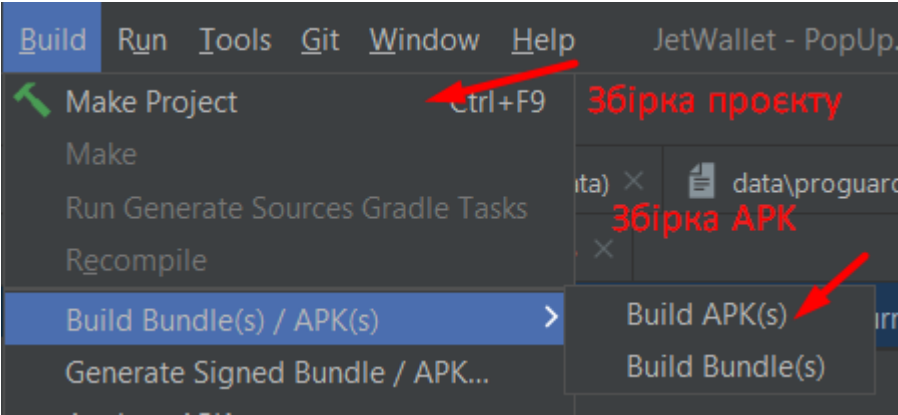


Рисунок 4.2 – Збирання проєкту і APK

APK з’явиться у директорії \JetWallet\app\build\outputs\apk\prod\release (рис. 4.3).

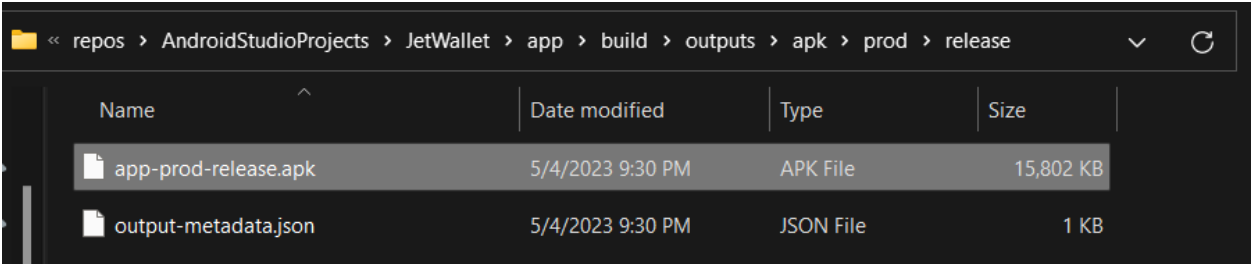


Рисунок 4.3 – Директорія де з’явиться APK

4.1.1 Порівняння захищеного та не захищеного APK

За допомогою ADB та функції у Android Studio «Device File Explorer» можна легко подивитися файли додатка (див. табл. 4.1).

Таблиця 4.1 – Порівняння видимості внутрішніх даних

isDebuggable = true	isDebuggable = false
com.zhirokejwallet cache code_cache databases com.google.android.data com.google.android.data wallet_database wallet_database-shm wallet_database-wal files wallet_db_password.bin shared_prefs com.google.mlkit.interna secured_user_prefs.xml drwxrwx--x drwxrws--x drwxrws--x drwxrwx--x -rw-rw---- -rw----- -rw----- -rw----- drwxrwx--x -rw----- drwxrwx--x -rw-rw---- -rw-rw---- 2023-04-17 13:08 2023-03-29 14:11 2023-04-26 22:00 2023-04-30 15:15 2023-04-30 16:44 2023-04-30 16:44 2023-05-03 23:40 2023-04-29 19:22 2023-03-29 14:11 2023-03-29 14:11 2023-04-30 16:43 2023-04-30 16:43 2023-04-29 16:40 4 KB 4 KB 4 KB 60 KB 24.5 KB 4 KB 32 KB 181.1 KB 4 KB 66 B 4 KB 154 B 1.2 KB	com.zhirokejwallet run-as: package not debuggable drwxrwx--x 2023-04-17 13:08 4 KB

Як можна побачити, після того, як було вимкнено режим налагодження, Android система забороняє перегляд файлів додатка.

Порівняння APK з R8 та без (див. табл. 4.2 – 4.3). Декомпіляція коду була зроблена за допомогою сайту [javadecompilers](#) [15].

Таблиця 4.2 – Порівняння розміру APK з R8 та без

З R8	Без R8
Type: APK File Size: 15.4 MB	Type: APK File Size: 24.9 MB

Таблиця 4.3 – Порівняння коду APK з R8 та без

З R8	C0003d.java <pre>/* renamed from: a.d */ public final class C0003d extends C1771k implements C1544l<C1149s0, C1145r0> { /* renamed from: o */ public final /* synthetic */ C0000a<Object> f4o; /* renamed from: p */ public final /* synthetic */ C0236e f5p; /* renamed from: q */ public final /* synthetic */ String f6q; /* renamed from: r */ public final /* synthetic */ C1015a<Object, Object> f7r; /* renamed from: s */ public final /* synthetic */ C1176x2<C1544l<Object, C5915k>> f8s; /* JADX INFO: super call moved to the top of the method (can break code semantics) public C0003d(C0000a aVar, C0236e eVar, String str, C1015a aVar2, C1103j1 j1Var) { super(1); this.f4o = aVar; this.f5p = eVar; this.f6q = str; this.f7r = aVar2; this.f8s = j1Var; } }</pre>

Кінець таблиці 4.3

Без R8	<pre> compiled from: DataModule.kt */ lic final class DataModuleKt { public static final Module dataModule() { return ModuleDSLKt.module\$default(false, DataModuleKt\$dataModule\$1.INSTANCE, 1, (Object) null); } /* access modifiers changed from: private */ public static final WalletDatabase provideWalletRoomDatabase(Context context, PasswordProvider passwordProvider, byte[] password = passwordProvider.getPassword(fileName: "wallet_db_password.bin"); RoomDatabase.Builder \$this\$provideWalletRoomDatabase_u24lambda_u240 = Room.databaseBuilder(context, WalletRoomDatabase.class, "wallet_db", \$this\$provideWalletRoomDatabase_u24lambda_u240.openHelperFactory(new SupportFactory(password))); WalletDatabase build = \$this\$provideWalletRoomDatabase_u24lambda_u240.build(); Intrinsics.checkNotNullExpressionValue(build, expression: "databaseBuilder(context,...d))\n build()\n return build; } /* access modifiers changed from: private */ public static final BankCardDao provideBankCardDao(WalletDatabase walletDatabase) { return walletDatabase.getBankCardDao(); } /* access modifiers changed from: private */ public static final SharedPreferences provideEncryptedSharedPreferences(Context context) { MasterKey masterKey = new MasterKey.Builder(context).setKeyScheme(MasterKey.KeyScheme.AES256_GCM).build(); Intrinsics.checkNotNullExpressionValue(masterKey, expression: "Builder(context)\n ...256_GCM)\n SharedPreferences create = EncryptedSharedPreferences.create(context, fileName: "secured_user_prefs", masterKey, Context.MODE_PRIVATE); Intrinsics.checkNotNullExpressionValue(create, expression: "create(\n context,...onScheme.AES256_GCM\n return create; </pre>
--------	--

Можна зробити висновки, що після підключення R8, розмір APK було зменшено більш ніж у півтора раза і код став важко читабельним завдяки обфускації.

4.2 Експлуатація додатка

Для початку роботи з додатком, користувачеві треба відкрити додаток, для цього треба натиснути на ярлик JetWallet.

Після старту додатка, користувач може має такі дії:

- розблокування за допомогою вводу PIN коду;
- сканування відбитку пальцю;
- створення PIN коду, якщо його нема;
- скинути налаштування.

Після розблокування додатка, користувача буде перекинуто на головний екран. На екрані «Каруселька карток» він може:

- гортати картки – зробити горизонтальний свайп у карусельці карток;

- перегортати картку – двійне, швидке, послідовне натискання;
- копіювати номер картки – затиснути фронтальну сторону картки;
- видалити картку – натиснути на іконку «відро для сміття», яка розташована на фронтальній стороні картки;
- редагувати картку – натиснути на іконку «олівець», яка розташована на фронтальній стороні картки;
- додати картку – натиснути на кнопку з іконкою «+», яка розташована у нижньому правому куту екрана.

Після того, як користувач відкрив спливаюче вікно для додавання карток, він має два способу додавання карток:

- ручним способом – заповнити поля картки за допомогою клавіатури;
- скануванням – заповнити номер картки і термін придатності за допомогою сканера карток. Треба свайпнути вправо та дозволити доступ системі до камери. Після цього треба навестися камерою на фронтальний бік картки, та держати картку у рамці декілька секунд.

Щоб додати малюнок для картки, треба натиснути на іконку «картинка з +», ця іконка знаходиться на картці, у правому верхньому куті. Після треба обрати потрібну картинку у вікні, що відкрилося.

Щоб видалити малюнок, треба натиснути на іконку «х», ця іконка знаходиться на картці, у правому верхньому куті.

Після того як відкрилося спливаюче вікно редагування, користувач має такі ж самі дії що і у вікні додавання, крім можливості сканування картки.

4.3 Тестування додатка

4.3.1 Unit тести додатка

Для перевірки шифрування даних, було написано декілька unit тестів. Перші два тести перевіряють можливість безпечного зберігання PIN коду додатка у SharedPreferences. Код Unit тестів можна подивитися у додатку А під файлом SecuredPrefsTest.kt. Тести:

– тест `SecuredPrefsTest.reading_of_encrypted_data_failed()` – перевіряє, чи було проведене шифрування при додаванні PIN коду, якщо результат зчитування PIN коду не сходиться з оригінальними даними, це вважається, що тест було пройдено успішно;

– тест `SecuredPrefsTest.reading_of_decrypted_data_succeeded()` – перевіряє, чи може система зчитувати зашифровані PIN коду. Якщо результат зчитування PIN коду сходиться з оригіналом даними, це вважається, що тест було пройдено успішно.

По результатам тестування, можна побачити, що зберігання PIN коду працює і є безпечним (рис. 4.4).

Tests	Duration	Pixel_3a_API_29
✓ Test Results	807 ms	2/2
✓ SecuredPrefsTest	807 ms	2/2
✓ reading_of_decrypted_data_succeeded	669 ms	✓
✓ reading_of_encrypted_data_failed	138 ms	✓

Рисунок 4.4 – Результати SecuredPrefsTest

Третій і четвертий тест перевіряє можливість безпечного зберігання банківської картки в базі даних. Код Unit тестів можна подивитися у додатку А під файлом `WalletDatabaseTest.kt`. Тести:

– тест `WalletDatabaseTest.reading_of_encrypted_data_failed()` – перевіряє, чи було проведене шифрування при додаванні банківської картки, якщо результат зчитування банківської картки не сходиться з оригінальними даними, це вважається, що тест було пройдено успішно;

– тест `WalletDatabaseTest.reading_of_decrypted_data_succeeded()` – перевіряє, чи може система зчитувати зашифровані банківської картки. Якщо результат зчитування банківської картки сходиться з оригіналом даними, це вважається, що тест було пройдено успішно.

По результатам тестування, можна побачити, що зберігання банківських карток працює і є безпечним (рис. 4.5).

Tests	Duration	Pixel_3a_API_29
✓ Test Results	1 s	2/2
✓ WalletDatabaseTest	1 s	2/2
✓ reading_of_decrypted_data_succeeded	1 s	✓
✓ reading_of_encrypted_data_failed	618 ms	✓

Рисунок 4.5 – Результати WalletDatabaseTest

4.3.2 Тестування на реальних пристроях

Тестування проводилось на різних девайсах та з різними версіями андроїда (див. табл. 4.4).

Таблиця 4.4 – Результати тестування додатка

Критерії тестування	Пристрій та андроїд версія		
	Xiaomi Redmi 5+ (Android 8.1)	Xiaomi Redmi 9 (Android 12)	Samsung A32 (Android 13)
Адаптивність інтерфейсу	Коректно	Коректно	Коректно
Захист від шпигунського програмного забезпечення	Коректно	Коректно	Коректно
Шифрування даних	Коректно	Коректно	Коректно
Захист внутрішньої пам'яті	Коректно	Коректно	Коректно
Плавність роботи	Відмінно	Відмінно	Відмінно
Швидкість відкриття додатка	1.5 сек	1 сек	1 сек
Швидкість завантаження даних	Прийнятна	Швидко	Швидко

На всіх девайсах захист даних працює коректно. З точки зору швидкості UI, все також працює швидко, навіть на трохи застарілому девайсі як Xiaomi Redmi 5+, додаток працює швидко.

ВИСНОВКИ

Разом із бакалаврською дипломною роботою, було розроблено нативний Android додаток JetWallet для безпечного зберігання карток.

Перед початком розробки, було досліджено вимоги до застосування і проведено аналіз існуючих аналогів. Також було порівнянні існуючі бібліотеки, архітектурні патерни, середовища розробки та мови програмування.

Розробка велась в IDE Android Studio. При виконанні розробки, були дотримані принципи Чистої Архітектури та було реалізовано патерн MVI. Завдяки цим рішенням, було досягнуто стабільної, з меншою кількістю багів і більш підтримуваної системи.

Стосовно безпеки даних, були використані такі методи: шифрування даних, обфускація APK, обмеження доступу до внутрішніх файлів додатка, екран блокування і блокування запису екрана.

Розроблена система відповідає всім зазначеним у темі вимогам. Знайти репозиторій проєкта можна на GitHub [16].

Як можна покращати додаток:

- додати діалогове вікно під час видалення картки;
- переписати логіку валідації, під час додавання/редагування картки;
- покращати розпізнавання карток;
- додати можливість додавання карток за допомогою NFC.

ПЕРЕЛІК ДЖЕРЕЛ І ПОСИЛАНЬ

1. SecureCard [Electronic Resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.texode.securecard>
2. Google Wallet [Electronic Resource]. – Access mode: <https://play.google.com/store/apps/details?id=com.google.android.apps.walletnfcrel>
3. Native and cross-platform app development [Electronic Resource]. – Access mode: <https://kotlinlang.org/docs/native-and-cross-platform.html#what-is-native-mobile-app-development>
4. Ghosh A. Comparison of Encryption Algorithms [Text] / A. Ghosh // International Research Journal of Engineering and Technology (IRJET). – 2020. Vol. 7. No. 6. p-ISSN: 2395-0072.
5. Kotlin vs. Java for Android development August 17, 2022 Levi Vaguez [Electronic Resource]. – Access mode: <https://blog.logrocket.com/kotlin-vs-java-android-development/>
6. Kotlin vs Java performance [Electronic Resource]. – Access mode:
7. https://www.reddit.com/r/Kotlin/comments/ubseb8/kotlin_vs_java_performance/
8. Hilt is stable! Easier dependency injection on Android [Electronic Resource]. – Access mode: <https://medium.com/androiddevelopers/hilt-is-stable-easier-dependency-injection-on-android-53aca3f38b9c>
9. Why koin? [Electronic Resource]. – Access mode: <https://insert-koin.io/docs/setup/why/>
10. MVI Architecture with Android [Electronic Resource]. – Access mode: <https://medium.com/swlh/mvi-architecture-with-android-fcde123e3c4a>
11. The Clean Architecture [Electronic Resource]. – Access mode: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
12. Material Design 3 for Compose is now stable [Electronic Resource]. – Access mode: <https://material.io/blog/material-3-compose-stable>

13. Photo picker [Electronic Resource]. – Access mode:
<https://developer.android.com/training/data-storage/shared/photopicker>

14. Request runtime permissions [Electronic Resource]. – Access mode:
<https://developer.android.com/training/permissions/requesting>

15. EncryptedSharedPreferences [Electronic Resource]. – Access mode:
<https://developer.android.com/reference/androidx/security/crypto/EncryptedSharedPreferences>

16. Android Apk decompiler [Electronic Resource]. – Access mode:
<http://www.javadecompilers.com/apk>

17. JetWallet [Electronic Resource]. – Access mode:
<https://github.com/ZhEgor/JetWallet>

ДОДАТОК А**Код програми**

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:name=".App"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.JetWallet"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:configChanges="orientation"
            android:screenOrientation="portrait"
            android:theme="@style/Theme.JetWallet">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <meta-data
            android:name="com.google.mlkit.vision.DEPENDENCIES"
            android:value="ocr" />
    </application>

</manifest>

```

```
package com.zhiroke.jetwallet.navigation.subgraphs
```

```

import androidx.navigation.NavGraphBuilder
import androidx.navigation.compose.composable
import com.zhiroke.core.navigation.directions.AuthNavDirections

```

```
import com.zhiroke.presentation.auth.presentation.auth.AuthRoute
```

```
fun NavGraphBuilder.authGraph() {
    composable(route = AuthNavDirections.auth.destination) {
        AuthRoute()
    }
}
```

```
package com.zhiroke.jetwallet.navigation.subgraphs
```

```
import androidx.navigation.NavGraphBuilder
import androidx.navigation.compose.composable
import com.zhiroke.core.navigation.directions.MyWalletNavDirections
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.CardCarouselRoute
```

```
fun NavGraphBuilder.myWalletGraph() {
    composable(route = MyWalletNavDirections.cardCarousel.destination) {
        CardCarouselRoute()
    }
}
```

```
package com.zhiroke.jetwallet.navigation
```

```
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.navigation.NavHostController
import androidx.navigation.compose.NavHost
import com.zhiroke.core.navigation.directions.AuthNavDirections
import com.zhiroke.jetwallet.navigation.subgraphs.authGraph
import com.zhiroke.jetwallet.navigation.subgraphs.myWalletGraph
```

```
@Composable
```

```
fun NavigationGraph(
    modifier: Modifier = Modifier,
    navController: NavHostController,
    startDestination: String = AuthNavDirections.auth.destination
) {
    NavHost(modifier = modifier, navController = navController, startDestination
= startDestination) {
```

```

        authGraph()
        myWalletGraph()
    }
}

package com.zhiroke.jetwallet.ui.theme

import androidx.compose.ui.graphics.Color

/** Material 3 light colors palette */
val LightPrimary = Color(0xFF714D9F)
val White = Color(0xFFFFFFFF)
val LightPrimaryContainer = Color(0xFFEEDCFF)
val LightOnPrimaryContainer = Color(0xFF2A0054)
val LightSecondary = Color(0xFF00658F)
val LightSecondaryContainer = Color(0xFFC7E7FF)
val LightOnSecondaryContainer = Color(0xFF001E2E)
val LightTertiary = Color(0xFF006A64)
val LightTertiaryContainer = Color(0xFFFFDAD6)
val LightOnTertiaryContainer = Color(0xFF00201D)
val LightError = Color(0xFFBA1A1A)
val LightErrorContainer = Color(0xFFFFDAD6)
val LightOnErrorContainer = Color(0xFF410002)
val LightBackground = Color(0xFFFFF7EDFF)
val LightOnBackground = Color(0xFF24005A)
val LightSurface = Color(0xFFFFF7EDFF)
val LightOnSurface = Color(0xFF24005A)
val LightOutline = Color(0xFF7B757F)
val LightSurfaceVariant = Color(0xFFE8E0EB)
val LightOnSurfaceVariant = Color(0xFF4A454E)

/** Material 3 dark colors palette */
val DarkPrimary = Color(0xFFD9B9FF)
val DarkOnPrimary = Color(0xFF401B6D)
val DarkPrimaryContainer = Color(0xFF583485)
val DarkOnPrimaryContainer = Color(0xFFEEDCFF)
val DarkSecondary = Color(0xFF86CFFF)
val DarkOnSecondary = Color(0xFF00344C)
val DarkSecondaryContainer = Color(0xFF004C6D)
val DarkOnSecondaryContainer = Color(0xFFC7E7FF)
val DarkTertiary = Color(0xFF50DBC9)

```

```

val DarkOnTertiary = Color(0xFF003733)
val DarkTertiaryContainer = Color(0xFF00504B)
val DarkOnTertiaryContainer = Color(0xFF71F7EB)
val DarkError = Color(0xFFFFFB4AB)
val DarkOnError = Color(0xFF690005)
val DarkErrorContainer = Color(0xFF93000A)
val DarkOnErrorContainer = Color(0xFFFFDAD6)
val DarkBackground = Color(0xFF1D1B1E)
val DarkOnBackground = Color(0xFFE7E1E5)
val DarkSurface = Color(0xFF1D1B1E)
val DarkOnSurface = Color(0xFFEADDFF)
val DarkOutline = Color(0xFF958E99)
val DarkSurfaceVariant = Color(0xFF4A454E)
val DarkOnSurfaceVariant = Color(0xFFCCC4CF)

```

```
package com.zhiroke.jetwallet.ui.theme
```

```

import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Shapes
import com.zhiroke.core.theme.dimensions.dp_12
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_24
import com.zhiroke.core.theme.dimensions.dp_4
import com.zhiroke.core.theme.dimensions.dp_8

```

```

val replyShapes = Shapes(
    extraSmall = RoundedCornerShape(dp_4),
    small = RoundedCornerShape(dp_8),
    medium = RoundedCornerShape(dp_12),
    large = RoundedCornerShape(dp_16),
    extraLarge = RoundedCornerShape(dp_24)
)

```

```
package com.zhiroke.jetwallet.ui.theme
```

```

import android.app.Activity
import android.os.Build
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.*
import androidx.compose.runtime.Composable

```

```

import androidx.compose.runtime.SideEffect
import androidx.compose.ui.graphics.toArgb
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalView
import androidx.core.view.ViewCompat

private val DarkColorScheme = darkColorScheme(
    primary = DarkPrimary,
    onPrimary = DarkOnPrimary,
    primaryContainer = DarkPrimaryContainer,
    onPrimaryContainer = DarkOnPrimaryContainer,
    secondary = DarkSecondary,
    onSecondary = DarkOnSecondary,
    secondaryContainer = DarkSecondaryContainer,
    onSecondaryContainer = DarkOnSecondaryContainer,
    tertiary = DarkTertiary,
    onTertiary = DarkOnTertiary,
    tertiaryContainer = DarkTertiaryContainer,
    onTertiaryContainer = DarkOnTertiaryContainer,
    background = DarkBackground,
    onBackground = DarkOnBackground,
    surface = DarkSurface,
    onSurface = DarkOnSurface,
    surfaceVariant = DarkSurfaceVariant,
    onSurfaceVariant = DarkOnSurfaceVariant,
    error = DarkError,
    onError = DarkOnError,
    errorContainer = DarkErrorContainer,
    onErrorContainer = DarkOnErrorContainer,
    outline = DarkOutline,
    outlineVariant = DarkSurfaceVariant,
)

private val LightColorScheme = lightColorScheme(
    primary = LightPrimary,
    onPrimary = White,
    primaryContainer = LightPrimaryContainer,
    onPrimaryContainer = LightOnPrimaryContainer,
    secondary = LightSecondary,
    onSecondary = White,
    secondaryContainer = LightSecondaryContainer,
    onSecondaryContainer = LightOnSecondaryContainer,

```

```

    tertiary = LightTertiary,
    onTertiary = White,
    tertiaryContainer = LightTertiaryContainer,
    onTertiaryContainer = LightOnTertiaryContainer,
    background = LightBackground,
    onBackground = LightOnBackground,
    surface = LightSurface,
    onSurface = LightOnSurface,
    surfaceVariant = LightSurfaceVariant,
    onSurfaceVariant = LightOnSurfaceVariant,
    error = LightError,
    onError = White,
    errorContainer = LightErrorContainer,
    onErrorContainer = LightOnErrorContainer,
    outline = LightOutline,
    outlineVariant = LightSurfaceVariant,
)

@Composable
fun JetWalletTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colorScheme = when {
        dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
            val context = LocalContext.current
            if (darkTheme) dynamicDarkColorScheme(context) else
dynamicLightColorScheme(context)
        }

        darkTheme -> DarkColorScheme
        else -> LightColorScheme
    }

    val view = LocalView.current
    if (!view.isInEditMode) {
        SideEffect {
            (view.context as Activity).window.statusBarColor =
colorScheme.primary.toArgb()

ViewCompat.getWindowInsetsController(view)?.isAppearanceLightStatusBars =

```

```

darkTheme
    }
}

MaterialTheme(
    colorScheme = colorScheme,
    typography = Typography,
    shapes = replyShapes,
    content = content
)
}

package com.zhiroke.jetwallet.ui.theme

import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp

// Set of Material typography styles to start with
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    )
    /* Other default text styles to override
    titleLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 22.sp,
        lineHeight = 28.sp,
        letterSpacing = 0.sp
    ),
    labelSmall = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,
        fontSize = 11.sp,

```

```

        lineHeight = 16.sp,
        letterSpacing = 0.5.sp
    )
    */
)

package com.zhiroke.jetwallet.ui

import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.navigation.compose.rememberNavController
import com.zhiroke.core.navigation.utils.attachNavController
import com.zhiroke.jetwallet.navigation.NavigationGraph
import com.zhiroke.jetwallet.ui.theme.JetWalletTheme

@Composable
internal fun JetWallet() {
    val navController = rememberNavController()
    attachNavController(navController = navController)

    JetWalletTheme {
        // A surface container using the 'background' color from the theme
        Surface(
            modifier = Modifier.fillMaxSize(),
            color = MaterialTheme.colorScheme.background
        ) {
            NavigationGraph(navController = navController)
        }
    }
}

package com.zhiroke.jetwallet.utils.initmanager

interface InitManager {
    fun init()
}

```

```

package com.zhiroke.jetwallet.utils.initmanager

import android.app.Application
import com.zhiroke.core.common.di.modules.commonModule
import com.zhiroke.core.navigation.di.modules.navigationModule
import com.zhiroke.data.di.modules.dataModule
import com.zhiroke.domain.di.modules.repositoriesModule
import com.zhiroke.presentation.auth.di.modules.authModules
import com.zhiroke.presentation.mywallet.di.modules.myWalletModules
import org.koin.android.ext.koin.androidContext
import org.koin.core.context.startKoin

class KoinInitManager(private val application: Application) : InitManager {

    override fun init() {
        startKoin {
            androidContext(application)
            modules(
                dataModule(),
                repositoriesModule(),
                navigationModule(),
                commonModule(),
                authModules(),
                myWalletModules()
            )
        }
    }
}

package com.zhiroke.jetwallet

import android.app.Application
import com.zhiroke.jetwallet.utils.initmanager.InitManager
import com.zhiroke.jetwallet.utils.initmanager.KoinInitManager

class App : Application() {

    override fun onCreate() {
        super.onCreate()
        initApp()
    }
}

```

```

    }

    private fun initApp() {
        listOf<InitManager>(KoinInitManager(application =
this@App)).forEach(InitManager::init)
    }
}

package com.zhiroke.jetwallet

import android.os.Bundle
import android.view.WindowManager
import androidx.activity.compose.setContent
import androidx.fragment.app.FragmentActivity
import com.zhiroke.core.navigation.utils.detachNavController
import com.zhiroke.jetwallet.ui.JetWallet
import com.zhiroke.presentation.auth.utils.managers.attachBiometricAuthManager
import com.zhiroke.presentation.auth.utils.managers.detachBiometricAuthManager

class MainActivity : FragmentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        window.setFlags(WindowManager.LayoutParams.FLAG_SECURE,
WindowManager.LayoutParams.FLAG_SECURE)
        setContent(content = { JetWallet() })
        attachBiometricAuthManager(activity = this)
    }

    override fun onDestroy() {
        detachNavController()
        detachBiometricAuthManager()
        super.onDestroy()
    }
}

import java.io.FileInputStream
import java.util.Properties

# Add project specific ProGuard rules here.
# You can control the set of applied configuration files using the

```

```

# proguardFiles setting in build.gradle.kts.
#
# For more details, see
#   http://developer.android.com/guide/developing/tools/proguard.html

# If your project uses WebView with JS, uncomment the following
# and specify the fully qualified class name to the JavaScript interface
# class:
-keepclassmembers class fqcn.of.javascript.interface.for.webview {
#   public *;
#}

# Uncomment this to preserve the line number information for
# debugging stack traces.
-keepattributes SourceFile,LineNumberTable

# If you keep the line number information, uncomment this to
# hide the original source file name.
-renamesourcefileattribute SourceFile
-keep class net.sqlcipher.** {
    *;
}

object Configs {
    const val compileSdk = 33
    const val minSdk = 26
    const val targetSdk = 33
    const val versionCode = 2
    const val versionName = "1.0.1"
    const val packageName = "com.zhiroke.jetwallet"
}

object Libs {

    object AndroidX {
        const val coreKtx = "androidx.core:core-ktx:1.9.0"
        const val appCompat = "androidx.appcompat:appcompat:1.5.1"
        const val activityCompose = "androidx.activity:activity-compose:1.6.1"
        const val biometric = "androidx.biometric:biometric-
ktx:${Versions.biometric}"
    }
}

```

```

    }

    object Compose {
        const val ui = "androidx.compose.ui:ui:${Versions.compose}"
        const val runtime =
"androidx.compose.runtime:runtime:${Versions.compose}"
        const val uiUtils = "androidx.compose.ui:ui-util:${Versions.compose}"
        const val materialIconsExtended = "androidx.compose.material:material-
icons-extended:${Versions.compose}"
        const val material3 =
"androidx.compose.material3:material3:${Versions.material3}"
        const val toolingPreview = "androidx.compose.ui:ui-tooling-
preview:${Versions.compose}"
        const val tooling = "androidx.compose.ui:ui-tooling:${Versions.compose}"
        const val accompanistPager = "com.google.accompanist:accompanist-
pager:${Versions.accompanist}"
        const val accompanistPagerIndicator =
"com.google.accompanist:accompanist-pager-indicators:${Versions.accompanist}"
        const val testJunit4 = "androidx.compose.ui:ui-test-
junit4:${Versions.compose}"
        const val testManifest = "androidx.compose.ui:ui-test-
manifest:${Versions.compose}"
    }

    object Permissions {
        const val accompanist = "com.google.accompanist:accompanist-
permissions:${Versions.accompanist}"
    }

    object Coroutines {
        const val core = "org.jetbrains.kotlinx:kotlinx-coroutines-
core:${Versions.coroutines}"
        const val test = "org.jetbrains.kotlinx:kotlinx-coroutines-
test:${Versions.coroutines}"
    }

    object Gms {
        const val textRecognizerMlKit = "com.google.android.gms:play-services-
mlkit-text-recognition:${Versions.textRecognizerMlKit}"
    }

    object CameraX {

```

```

        const val mlKit = "androidx.camera:camera-mlkit-
vision:${Versions.cameraXmlKit}"
        const val camera2 = "androidx.camera:camera-camera2:${Versions.cameraX}"
        const val lifecycle = "androidx.camera:camera-
lifecycle:${Versions.cameraX}"
        const val view = "androidx.camera:camera-view:${Versions.cameraX}"
    }

    object Koin {
        const val core = "io.insert-koin:koin-core:${Versions.koin}"
        const val android = "io.insert-koin:koin-android:${Versions.koin}"
        const val compose = "io.insert-koin:koin-androidx-
compose:${Versions.koin}"
    }

    object Coil {
        const val compose = "io.coil-kt:coil-compose:${Versions.coil}"
    }

    object Room {
        const val runtime = "androidx.room:room-runtime:${Versions.room}"
        const val ktx = "androidx.room:room-ktx:${Versions.room}"
        const val compiler = "androidx.room:room-compiler:${Versions.room}"
    }

    object SqlCipher {
        const val cipher = "net.zetetic:android-database-
sqlcipher:${Versions.sqlcipher}"
    }

    object Security {
        const val crypto = "androidx.security:security-
crypto:${Versions.securityCrypto}"
    }

    object Navigation {
        const val navigationCompose = "androidx.navigation:navigation-
compose:${Versions.navigation}"
    }

    object Lifecycle {
        const val runtimeKtx = "androidx.lifecycle:lifecycle-runtime-

```

```

ktx:${Versions.lifecycle}"

    const val runtimeCompose = "androidx.lifecycle:lifecycle-runtime-
compose:${Versions.lifecycle}"

    const val viewModelCompose = "androidx.lifecycle:lifecycle-viewmodel-
compose:${Versions.lifecycle}"

    }

    object Test {

        const val junit = "junit:junit:4.13.2"
        const val androidJUnit = "androidx.test.ext:junit:1.1.3"
        const val espresso = "androidx.test.espresso:espresso-core:3.4.0"
        const val googleTruth = "com.google.truth:truth:1.1.3"

    }

}

```

```

object Versions {

    const val kotlin = "1.6.21"
    const val koin = "3.2.2"
    const val room = "2.4.3"
    const val sqlcipher = "4.5.0@aar"
    const val navigation = "2.5.0"
    const val coroutines = "1.5.2"
    const val cameraX = "1.2.0"
    const val cameraXMLKit = "1.2.0-beta02"
    const val textRecognizerMlKit = "18.0.2"
    const val lifecycle = "2.6.0-alpha01"
    const val compose = "1.2.1"
    const val coil = "2.1.0"
    const val composeCompiler = "1.3.2"
    const val accompanist = "0.28.0"
    const val material3 = "1.1.0-alpha02"
    const val biometric = "1.2.0-alpha04"
    const val securityCrypto = "1.1.0-alpha04"

}

```

```

package com.zhiroke.core.cardrecognition.composable

```

```

import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.zhiroke.core.common.permissions.RequestPermissionHelper

```

```

@Composable
fun CardRecognizer(
    modifier: Modifier = Modifier,
    onDismissRequest: () -> Unit,
    onFrontSideCardRecognized: (cardNumber: String, expirationDate: String) ->
Unit,
    onBackSideCardRecognized: (verificationNumber: String) -> Unit
) {

    RequestPermissionHelper(
        permission = android.Manifest.permission.CAMERA,
        permissionNotGrantedContent = { onRequestPermission ->
            Rationale(onDismissRequest = onDismissRequest, onRequestPermission =
onRequestPermission)
        },
        permissionGrantedContent = {
            UnsafeCardRecognizer(
                modifier = modifier,
                onFrontSideCardRecognized = onFrontSideCardRecognized,
                onBackSideCardRecognized = onBackSideCardRecognized
            )
        }
    )
}

```

```

package com.zhiroke.core.cardrecognition.composable

```

```

import androidx.compose.material3.AlertDialog
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.res.stringResource
import com.zhiroke.core.cardrecognition.R

```

```

@Composable
internal fun Rationale(
    onDismissRequest: () -> Unit,
    onRequestPermission: () -> Unit,
) {

```

```

AlertDialog(
    title = {
        Text(text =
stringResource(R.string.rational_permission_request_title))
    },
    text = {
        Text(text =
stringResource(R.string.rational_permission_request_rational_text))
    },
    confirmButton = {
        Button(onClick = onRequestPermission) {

Text(stringResource(R.string.rational_permission_request_confirm_button))

        }
    },
    onDismissRequest = onDismissRequest
)
}

```

```
package com.zhiroke.core.cardrecognition.composable
```

```

import android.util.Log
import androidx.camera.mlkit.vision.MlKitAnalyzer
import androidx.camera.view.CameraController
import androidx.camera.view.LifecycleCameraController
import androidx.camera.view.PreviewView
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.runtime.Composable
import androidx.compose.runtime.DisposableEffect
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalLifecycleOwner
import androidx.compose.ui.viewinterop.AndroidView
import com.google.mlkit.vision.text.TextRecognition

```

```

import com.google.mlkit.vision.text.latin.TextRecognizerOptions
import com.zhiroke.core.cardrecognition.recognizer.DigitRecognizer
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.asExecutor

@OptIn(ExperimentalCoroutinesApi::class)
@Composable
internal fun UnsafeCardRecognizer(
    modifier: Modifier = Modifier,
    onFrontSideCardRecognized: (cardNumber: String, expirationDate: String) ->
Unit,
    onBackSideCardRecognized: (verificationNumber: String) -> Unit
) {
    val context = LocalContext.current
    val lifecycleOwner = LocalLifecycleOwner.current

    val cameraController = remember { LifecycleCameraController(context) }
    val cameraView = remember { PreviewView(context) }
    val cameraExecutor = remember {
Dispatchers.Default.limitedParallelism(2).asExecutor() }
    val textScanner = remember {
TextRecognition.getClient(TextRecognizerOptions.DEFAULT_OPTIONS) }

    LaunchedEffect(Unit) {
        var recognizedCardNumber: String? = null
        var recognizedExpirationDate: String? = null

        val cardNumberRecognizer = DigitRecognizer(pattern = "xxxx xxxx xxxx
xxxx") { cardNumber ->
            recognizedCardNumber = cardNumber
            recognizedExpirationDate?.let { expirationDate ->
                onFrontSideCardRecognized.invoke(cardNumber, expirationDate)
            }
            Log.d("JET_TAG", "\n\n\n\n\nbest match: $cardNumber")
        }

        val expirationDateRecognizer = DigitRecognizer(pattern = "xx/xx") {
expirationDate ->
            recognizedExpirationDate = expirationDate
            recognizedCardNumber?.let { cardNumber ->
                onFrontSideCardRecognized.invoke(cardNumber, expirationDate)

```

```

    }
    Log.d("JET_TAG", "\n\n\n\n\nbest match: $recognizedExpirationDate")
  }
  val verificationNumberRecognizer = DigitRecognizer(pattern = "xxx") {
verificationNumber ->
    if (recognizedCardNumber == null && recognizedExpirationDate ==
null) {
        onBackSideCardRecognized.invoke(verificationNumber)
    }
    Log.d("JET_TAG", "\n\n\n\n\nbest match: $recognizedExpirationDate")
  }

cameraController.setEnabledUseCases(CameraController.IMAGE_ANALYSIS)
cameraController.bindToLifecycle(lifecycleOwner)
cameraController.setImageAnalysisAnalyzer(
    cameraExecutor,
    MlKitAnalyzer(
        listOf(textScanner),
        CameraController.COORDINATE_SYSTEM_VIEW_REFERENCED,
        cameraExecutor
    ) { result ->
        val text = result.getValue(textScanner)
        text?.textBlocks?.forEach { textBlock ->
            cardNumberRecognizer.feed(textBlock.text)
            expirationDateRecognizer.feed(textBlock.text.filterNot {
it.isLetter() })
            verificationNumberRecognizer.feed(textBlock.text)
        }
    }
)

cameraView.controller = cameraController
}

DisposableEffect(Unit) {
    onDispose {
        cameraView.controller = null
        cameraController.unbind()
    }
}

Box(modifier = modifier.background(Color.Black)) {

```

```

        AndroidView(factory = { cameraView }, modifier = Modifier.fillMaxSize())
    }
}

package com.zhiroke.core.cardrecognition.recognizer

import android.util.Log

internal class DigitRecognizer(
    pattern: String,
    iterationLimitMin: Int = 5,
    iterationLimitMax: Int = 10,
    onSuccess: (String) -> Unit
) : Recognizer(
    pattern = pattern,
    iterationLimitMin = iterationLimitMin,
    iterationLimitMax = iterationLimitMax,
    onSuccess = onSuccess
) {

    override fun findBestMatch(selection: Selection): String {
        val unrecognizedSymbol = DIGIT_SYMBOL
        val bestMatch = CharArray(pattern.length) { unrecognizedSymbol }

        for (i in pattern.indices) {
            if (pattern[i] == DIGIT_SYMBOL) {
                val bestMatchesForSymbol: HashMap<Char, Int> = hashMapOf()
                for (j in selection.indices) {
                    val matchingSymbol = selection[j][i]
                    if (matchingSymbol.isDigit()) {
                        bestMatchesForSymbol[matchingSymbol] =
bestMatchesForSymbol[matchingSymbol]?.let { it + 1 } ?: 1 // creating selection
of repeating ones
                    }
                }
                bestMatchesForSymbol.maxByOrNull { it.value }?.key?.let { symbol
->
                    bestMatch[i] = symbol
                }
            }
        }
    }
}

```

```

        } else {
            bestMatch[i] = pattern[i]
        }
    }

    return bestMatch.filterNot { symbol -> symbol == unrecognizedSymbol
}.joinToString("").also {
    Log.d("JET_TAG", "$currentIteration. potential best match: $it")
}
}
}

package com.zhiroke.core.cardrecognition.recognizer

import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.SupervisorJob
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.flow.update
import kotlinx.coroutines.launch

internal typealias Selection = List<List<Char>>

/**
 * @param pattern example: "xxxx xxxx xxxx xxxx".
 * @param iterationLimitMin with this amount of iteration we guarantee not to
throw a best match
 * @param iterationLimitMax with this amount of iteration we guarantee to throw
a best match
 */
internal abstract class Recognizer(
    protected val pattern: String,
    protected val iterationLimitMin: Int,
    protected val iterationLimitMax: Int,
    protected val onSuccess: (String) -> Unit
) {

    private val job = SupervisorJob()
    private val scope = CoroutineScope(Dispatchers.Default + job)

```

```

private val fedSelection: MutableStateFlow<Selection> =
MutableStateFlow(listOf())
protected var currentIteration = 0
private var isFeedingStopped = false

init {
    observeSelection()
}

fun feed(text: String) {
    if (isFeedingStopped) return
    if (matchesPattern(text)) {
        fedSelection.update { selection ->
            selection.plus(listOf(text.toList()))
        }
    }
}

protected abstract fun findBestMatch(selection: Selection): String

private fun observeSelection() {
    scope.launch {
        fedSelection.collectLatest { selection ->
            currentIteration++
            findBestMatch(selection = selection).let { bestMatch ->
                if ((bestMatch.length == pattern.length && currentIteration
<= iterationLimitMin) || currentIteration >= iterationLimitMax) {
                    isFeedingStopped = true
                    bestMatch.filterNot(pattern::contains).let(onSuccess)
                    return@collectLatest
                }
            }
        }
    }
}

private fun matchesPattern(text: String): Boolean {
    if (text.length != pattern.length) return false
    for (i in pattern.indices) { // check for a pattern
        if (pattern[i] != DIGIT_SYMBOL && text[i] != pattern[i]) return
false
    }
}

```

```

        return true
    }

    companion object {

        const val DIGIT_SYMBOL = 'x'
    }
}

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-feature android:name="android.hardware.camera" />
    <uses-permission android:name="android.permission.CAMERA" />
</manifest>

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.VIBRATE" />
</manifest>

package com.zhiroke.core.common.base.error

import com.zhiroke.core.common.base.BaseEvent

interface ErrorEvent : BaseEvent {
    val errorMessage: String?
}

data class DefaultErrorEvent(
    override val errorMessage: String?
) : ErrorEvent

package com.zhiroke.core.common.base.error

import com.zhiroke.core.common.base.BaseEvent
import kotlinx.coroutines.flow.Flow

```

```

import kotlinx.coroutines.flow.flowOf

interface ErrorInteractor<Event> {

    suspend fun tryCatch(
        onFailureHandler: ((e: Exception) -> ErrorEvent)? = null,
        block: suspend () -> Event
    ): Event
}

interface SuspendingErrorInteractor : ErrorInteractor<BaseEvent> {

    /**
     * Executes a block, in case of failure, we handle it with a defined or
     * default handler
     */
    override suspend fun tryCatch(
        onFailureHandler: ((e: Exception) -> ErrorEvent)?,
        block: suspend () -> BaseEvent
    ): BaseEvent {
        return try {
            block.invoke()
        } catch (e: Exception) {
            e.printStackTrace()
            onFailureHandler?.invoke(e) ?: DefaultErrorEvent(errorMessage =
e.message)
        }
    }
}

interface SubscriptionErrorInteractor : ErrorInteractor<Flow<BaseEvent>> {

    /**
     * Wrapping in result in Flow, implementation for @see
     * SubscriptionInteractor
     */
    override suspend fun tryCatch(
        onFailureHandler: ((e: Exception) -> ErrorEvent)?,
        block: suspend () -> Flow<BaseEvent>
    ): Flow<BaseEvent> {
        return try {
            block.invoke()

```

```

        } catch (e: Exception) {
            e.printStackTrace()
            flowOf(onFailureHandler?.invoke(e) ?: DefaultErrorEvent(errorMessage
= e.message))
        }
    }
}

```

```
package com.zhiroke.core.common.base.error
```

```

interface ErrorState {
    val errorMessage: String?
}

```

```
package com.zhiroke.core.common.base
```

```
interface BaseEvent
```

```
package com.zhiroke.core.common.base
```

```
import kotlinx.coroutines.flow.Flow
```

```
interface BaseInteractor<in State : BaseState, in Event: BaseEvent> {
```

```

    fun canHandle(event: Event): Boolean
}

```

```

interface SuspendingInteractor<in State : BaseState, in Event : BaseEvent>:
BaseInteractor<State, Event> {

```

```

    suspend fun invoke(state: State, event: Event): BaseEvent
}

```

```

interface SubscriptionInteractor<in State : BaseState, in Event : BaseEvent> :
BaseInteractor<State, Event> {

```

```

    suspend fun invoke(state: State, event: Event): Flow<BaseEvent>
}

```

```
package com.zhiroke.core.common.base
```

```
interface BaseReducer<State : BaseState> {

    val initialState: State

    fun reduce(state: State, event: BaseEvent): State
}
```

```
package com.zhiroke.core.common.base
```

```
interface BaseState
```

```
package com.zhiroke.core.common.base
```

```
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.zhiroke.core.common.coroutines.DispatchersProvider
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.flow.update
```

```
abstract class BaseViewModel<State: BaseState, Event : BaseEvent>(
    private val dispatchers: DispatchersProvider,
    private val reducer: BaseReducer<State>,
    private val interactors: Set<BaseInteractor<State, Event>>,
    initialEvent: Event
) : ViewModel() {

    private val currentSubscriptions: HashMap<SubscriptionInteractor<State,
Event>, Job> = hashMapOf()
    private val _state = MutableStateFlow(reducer.initialState)
    val state: StateFlow<State> get() = _state

    init {
        sendEvent(event = initialEvent)
    }
}
```

```

@Suppress("UNCHECKED_CAST")
protected fun sendEvent(event: Event) {
    _state.update { state -> reducer.reduce(state = state, event = event) }
    viewModelScope.launch {
        val deferred = mutableListOf<Deferred<Unit>>()
        interactors.filter { interactor -> interactor.canHandle(event =
event) }.forEach { interactor ->
            when(interactor) {
                is SuspendableInteractor<State, Event> -> deferred.add(
                    async(dispatchers.io) {
                        sendEvent(event = interactor.invoke(state =
_state.value, event = event) as Event)
                    }
                )
                is SubscriptionInteractor<State, Event> -> {
                    currentSubscriptions[interactor]?.cancel() // prevent
redundant subscription
                    currentSubscriptions[interactor] =
launch(dispatchers.io) {
                        interactor.invoke(state = _state.value, event =
event).collectLatest { event ->
                            sendEvent(event = event as Event)
                        }
                    }
                }
            }
        }
        deferred.forEach { it.await() }
    }
}

override fun onCleared() {
    viewModelScope.cancel()
    super.onCleared()
}
}

```

```
package com.zhiroke.core.common.coroutines
```

```
import kotlinx.coroutines.CoroutineDispatcher
import kotlinx.coroutines.Dispatchers
```

```

interface DispatchersProvider {
    val io: CoroutineDispatcher
    val main: CoroutineDispatcher
    val default: CoroutineDispatcher
}

class AppDispatchersProvider : DispatchersProvider {

    override val io: CoroutineDispatcher
        get() = Dispatchers.IO
    override val main: CoroutineDispatcher
        get() = Dispatchers.Main
    override val default: CoroutineDispatcher
        get() = Dispatchers.Default
}

package com.zhiroke.core.common.di.modules

import com.zhiroke.core.common.coroutines.AppDispatchersProvider
import com.zhiroke.core.common.coroutines.DispatchersProvider
import org.koin.core.module.dsl.bind
import org.koin.core.module.dsl.factoryOf
import org.koin.dsl.module

fun commonModule() = module {

    factoryOf(::AppDispatchersProvider) { bind<DispatchersProvider>() }

}

package com.zhiroke.core.common.permissions

import androidx.compose.runtime.Composable
import com.google.accompanist.permissions.*

@OptIn(ExperimentalPermissionsApi::class)
@Composable
fun RequestPermissionHelper(

```

```

        permission: String,
        permissionNotGrantedContent: @Composable (requestPermission: () -> Unit) ->
Unit = { },
        permissionGrantedContent: @Composable () -> Unit = { }
    ) {
        val permissionState = rememberPermissionState(permission)

        if (permissionState.status.isGranted) {
            permissionGrantedContent.invoke()
        } else {
            permissionNotGrantedContent.invoke {
                permissionState.launchPermissionRequest()
            }
        }
    }
}

```

```

package com.zhiroke.core.common.utils

```

```

import android.content.ClipData
import android.content.ClipboardManager
import android.content.Context
import android.os.Build
import android.os.VibrationEffect
import android.os.Vibrator
import android.os.VibratorManager

```

```

fun Context.copyToClipboardWithVibration(text: CharSequence) {
    copyToClipboard(text)
    vibrate()
}

```

```

fun Context.copyToClipboard(text: CharSequence) {
    val clipboard = getSystemService(Context.CLIPBOARD_SERVICE) as
ClipboardManager
    val clip = ClipData.newPlainText("label", text)
    clipboard.setPrimaryClip(clip)
}

```

```

fun Context.vibrate() {
    when {

```

```

        Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
            val vibratorManager =
                getSystemService(Context.VIBRATOR_MANAGER_SERVICE) as VibratorManager

            vibratorManager.defaultVibrator.vibrate(VibrationEffect.createOneShot(200,
                VibrationEffect.DEFAULT_AMPLITUDE))
        }
    } else -> {
        val vibratorManager = getSystemService(Context.VIBRATOR_SERVICE) as
        Vibrator

        vibratorManager.vibrate(VibrationEffect.createOneShot(200,
            VibrationEffect.DEFAULT_AMPLITUDE))
    }
}

```

```
package com.zhiroke.core.common.utils
```

```

inline fun <T> T.ifThen(predicate: () -> Boolean, other: T.() -> T): T {
    return if (predicate.invoke()) {
        other.invoke(this)
    } else this
}

```

```

inline fun <T> T.ifThen(predicate: Boolean, other: T.() -> T): T {
    return if (predicate) {
        other.invoke(this)
    } else this
}

```

```

inline fun <T, R> R.ifNotNullThen(value: T?, other: R.(T) -> R): R {
    return if (value != null) {
        other.invoke(this, value)
    } else this
}

```

```
package com.zhiroke.core.common.utils
```

```
/**
```

```

* Use case:
* val response: String? = getString()
* val result = response?.let { handling } otherwise { handling }
*/
internal infix fun <T> T?.otherwise(block: () -> T): T {
    return this ?: block.invoke()
}

package com.zhiroke.core.common.utils

import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember

@Composable
inline fun <T> rememberLambda(lambda: T) = remember { lambda }

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

</manifest>

package com.zhiroke.core.components.animations

import androidx.compose.animation.core.*
import androidx.compose.foundation.background
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.geometry.Offset
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.Color

@Composable
fun animatedShimmerBrush(
    shimmerColors: List<Color> = listOf(
        Color.LightGray.copy(alpha = 0.5f),
        Color.LightGray.copy(alpha = 0.2f),
        Color.LightGray.copy(alpha = 0.5f),

```

```

    )
): Brush {
    val transition = rememberInfiniteTransition()
    val translateAnim = transition.animateFloat(
        initialValue = 0f,
        targetValue = 1000f,
        animationSpec = infiniteRepeatable(
            animation = tween(
                durationMillis = 1200,
                easing = FastOutSlowInEasing
            ),
            repeatMode = RepeatMode.Reverse
        )
    )

    return Brush.linearGradient(
        colors = shimmerColors,
        start = Offset.Zero,
        end = Offset(x = translateAnim.value, y = translateAnim.value)
    )
}

```

@Composable

```
fun ShimmeringTextSize(modifier: Modifier = Modifier, text: String) {
```

```

    Text(
        modifier = Modifier
            .background(brush = animatedShimmerBrush())
            .then(modifier),
        text = text,
        color = Color.Transparent
    )
}

```

```
package com.zhiroke.core.components.buttons
```

```

import androidx.annotation.StringRes
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text

```

```
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.tooling.preview.Preview
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes
```

```
@Composable
fun ButtonText(
    modifier: Modifier = Modifier,
    text: String,
    enabled: Boolean = true,
    onClick: () -> Unit,
) {

    val buttonColors = ButtonDefaults.buttonColors(
        containerColor = MaterialColor.secondary,
        contentColor = MaterialColor.onSecondary
    )

    Button(
        modifier = modifier,
        colors = buttonColors,
        shape = MaterialShapes.small,
        onClick = onClick,
        enabled = enabled,
    ) {

        Text(text = text)
    }
}
```

```
@Composable
fun ButtonText(
    modifier: Modifier = Modifier,
    @StringRes textResId: Int,
    onClick: () -> Unit,
    enabled: Boolean = true,
) {

    ButtonText(
```

```

        modifier = modifier,
        text = stringResource(id = textResId),
        onClick = onClick,
        enabled = enabled,
    )
}

@Preview
@Composable
private fun ButtonTextPreview() {
    ButtonText(text = "Create") {

    }
}

package com.zhiroke.core.components.card

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.BoxScope
import androidx.compose.foundation.layout.aspectRatio
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier

@Composable
fun EuropeanCardRatioContainer(modifier: Modifier = Modifier, content:
@Composable BoxScope.() -> Unit) {

    Box(
        modifier = modifier
            .fillMaxWidth()
            .aspectRatio(ratio = 1.647f),
        content = content
    )
}

package com.zhiroke.core.components.cardwrapper

import androidx.compose.foundation.gestures.detectDragGestures

```



```

        }
    }
    .graphicsLayer {
        transformOrigin = TransformOrigin(0.5f, 0.5f)
        rotationX = rotateX
        rotationY = rotateY
        cameraDistance = 8f * density
    }
) {

    cardContent.invoke()
}
}
}

```

```
package com.zhiroke.core.components.column
```

```

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.ColumnScope
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier

```

```
@Composable
```

```

inline fun ScrollableColumn(
    modifier: Modifier = Modifier,
    verticalArrangement: Arrangement.Vertical = Arrangement.Top,
    horizontalAlignment: Alignment.Horizontal = Alignment.Start,
    content: @Composable ColumnScope.() -> Unit
) {

    val state = rememberScrollState()

    Column(
        modifier = modifier.verticalScroll(state = state, reverseScrolling =
true),
        verticalArrangement = verticalArrangement,
        horizontalAlignment = horizontalAlignment,

```

```

        content = content
    )
}

package com.zhiroke.core.components.pager

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import com.google.accompanist.pager.*
import com.zhiroke.core.theme.dimensions.dp_2
import com.zhiroke.core.theme.utils.MaterialColor

@OptIn(ExperimentalPagerApi::class)
@Composable
fun HorizontalPagerWithTransition(
    modifier: Modifier = Modifier,
    count: Int,
    content: @Composable PagerScope.(page: Int) -> Unit
) {

    val pagerState = rememberPagerState()

    Column(horizontalAlignment = Alignment.CenterHorizontally) {

        HorizontalPager(
            modifier = modifier,
            count = count,
            state = pagerState,
        ) { page ->

            PagerItemWithTransition(page = page) {

                content.invoke(this@HorizontalPager, page)
            }
        }

        HorizontalPagerIndicator(

```

```

        pagerState = pagerState,
        activeColor = MaterialColor.primary,
        inactiveColor = MaterialColor.primaryContainer,
        indicatorShape = RoundedCornerShape(dp_2)
    )
}
}

package com.zhiroke.core.components.pager

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.BoxScope
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.graphicsLayer
import androidx.compose.ui.util.lerp
import com.google.accompanist.pager.ExperimentalPagerApi
import com.google.accompanist.pager.PagerScope
import com.google.accompanist.pager.calculateCurrentOffsetForPage
import kotlin.math.absoluteValue

@OptIn(ExperimentalPagerApi::class)
@Composable
fun PagerScope.PagerItemWithTransition(
    modifier: Modifier = Modifier,
    page: Int,
    content: @Composable BoxScope.() -> Unit
) {

    StatelessPagerItem(
        modifier = modifier.graphicsLayer {

            val pageOffset = calculateCurrentOffsetForPage(page =
page).absoluteValue

            lerp(
                start = 0.85f,
                stop = 1f,
                fraction = 1f - pageOffset.coerceIn(0f, 1f)
            ).also { scale ->

```

```

        scaleX = scale
        scaleY = scale
    }

    alpha = lerp(
        start = 0.5f,
        stop = 1f,
        fraction = 1f - pageOffset.coerceIn(0f, 1f)
    )
},
content = content
)
}

```

```

@Composable
private fun StatelessPagerItem(
    modifier: Modifier = Modifier,
    content: @Composable BoxScope.() -> Unit
) {

    Box(modifier = modifier) {
        content.invoke(this)
    }
}

```

```
package com.zhiroke.core.components.popup.components
```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.tooling.preview.Preview
import com.zhiroke.core.theme.dimensions.*
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes

```

```

@Composable
internal fun Dash() {

```

```

        Spacer(
            modifier = Modifier
                .padding(vertical = dp_16)
                .size(width = dp_64, height = dp_6)
                .clip(MaterialShapes.extraSmall)
                .background(color = MaterialColor.secondary)
        )
    }

@Preview
@Composable
fun DashPreview() {
    Dash()
}

package com.zhiroke.core.components.popup

import androidx.activity.compose.BackHandler
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.ExperimentalMaterialApi
import androidx.compose.material.ModalBottomSheetLayout
import androidx.compose.material.ModalBottomSheetValue
import androidx.compose.material.rememberModalBottomSheetState
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.ui.Alignment
import androidx.compose.ui.ExperimentalComposeUiApi
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalSoftwareKeyboardController
import com.zhiroke.core.components.popup.components.Dash
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.utils.MaterialColor

@OptIn(ExperimentalMaterialApi::class, ExperimentalComposeUiApi::class)
@Composable
fun PopUp(
    popUpState: PopUpState,
    heightFraction: Float = 0.85f,

```

```

onHide: () -> Unit,
content: @Composable BoxScope.() -> Unit
) {

    val keyboardController = LocalSoftwareKeyboardController.current

    val state = rememberModalBottomSheetState(
        initialValue = ModalBottomSheetValue.Hidden,
        skipHalfExpanded = true,
        confirmStateChange = { newState ->
            if (newState == ModalBottomSheetValue.Hidden) {
                onHide.invoke()
            }
            true
        }
    )

    BackHandler(onBack = onHide, enabled = popUpState.isShown)

    LaunchedEffect(key1 = popUpState) {
        if (popUpState.isShown) {
            state.show()
        } else if (state.isVisible) {
            keyboardController?.hide()
            state.hide()
        } else {
            keyboardController?.hide()
        }
    }

    ModalBottomSheetLayout(
        sheetState = state,
        sheetShape = RoundedCornerShape(topStart = dp_16, topEnd = dp_16),
        sheetContent = {

            Column(
                modifier = Modifier
                    .fillMaxWidth()
                    .fillMaxHeight(fraction = heightFraction),
                horizontalAlignment = Alignment.CenterHorizontally
            ) {

```

```

        Dash()

        Box(modifier = Modifier.fillMaxSize(), content = content)
    }
},
sheetBackgroundColor = MaterialColor.background,
content = {}
)
}

```

```
package com.zhiroke.core.components.popup
```

```

data class PopUpState(
    val isShown: Boolean
) {

    companion object {

        fun initialState() = PopUpState(isShown = false)
    }
}

```

```
package com.zhiroke.core.components.toasts
```

```

import androidx.compose.foundation.layout.*
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_360
import com.zhiroke.core.theme.dimensions.dp_8
import com.zhiroke.core.theme.dimensions.dp_96
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes

```

```

@Composable
fun ErrorToast(text: String?) {

```

```

        text ?: return
    }
    ErrorContainer {
        ErrorBlock(modifier = Modifier.align(Alignment.BottomCenter), text =
text)
    }
}

@Composable
private fun ErrorBlock(modifier: Modifier = Modifier, text: String) {
    Surface(
        modifier = modifier
            .width(width = dp_360)
            .defaultMinSize(minHeight = dp_96)
            .padding(all = dp_16),
        shape = MaterialShapes.large,
        color = MaterialColor.errorContainer,
        contentColor = MaterialColor.onErrorContainer
    ) {
        Text(
            modifier = Modifier.padding(all = dp_8),
            text = text,
        )
    }
}

@Composable
private fun ErrorContainer(content: @Composable BoxScope.() -> Unit) {
    Box(
        modifier = Modifier.fillMaxSize()
    ) {
        content.invoke(this)
    }
}

@Preview
@Composable
fun ErrorPreview() {
    ErrorContainer {
        ErrorBlock(modifier = Modifier.align(Alignment.BottomCenter), text =
"too many attempts!")
    }
}

```

```

package com.zhiroke.core.components.utils

import androidx.compose.runtime.Stable

@Stable
class StableList<T>(private val list: List<T>) : List<T> by list {

    override fun hashCode(): Int {
        return list.hashCode()
    }

    override fun equals(other: Any?): Boolean {
        return (other as? StableList<T>)?.list?.let {
            it == list
        } ?: false
    }
}

fun <T> List<T>.toStableList(): StableList<T> = StableList(list = this)

fun <T> emptyStableList(): StableList<T> = StableList(emptyList())

fun <T> stableListOf(): StableList<T> = StableList(listOf())

fun <T> stableListOf(vararg elements: T): StableList<T> =
    StableList(listOf(*elements))

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

</manifest>

package com.zhiroke.core.navigation.di.modules

import com.zhiroke.core.navigation.domain.NavigationManager
import org.koin.core.module.dsl.singleOf
import org.koin.dsl.module

```

```

fun navigationModule() = module {

    singleOf(::NavigationManager)

}

package com.zhiroke.core.navigation.directions

import androidx.navigation.NamedNavArgument
import com.zhiroke.core.navigation.NavigationCommand

object AuthNavDirections {

    val auth = object : NavigationCommand {
        override val arguments: List<NamedNavArgument> = emptyList()
        override val destination: String = "auth"
    }

}

package com.zhiroke.core.navigation.directions

import androidx.navigation.NamedNavArgument
import com.zhiroke.core.navigation.NavigationCommand

object MyWalletNavDirections {

    val cardCarousel = object : NavigationCommand {
        override val arguments: List<NamedNavArgument> = emptyList()
        override val destination: String = "cardCarousel"
    }

}

package com.zhiroke.core.navigation.domain

import androidx.navigation.NavHostController
import androidx.navigation.NavOptionsBuilder
import com.zhiroke.core.navigation.NavigationCommand

```

```

class NavigationManager {

    private var navController: NavHostController? = null

    fun navigateTo(direction: NavigationCommand, builder: NavOptionsBuilder.() -
> Unit = {}) {
        navController?.navigate(route = direction.destination, builder =
builder)
    }

    fun popBackStack() {
        navController?.popBackStack()
    }

    fun attachNavController(navController: NavHostController) {
        this.navController = navController
    }

    fun detachNavController() {
        navController = null
    }
}

package com.zhiroke.core.navigation.utils

import androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.navigation.NavHostController
import com.zhiroke.core.navigation.domain.NavigationManager
import org.koin.core.context.GlobalContext

fun attachNavController(navController: NavHostController) {
    getNavigationManager().attachNavController(navController = navController)
}

fun detachNavController() {
    getNavigationManager().detachNavController()
}

@Composable

```

```
fun rememberNavigationManager(): NavigationManager {
    return remember(calculation = ::getNavigationManager)
}
```

```
private fun getNavigationManager(): NavigationManager {
    return GlobalContext.get().get<NavigationManager>()
}
```

```
package com.zhiroke.core.navigation
```

```
import androidx.navigation.NamedNavArgument
```

```
interface NavigationCommand {

    val arguments: List<NamedNavArgument>
    val destination: String
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

</manifest>
```

```
package com.zhiroke.core.theme.color
```

```
import androidx.compose.ui.graphics.Color
```

```
/** Specials */
val Black = Color(0xFF000000)
val Gray = Color(0xFF888888)
```

```
package com.zhiroke.core.theme.demensions
```

```
import androidx.compose.ui.unit.dp
```

```
val dp_0 = 0.dp
val dp_2 = 2.dp
val dp_4 = 4.dp
```

```

val dp_6 = 6.dp
val dp_8 = 8.dp
val dp_12 = 12.dp
val dp_16 = 16.dp
val dp_24 = 24.dp
val dp_32 = 32.dp
val dp_48 = 48.dp
val dp_64 = 64.dp
val dp_80 = 80.dp
val dp_96 = 96.dp
val dp_360 = 360.dp

```

```

package com.zhiroke.core.theme.utils

```

```

import androidx.compose.material3.ColorScheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Shapes
import androidx.compose.material3.Typography
import androidx.compose.runtime.Composable

```

```

val MaterialColor: ColorScheme @Composable get() = MaterialTheme.colorScheme
val MaterialTypography: Typography @Composable get() = MaterialTheme.typography
val MaterialShapes: Shapes @Composable get() = MaterialTheme.shapes

```

```

package com.zhiroke.data.base

```

```

import androidx.room.Delete
import androidx.room.Insert
import androidx.room.Update

```

```

interface BaseDao<in T> {

    @Insert
    fun insert(entity: T)

    @Insert
    fun insert(entities: List<T>)

    @Update

```

```

        fun update(entity: T)

        @Update
        fun update(entities: List<T>)

        @Delete
        fun delete(entity: T)

        @Delete
        fun delete(entities: List<T>)
    }

package com.zhiroke.data.datasources.keystore

import android.security.keystore.KeyGenParameterSpec
import android.security.keystore.KeyProperties
import java.io.InputStream
import java.io.OutputStream
import java.security.KeyStore
import javax.crypto.Cipher
import javax.crypto.KeyGenerator
import javax.crypto.SecretKey
import javax.crypto.spec.IvParameterSpec

class KeyStoreHelper {

    private val keyStore = KeyStore.getInstance(KEY_STORE_TYPE).apply {
        load(null) }

    private fun getEncryptionCipher(): Cipher {
        return
        Cipher.getInstance("$ENCRYPTION_ALGORITHM/$ENCRYPTION_MODE/$ENCRYPTION_PADDING")
        .apply {
            init(
                Cipher.ENCRYPT_MODE,
                keyStore.getKey(
                    alias = ALIAS,
                    algorithm = ENCRYPTION_ALGORITHM,
                    mode = ENCRYPTION_MODE,
                    padding = ENCRYPTION_PADDING

```

```

        ),
    )
}

private fun getDecryptionCipherForIv(iv: ByteArray): Cipher {
    return
    Cipher.getInstance("$ENCRYPTION_ALGORITHM/$ENCRYPTION_MODE/$ENCRYPTION_PADDING")
    .apply {
        init(
            Cipher.DECRYPT_MODE,
            keyStore.getKey(
                alias = ALIAS,
                algorithm = ENCRYPTION_ALGORITHM,
                mode = ENCRYPTION_MODE,
                padding = ENCRYPTION_PADDING
            ),
            IvParameterSpec(iv)
        )
    }
}

fun encrypt(outputStream: OutputStream, bytes: ByteArray): ByteArray {
    val encryptionCipher = getEncryptionCipher()
    val encryptedBytes = encryptionCipher.doFinal(bytes)

    outputStream.use { stream ->
        stream.write(encryptionCipher.iv.size)
        stream.write(encryptionCipher.iv)
        stream.write(encryptedBytes.size)
        stream.write(encryptedBytes)
    }
    return encryptedBytes
}

fun decrypt(inputStream: InputStream): ByteArray {
    return inputStream.use { stream ->
        val iv = ByteArray(size = stream.read())
        stream.read(iv)

        val encryptedBytes = ByteArray(size = stream.read())
        stream.read(encryptedBytes)
    }
}

```

```

        getDecryptionCipherForIv(iv = iv).doFinal(encryptedBytes)
    }
}

private companion object {
    const val ENCRYPTION_ALGORITHM = KeyProperties.KEY_ALGORITHM_AES
    const val KEY_STORE_TYPE = "AndroidKeyStore"
    const val ALIAS = "jet_wallet_alias"
    const val ENCRYPTION_PADDING = KeyProperties.ENCRYPTION_PADDING_PKCS7
    const val ENCRYPTION_MODE = KeyProperties.BLOCK_MODE_CBC
}

private fun KeyStore.getKey(
    alias: String,
    algorithm: String,
    mode: String,
    padding: String,
): SecretKey {
    // getExistingKey
    return (getEntry(alias, null) as? KeyStore.SecretKeyEntry)?.secretKey
        ?: createDecryptEncryptKey( // createNewKey
            alias = alias,
            algorithm = algorithm,
            mode = mode,
            padding = padding
        )
}

private fun createDecryptEncryptKey(
    alias: String,
    algorithm: String,
    mode: String,
    padding: String,
): SecretKey {
    return KeyGenerator.getInstance(algorithm).apply {
        init(
            KeyGenParameterSpec.Builder(alias, KeyProperties.PURPOSE_ENCRYPT or
            KeyProperties.PURPOSE_DECRYPT)
                .setRandomizedEncryptionRequired(true)
                .setUserAuthenticationRequired(false)

```

```

        .setEncryptionPaddings(padding)
        .setBlockModes(mode)
        .build()
    )
    }.generateKey()
}

package com.zhiroke.data.datasources.room.dao

import androidx.room.Dao
import androidx.room.Query
import com.zhiroke.data.base.BaseDao
import com.zhiroke.data.models.BankCardLocal
import kotlinx.coroutines.flow.Flow

@Dao
interface BankCardDao : BaseDao<BankCardLocal> {

    @Query("DELETE FROM ${BankCardLocal.TABLE_NAME} WHERE\n${BankCardLocal.COLUMN_ID} = :id")
    fun deleteBankCardById(id: String)

    @Query("SELECT * FROM ${BankCardLocal.TABLE_NAME}")
    fun readAll(): Flow<List<BankCardLocal>>

    @Query("DELETE FROM ${BankCardLocal.TABLE_NAME}")
    fun dropTable()
}

package com.zhiroke.data.datasources.room.database

import androidx.room.Database
import androidx.room.RoomDatabase
import com.zhiroke.data.models.BankCardLocal
import com.zhiroke.data.datasources.room.dao.BankCardDao

internal const val WALLET_DATABASE_NAME = "wallet_database"
private const val WALLET_DATABASE_VERSION = 1

```

```

@Database(
    entities = [BankCardLocal::class],
    version = WALLET_DATABASE_VERSION,
    exportSchema = true,
)
internal abstract class WalletDatabase : RoomDatabase() {

    abstract fun getBankCardDao(): BankCardDao
}

package com.zhiroke.data.datasources.securedprefs

import android.content.SharedPreferences
import com.zhiroke.data.util.drop
import com.zhiroke.data.util.get
import com.zhiroke.data.util.set

interface SecuredPrefs {

    suspend fun setUnlockTime(timeInMillis: Long)

    suspend fun getUnlockTime(): Long?

    suspend fun setPIN(pin: String)

    suspend fun getPIN(): String?

    suspend fun dropData()
}

class SecuredPrefsImpl(
    private val prefs: SharedPreferences
) : SecuredPrefs {

    override suspend fun setUnlockTime(timeInMillis: Long) {
        prefs[UNLOCK_TIME] = timeInMillis
    }

    override suspend fun getUnlockTime(): Long? {
        return prefs[UNLOCK_TIME]
    }
}

```

```

        override suspend fun setPIN(pin: String) {
            prefs[PIN] = pin
        }

        override suspend fun getPIN(): String? {
            return prefs[PIN]
        }

        override suspend fun dropData() {
            prefs.drop()
        }

        companion object {
            const val UNLOCK_TIME = "unlock_time"
            const val PIN = "pin"
        }
    }
}

package com.zhiroke.data.di.modules

import android.content.Context
import android.content.SharedPreferences
import androidx.room.Room
import androidx.security.crypto.EncryptedSharedPreferences
import androidx.security.crypto.MasterKey
import com.zhiroke.data.datasources.keystore.KeyStoreHelper
import com.zhiroke.data.datasources.room.dao.BankCardDao
import com.zhiroke.data.datasources.room.database.WALLET_DATABASE_NAME
import com.zhiroke.data.datasources.room.database.WalletDatabase
import com.zhiroke.data.datasources.securedprefs.SecuredPrefs
import com.zhiroke.data.datasources.securedprefs.SecuredPrefsImpl
import com.zhiroke.data.util.PasswordProvider
import net.sqlcipher.database.SupportFactory
import org.koin.core.module.dsl.bind
import org.koin.core.module.dsl.factoryOf
import org.koin.dsl.module

fun dataModule() = module {

    factoryOf(::provideWalletRoomDatabase)

```

```

        factoryOf(::provideBankCardDao)
        factoryOf(::provideEncryptedSharedPreferences)
        factoryOf(::KeyStoreHelper)
        factoryOf(::PasswordProvider)
        factoryOf(::SecuredPrefsImpl) { bind<SecuredPrefs>() }

    }

    internal fun provideWalletRoomDatabase(context: Context, passwordProvider:
    PasswordProvider): WalletDatabase {
        val password = passwordProvider.getPassword(fileName =
        "wallet_db_password.bin")
        return Room.databaseBuilder(context, WalletDatabase::class.java,
        WALLET_DATABASE_NAME).run {
            openHelperFactory(SupportFactory(password))
            build()
        }
    }

    private fun provideBankCardDao(walletDatabase: WalletDatabase): BankCardDao {
        return walletDatabase.getBankCardDao()
    }

    internal fun provideEncryptedSharedPreferences(context: Context):
    SharedPreferences {
        val masterKey = MasterKey.Builder(context)
            .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
            .build()

        return EncryptedSharedPreferences.create(
            context,
            "secured_user_prefs",
            masterKey,
            EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
            EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
        )
    }

    package com.zhiroke.data.models

    import androidx.room.ColumnInfo

```

```

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = BankCardLocal.TABLE_NAME)
data class BankCardLocal(
    @PrimaryKey
    @ColumnInfo(name = COLUMN_ID) val id: String,
    @ColumnInfo(name = COLUMN_NUMBER) val number: String,
    @ColumnInfo(name = COLUMN_CARDHOLDER_NAME) val cardholderName: String,
    @ColumnInfo(name = COLUMN_EXPIRATION_DATE) val expirationDate: String,
    @ColumnInfo(name = COLUMN_VERIFICATION_NUMBER) val verificationNumber:
String,
    @ColumnInfo(name = COLUMN_SKIN_URI) val skinUri: String?,
) {

    internal companion object {

        const val TABLE_NAME = "bank_card"

        const val COLUMN_ID = "id"
        const val COLUMN_NUMBER = "number"
        const val COLUMN_CARDHOLDER_NAME = "cardholder_name"
        const val COLUMN_EXPIRATION_DATE = "expiration_date"
        const val COLUMN_VERIFICATION_NUMBER = "verification_number"
        const val COLUMN_SKIN_URI = "skin_uri"
    }
}

package com.zhiroke.data.util

import android.content.Context
import com.zhiroke.data.datasources.keystore.KeyStoreHelper
import java.io.File
import java.io.FileInputStream
import java.io.FileOutputStream
import java.security.SecureRandom

class PasswordProvider(private val context: Context, private val keyStoreHelper:
KeyStoreHelper) {

    fun getPassword(fileName: String): ByteArray {

```

```

        val file = File(context.filesDir, fileName)
        return if (file.exists()) {
            keyStoreHelper.decrypt(inputStream = FileInputStream(file))
        } else {
            val password = generatePassword()
            keyStoreHelper.encrypt(bytes = password, outputStream =
FileOutputStream(file))
            password
        }
    }
}

private fun generatePassword(): ByteArray {
    val random = SecureRandom.getInstanceStrong()
    val result = ByteArray(32)

    random.nextBytes(result)

    // filter out zero byte values, as SQLCipher does not like them
    while (result.contains(0)) {
        random.nextBytes(result)
    }

    return result
}
}

package com.zhiroke.data.util

import android.content.SharedPreferences

internal operator fun SharedPreferences.set(key: String, value: Any?) {
    when (value) {
        is String? -> edit { it.putString(key, value) }
        is Int -> edit { it.putInt(key, value) }
        is Boolean -> edit { it.putBoolean(key, value) }
        is Float -> edit { it.putFloat(key, value) }
        is Long -> edit { it.putLong(key, value) }
        else -> throw UnsupportedOperationException("Not yet implemented")
    }
}
}

```

```

internal inline fun SharedPreferences.edit(operation: (SharedPreferences.Editor)
-> Unit) {
    val editor = this.edit()
    operation(editor)
    editor.apply()
}

internal inline operator fun <reified T : Any?> SharedPreferences.get(
    key: String,
    defaultValue: T? = null
): T {
    return when (T::class) {
        String::class -> getString(key, defaultValue as? String) as T
        Int::class -> getInt(key, defaultValue as? Int ?: -1) as T
        Boolean::class -> getBoolean(key, defaultValue as? Boolean ?: false) as
T
        Float::class -> getFloat(key, defaultValue as? Float ?: -1f) as T
        Long::class -> getLong(key, defaultValue as? Long ?: -1) as T
        else -> throw UnsupportedOperationException("Not yet implemented")
    }
}

internal fun SharedPreferences.drop() {
    edit().clear().apply()
}

package com.zhiroke.data.datasources.room

import android.content.Context
import androidx.room.Room
import androidx.test.core.app.ApplicationProvider
import androidx.test.ext.junit.runners.AndroidJUnit4
import com.google.common.truth.Truth.assertThat
import com.zhiroke.data.datasources.keystore.CryptoManager
import com.zhiroke.data.datasources.room.database.WALLET_DATABASE_NAME
import com.zhiroke.data.datasources.room.database.WalletDatabase
import com.zhiroke.data.di.modules.provideWalletRoomDatabase
import com.zhiroke.data.models.BankCardLocal
import com.zhiroke.data.util.PasswordProvider
import kotlinx.coroutines.flow.firstOrNull
import kotlinx.coroutines.runBlocking

```

```

import org.junit.After
import org.junit.Test
import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)
class WalletDatabaseTest {

    private val context = ApplicationProvider.getApplicationContext<Context>()
    private val passwordProvider = PasswordProvider(context = context,
cryptoManager = CryptoManager())
    private val securedDatabase = provideWalletRoomDatabase(
        context = context,
        passwordProvider = passwordProvider
    )

    @Test
    fun reading_of_encrypted_data_failed() {
        val securedBankCardDao = securedDatabase.getBankCardDao()
        val commonBankCardDao = Room.databaseBuilder(context,
WalletDatabase::class.java, WALLET_DATABASE_NAME).build().getBankCardDao()
        val bankCard = BankCardLocal("1", "1", "1", "1","1", "1")
        runBlocking {
            securedBankCardDao.insert(entity = bankCard)
            commonBankCardDao.readAll().firstOrNull()?.forEach { readBankCard ->
                if (readBankCard == bankCard) {
                    assertThat(true).isFalse()
                    return@runBlocking
                }
            }
            assertThat(false).isFalse()
        }
    }

    @Test
    fun reading_of_decrypted_data_succeeded() {
        val securedBankCardDao = securedDatabase.getBankCardDao()
        val bankCard = BankCardLocal("2", "2", "2", "2","2", "2")
        runBlocking {
            securedBankCardDao.insert(entity = bankCard)
            val list = securedBankCardDao.readAll().firstOrNull()
            list?.forEach { readBankCard ->
                if (readBankCard == bankCard) {

```

```

        assertTrue().isTrue()
        return@runBlocking
    }
}
    assertTrue(false).isTrue()
}
}

@After
fun deleteDatabase() {
    context.deleteDatabase(WALLET_DATABASE_NAME)
}
}

package com.zhiroke.data.datasources.securedprefs

import android.content.Context
import androidx.test.core.app.ApplicationProvider
import androidx.test.ext.junit.runners.AndroidJUnit4
import com.google.common.truth.Truth.assertThat
import com.zhiroke.data.di.modules.provideEncryptedSharedPreferences
import kotlinx.coroutines.runBlocking
import org.junit.After
import org.junit.Test
import org.junit.runner.RunWith

@RunWith(AndroidJUnit4::class)
class SecuredPrefsTest {

    private val context = ApplicationProvider.getApplicationContext<Context>()
    private val encryptedSharedPreferences =
provideEncryptedSharedPreferences(context)
    private val sharedPreferences = context.getSharedPreferences(
        "secured_user_prefs",
        Context.MODE_PRIVATE
    )

    @Test
    fun reading_of_encrypted_data_failed() {
        val securedPref: SecuredPrefs = SecuredPrefsImpl(prefs =
encryptedSharedPreferences)
    }
}

```

```

        val commonPrefs: SecuredPrefs = SecuredPrefsImpl(prefs =
sharedPreferences)
        val pin = "1234"
        runBlocking {
            securedPref.setPIN(pin = pin)
            assertThat(commonPrefs.getPIN()).isNotEqualTo(pin)
        }
    }
}

```

```

@Test
fun reading_of_decrypted_data_succeeded() {
    val securedPref: SecuredPrefs = SecuredPrefsImpl(prefs =
encryptedSharedPreferences)
    val pin = "1234"
    runBlocking {
        securedPref.setPIN(pin = pin)
        assertThat(securedPref.getPIN()).isEqualTo(pin)
    }
}

```

```

@After
fun deleteSharedPrefs() {
    context.deleteSharedPreferences("secured_user_prefs")
}
}

```

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

</manifest>

```

```
package com.zhiroke.domain.di.modules
```

```

import com.zhiroke.domain.repository.user.UserDataRepository
import com.zhiroke.domain.repository.user.UserDataRepositoryImpl
import com.zhiroke.domain.repository.wallet.bankcard.BankCardRepository
import com.zhiroke.domain.repository.wallet.bankcard.BankCardRepositoryImpl
import org.koin.core.module.dsl.bind
import org.koin.core.module.dsl.singleOf
import org.koin.dsl.module

```

```

fun repositoriesModule() = module {

    singleOf(::BankCardRepositoryImpl) { bind<BankCardRepository>() }
    singleOf(::UserDataRepositoryImpl) { bind<UserDataRepository>() }

}

package com.zhiroke.domain.models

import android.net.Uri
import androidx.compose.runtime.Stable
import androidx.core.net.toUri
import com.zhiroke.data.models.BankCardLocal

@Stable
data class BankCard(
    internal val id: String,
    val number: String,
    val cardholderName: String,
    val expirationDate: String,
    val verificationNumber: String,
    val skinUri: Uri?,
) {

    companion object {

        fun empty() = BankCard(id = "", number = "", cardholderName = "",
            expirationDate = "", verificationNumber = "", skinUri = null)

        fun dummy() = BankCard(id = "1", number = "1234123412341234",
            cardholderName = "Egor Zhir", expirationDate = "1212", verificationNumber =
            "123", skinUri = null)
    }
}

internal fun BankCard.toLocal() = BankCardLocal(
    id = id,
    number = number,
    cardholderName = cardholderName,

```

```

        expirationDate = expirationDate,
        verificationNumber = verificationNumber,
        skinUri = skinUri?.toString(),
    )

internal fun BankCardLocal.toUi() = BankCard(
    id = id,
    number = number,
    cardholderName = cardholderName,
    expirationDate = expirationDate,
    verificationNumber = verificationNumber,
    skinUri = skinUri?.toUri()
)

package com.zhiroke.domain.repository.user

interface UserDataRepository {

    suspend fun setUnlockTime(timeInMillis: Long)

    suspend fun getUnlockTime(): Long?

    suspend fun setPIN(pin: String)

    suspend fun getPIN(): String?

    suspend fun dropData()
}

package com.zhiroke.domain.repository.user

import com.zhiroke.data.datasources.room.dao.BankCardDao
import com.zhiroke.data.datasources.securedprefs.SecuredPrefs

class UserDataRepositoryImpl(
    private val bankCardDao: BankCardDao,
    private val securedPrefs: SecuredPrefs
) : UserDataRepository {

    override suspend fun setUnlockTime(timeInMillis: Long) {

```

```

        securedPrefs.setUnlockTime(timeInMillis = timeInMillis)
    }

    override suspend fun getUnlockTime(): Long? {
        return securedPrefs.getUnlockTime()
    }

    override suspend fun setPIN(pin: String) {
        securedPrefs.setPIN(pin = pin)
    }

    override suspend fun getPIN(): String? {
        return securedPrefs.getPIN()
    }

    override suspend fun dropData() {
        bankCardDao.dropTable()
        securedPrefs.dropData()
    }
}

```

```
package com.zhiroke.domain.repository.wallet.bankcard
```

```
import com.zhiroke.domain.models.BankCard
import kotlinx.coroutines.flow.Flow
```

```
interface BankCardRepository {

    suspend fun addBankCard(bankCard: BankCard)

    suspend fun updateBankCard(bankCard: BankCard)

    suspend fun deleteBankCard(bankCard: BankCard)

    fun fetchAll(): Flow<List<BankCard>>
}

```

```
package com.zhiroke.domain.repository.wallet.bankcard
```

```
import com.zhiroke.data.datasources.room.dao.BankCardDao
```

```

import com.zhiroke.domain.models.BankCard
import com.zhiroke.domain.models.toLocal
import com.zhiroke.domain.models.toUi
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.flow.*

internal class BankCardRepositoryImpl(private val bankCardDao: BankCardDao) :
BankCardRepository {

    override suspend fun addBankCard(bankCard: BankCard) {
        bankCardDao.insert(bankCard.toLocal())
    }

    override suspend fun updateBankCard(bankCard: BankCard) {
        bankCardDao.update(bankCard.toLocal())
    }

    override suspend fun deleteBankCard(bankCard: BankCard) {
        bankCardDao.deleteBankCardById(id = bankCard.id)
    }

    @OptIn(ExperimentalCoroutinesApi::class)
    override fun fetchAll(): Flow<List<BankCard>> {
        return bankCardDao.readAll().flatMapLatest { list -> flowOf(list.map {
localCard -> localCard.toUi() }) }
    }
}

```

```

<?xml version="1.0" encoding="utf-8"?>
<manifest>
</manifest>

```

```

package com.zhiroke.presentation.auth.di.modules

```

```

import org.koin.dsl.module

```

```

fun authModules() = module {
    includes(
        interactorsModule(),
        reducersModule(),

```

```

        utilsModule(),
        viewModelsModule()
    )
}

```

```
package com.zhiroke.presentation.auth.di.modules
```

```

import com.zhiroke.presentation.auth.presentation.auth.interactors.*
import org.koin.core.module.dsl.factoryOf
import org.koin.dsl.module

```

```

internal fun interactorsModule() = module {

    factoryOf(::KeyboardHandlerInteractor)
    factoryOf(::BiometricAuthInteractor)
    factoryOf(::CheckForBioAuthSupportInteractor)
    factoryOf(::SucceededToAuthInteractor)
    factoryOf(::ConfirmPINInteractor)
    factoryOf(::GetPINInteractor)
    factoryOf(::GetUnlockPINTimeInteractor)
    factoryOf(::PINAuthInteractor)
    factoryOf(::AuthErrorStopperInteractor)
    factoryOf(::HandlePINAuthFailureInteractor)
    factoryOf(::HandleFailedToConfirmPINInteractor)
    factoryOf(::DropAllDataInteractor)

}

```

```
package com.zhiroke.presentation.auth.di.modules
```

```

import com.zhiroke.presentation.auth.presentation.auth.AuthReducer
import org.koin.core.module.dsl.factoryOf
import org.koin.dsl.module

```

```

internal fun reducersModule() = module {

    factoryOf(::AuthReducer)

}

```

```
package com.zhiroke.presentation.auth.di.modules

import com.zhiroke.presentation.auth.utils.managers.BiometricAuthManager
import org.koin.core.module.dsl.singleOf
import org.koin.dsl.module

internal fun utilsModule() = module {

    singleOf(::BiometricAuthManager)

}
```

```
package com.zhiroke.presentation.auth.di.modules

import com.zhiroke.core.common.utils.koin.viewModelOf
import com.zhiroke.presentation.auth.presentation.auth.AuthViewModel
import org.koin.dsl.module

internal fun viewModelsModule() = module {

    viewModelOf(::AuthViewModel)

}
```

```
package com.zhiroke.presentation.auth.presentation.auth.components.buttons

import androidx.annotation.StringRes
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.TextUnit
import com.zhiroke.core.theme.utils.MaterialColor

@Composable
fun TertiaryTextButton(
```

```

        modifier: Modifier = Modifier,
        @StringRes resId: Int,
        fontSize: TextUnit,
        enabled: Boolean = true,
        onClick: () -> Unit
    ) {

        val buttonColors = ButtonDefaults.buttonColors(
            containerColor = MaterialColor.tertiary,
            contentColor = MaterialColor.onTertiary,
            disabledContainerColor = MaterialColor.tertiary.copy(alpha = 0.12f),
            disabledContentColor = MaterialColor.onTertiary.copy(alpha = 0.38f)
        )

        TextButton(
            modifier = modifier,
            colors = buttonColors,
            enabled = enabled,
            onClick = onClick,
        ) {

            Text(text = stringResource(id = resId), fontSize = fontSize)
        }
    }
}

```

```
package com.zhiroke.presentation.auth.presentation.auth.components.buttons
```

```

import androidx.compose.foundation.BorderStroke
import androidx.compose.foundation.interaction.MutableInteractionSource
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Shape
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.style.TextAlign
import com.zhiroke.core.components.R
import kotlinx.coroutines.delay

```

```
@Composable
```

```
fun TextButtonWithTimer(
```

```

        modifier: Modifier = Modifier,
        text: String,
        durationInSeconds: Int,
        onClick: () -> Unit,
        shape: Shape = ButtonDefaults.shape,
        colors: ButtonColors = ButtonDefaults.buttonColors(),
        elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),
        border: BorderStroke? = null,
        contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
        interactionSource: MutableInteractionSource = remember {
MutableInteractionSource() }
    ) {

        val context = LocalContext.current
        var enabled by remember { mutableStateOf(false) }
        var textWithTimer by remember { mutableStateOf(text) }

        LaunchedEffect(key1 = Unit) {
            for (secondsLeft in durationInSeconds downTo 0) {
                if (secondsLeft != 0) {
                    textWithTimer = text +
context.getString(R.string.placeholder_timer, secondsLeft)
                    enabled = false
                } else {
                    textWithTimer = text
                    enabled = true
                }
                delay(1000)
            }
        }

        Button(
            modifier = modifier,
            onClick = onClick,
            enabled = enabled,
            shape = shape,
            colors = colors,
            elevation = elevation,
            border = border,
            contentPadding = contentPadding,
            interactionSource = interactionSource
        ) {

```

```

        Text(
            text = textWithTimer,
            textAlign = TextAlign.Center
        )
    }
}

package com.zhiroke.presentation.auth.presentation.auth.components.dialkeyboard

import androidx.compose.foundation.layout.*
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import com.zhiroke.core.theme.dimensions.dp_24
import com.zhiroke.core.theme.dimensions.dp_48
import com.zhiroke.core.theme.dimensions.dp_80
import com.zhiroke.presentation.auth.R
import
com.zhiroke.presentation.auth.presentation.auth.interactors.PINKeyboardCommand

@Composable
internal fun DialKeyboardBlock(
    modifier: Modifier,
    biometricAuthSupported: Boolean,
    onCommand: (PINKeyboardCommand) -> Unit,
    onBioClick: (PINKeyboardCommand) -> Unit
) {

    val padding = dp_24
    val buttonSize = dp_80
    val fontSize = MaterialTheme.typography.headlineSmall.fontSize

    Column(
        modifier = modifier,
        verticalArrangement = Arrangement.spacedBy(padding),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {

```

```

Row(horizontalArrangement = Arrangement.spacedBy(padding)) {

    DigitDialButton(
        modifier = Modifier.size(buttonSize),
        digit = "1",
        fontSize = fontSize,
        onClick = onCommand
    )

    DigitDialButton(
        modifier = Modifier.size(buttonSize),
        digit = "2",
        fontSize = fontSize,
        onClick = onCommand
    )

    DigitDialButton(
        modifier = Modifier.size(buttonSize),
        digit = "3",
        fontSize = fontSize,
        onClick = onCommand
    )
}

Row(horizontalArrangement = Arrangement.spacedBy(padding)) {

    DigitDialButton(
        modifier = Modifier.size(buttonSize),
        digit = "4",
        fontSize = fontSize,
        onClick = onCommand
    )

    DigitDialButton(
        modifier = Modifier.size(buttonSize),
        digit = "5",
        fontSize = fontSize,
        onClick = onCommand
    )

    DigitDialButton(
        modifier = Modifier.size(buttonSize),

```

```

        digit = "6",
        fontSize = fontSize,
        onClick = onCommand
    )
}

Row(horizontalArrangement = Arrangement.spacedBy(padding)) {

    DigitDialButton(
        modifier = Modifier.size(buttonSize),
        digit = "7",
        fontSize = fontSize,
        onClick = onCommand
    )

    DigitDialButton(
        modifier = Modifier.size(buttonSize),
        digit = "8",
        fontSize = fontSize,
        onClick = onCommand
    )

    DigitDialButton(
        modifier = Modifier.size(buttonSize),
        digit = "9",
        fontSize = fontSize,
        onClick = onCommand
    )
}

Row(horizontalArrangement = Arrangement.spacedBy(padding)) {

    if (biometricAuthSupported) {
        IconDialButton(
            modifier = Modifier.size(buttonSize),
            resId = R.drawable.ic_fingerprint,
            iconSize = dp_48,
            command = PINKeyboardCommand.AuthWithBio,
            onClick = onBioClick
        )
    } else {
        Spacer(modifier = Modifier.size(buttonSize))
    }
}

```

```

    }

    DigitDialButton(
        modifier = Modifier.size(buttonSize),
        digit = "0",
        fontSize = fontSize,
        onClick = onCommand
    )

    IconDialButton(
        modifier = Modifier.size(buttonSize),
        resId = R.drawable.ic_backspace,
        iconSize = dp_24,
        command = PINKeyboardCommand.RemoveCharacter,
        onClick = onCommand
    )
}

}

}

@Preview
@Composable
private fun DialKeyboardBlockPreview() {

    DialKeyboardBlock(modifier = Modifier.fillMaxSize(), true, {}, {})
}

package com.zhiroke.presentation.auth.presentation.auth.components.dialkeyboard

import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.TextUnit
import com.zhiroke.core.common.utils.rememberLambda
import com.zhiroke.core.theme.dimensions.dp_0
import com.zhiroke.core.theme.utils.MaterialColor
import

```

```
com.zhiroke.presentation.auth.presentation.auth.interactors.PINKeyboardCommand
```

```
@Composable
internal fun DigitDialButton(
    modifier: Modifier = Modifier,
    digit: String,
    fontSize: TextUnit,
    onClick: (PINKeyboardCommand) -> Unit
) {

    val onClick = rememberLambda {
        onClick.invoke(PINKeyboardCommand.WriteCharacter(character = digit))
    }

    val buttonColors = ButtonDefaults.buttonColors(
        containerColor = MaterialColor.primary,
        contentColor = MaterialColor.onPrimary
    )

    Button(
        modifier = modifier,
        colors = buttonColors,
        shape = CircleShape,
        onClick = onClick,
        contentPadding = PaddingValues(dp_0)
    ) {

        Text(
            text = digit,
            fontSize = fontSize
        )
    }
}
```

```
package com.zhiroke.presentation.auth.presentation.auth.components.dialkeyboard
```

```
import androidx.annotation.DrawableRes
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material3.Button
```

```

import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Icon
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.unit.Dp
import com.zhiroke.core.common.utils.rememberLambda
import com.zhiroke.core.theme.dimensions.dp_0
import com.zhiroke.core.theme.utils.MaterialColor
import

com.zhiroke.presentation.auth.presentation.auth.interactors.PINKeyboardCommand

@Composable
internal fun IconDialButton(
    modifier: Modifier = Modifier,
    @DrawableRes resId: Int,
    iconSize: Dp,
    command: PINKeyboardCommand,
    onClick: (PINKeyboardCommand) -> Unit
) {

    val onClick = rememberLambda {
        onClick.invoke(command)
    }

    val buttonColors = ButtonDefaults.buttonColors(
        containerColor = MaterialColor.secondary,
        contentColor = MaterialColor.onSecondary
    )

    Button(
        modifier = modifier,
        colors = buttonColors,
        shape = CircleShape,
        onClick = onClick,
        contentPadding = PaddingValues(dp_0)
    ) {

        Icon(
            modifier = Modifier.size(iconSize),
            painter = painterResource(id = resId),
            contentDescription = null // TODO change me

```

```

    )
}
}

```

```
package com.zhiroke.presentation.auth.presentation.auth.components.overlays
```

```

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color

```

```
@Composable
```

```
internal fun LockedScreenOverlay(text: String) {
```

```

    Surface(
        modifier = Modifier.fillMaxSize(),
        color = Color.Black.copy(alpha = 0.9f),
        contentColor = Color.White,

```

```
    ) {
```

```

        Box(
            modifier = Modifier.fillMaxSize(),
            contentAlignment = Alignment.Center

```

```
        ) {
```

```
            Text(text = text)
```

```
        }
```

```
    }
```

```
}
```

```
package com.zhiroke.presentation.auth.presentation.auth.components.passwordfield
```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.shape.RoundedCornerShape

```

```

import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.unit.TextUnit
import com.zhiroke.core.theme.dimensions.dp_64
import com.zhiroke.core.theme.utils.MaterialColor

@Composable
internal fun PINCharacterItem(
    modifier: Modifier,
    char: Char?,
    textFontSize: TextUnit,
    visible: Boolean,
) {

    Surface(
        modifier = modifier
            .clip(RoundedCornerShape(dp_64))
            .background(color = MaterialColor.primary),
        color = MaterialColor.primaryContainer
    ) {

        Box(modifier = Modifier.fillMaxSize(), contentAlignment =
Alignment.Center) {
            if (visible) {
                Text(
                    text = char?.toString() ?: "",
                    fontSize = textFontSize
                )
            } else if (char != null) {
                Spacer(
                    modifier = Modifier
                        .fillMaxSize()
                        .background(color = MaterialColor.onPrimaryContainer)
                )
            }
        }
    }
}

```

```
    }
}
```

```
package com.zhiroke.presentation.auth.presentation.auth.components.passwordfield
```

```
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.size
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_32
import com.zhiroke.core.theme.dimensions.dp_64
```

```
@Composable
```

```
internal fun PINField(
```

```
    modifier: Modifier,
```

```
    pin: String,
```

```
    visible: Boolean
```

```
) {
```

```
    Row(
```

```
        modifier = modifier,
```

```
        horizontalArrangement = Arrangement.spacedBy(dp_16)
```

```
    ) {
```

```
        repeat(4) { index ->
```

```
            PINCharacterItem(
```

```
                modifier = Modifier.size(width = dp_32, height = dp_64),
```

```
                char = pin.getOrNull(index),
```

```
                textFontSize = MaterialTheme.typography.titleLarge.fontSize,
```

```
                visible = visible
```

```
            )
```

```
        }
```

```
    }
```

```
}
```

```
package
```

```
com.zhiroke.presentation.auth.presentation.auth.components.popups.components
```

```

import androidx.compose.material3.ButtonDefaults
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.presentation.auth.R
import
com.zhiroke.presentation.auth.presentation.auth.components.buttons.TextButtonWithTimer

```

```

@Composable
internal fun DropDataButton(
    modifier: Modifier = Modifier,
    onClick: () -> Unit
) {

    val colors = ButtonDefaults.buttonColors(
        contentColor = MaterialColor.onError,
        containerColor = MaterialColor.error,
        disabledContentColor = MaterialColor.onError.copy(alpha = 0.58f),
        disabledContainerColor = MaterialColor.error.copy(alpha = 0.22f)
    )

    TextButtonWithTimer(
        modifier = modifier,
        text = stringResource(id = R.string.button_drop_data_confirm),
        durationInSeconds = 15,
        colors = colors,
        onClick = onClick
    )
}

```

```

package com.zhiroke.presentation.auth.presentation.auth.components.popups

```

```

import androidx.compose.foundation.layout.*
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier

```

```

import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextAlign
import com.zhiroke.core.components.popup.PopUp
import com.zhiroke.core.components.popup.PopUpState
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_64
import com.zhiroke.core.theme.utils.MaterialTypography
import com.zhiroke.presentation.auth.R
import
com.zhiroke.presentation.auth.presentation.auth.components.popups.components.Drop
pDataButton

@Composable
internal fun DropDataPopUp(popUpState: PopUpState, onHide: () -> Unit,
onDropData: () -> Unit) {

    PopUp(popUpState = popUpState, heightFraction = 0.4f, onHide = onHide) {

        Column(
            modifier = Modifier
                .align(Alignment.TopCenter)
                .padding(horizontal = dp_16),
            verticalArrangement = Arrangement.spacedBy(dp_16)
        ) {

            Text(
                modifier = Modifier.align(Alignment.CenterHorizontally),
                text = stringResource(id = R.string.dont_remember_pin),
                style = MaterialTypography.titleLarge
            )

            Text(
                text = stringResource(id = R.string.drop_data_warning),
                style = MaterialTypography.titleMedium
            )
        }

        Row(
            modifier = Modifier
                .wrapContentSize()
                .align(Alignment.BottomCenter)
                .padding(all = dp_16),

```

```

        horizontalArrangement = Arrangement.spacedBy(dp_16)
    ) {

        if (popUpState.isShown) {
            DropDataButton(
                modifier = Modifier
                    .height(dp_64)
                    .weight(1f),
                onClick = onDropData
            )
        }

        Button(
            modifier = Modifier
                .height(dp_64)
                .weight(1f),
            onClick = onHide
        ) {
            Text(
                text = stringResource(id =
R.string.button_drop_data_cancel),
                textAlign = TextAlign.Center
            )
        }
    }
}

```

```
package com.zhiroke.presentation.auth.presentation.auth.interactors
```

```

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState
import kotlinx.coroutines.delay

```

```

internal class AuthErrorStopperInteractor : SuspendableInteractor<AuthState,
AuthEvent>, SuspendableErrorInteractor {

```

```

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {

```

```

        if (event !is AuthEvent.Error || event.errorMessage == null) return
AuthEvent.None
        delay(event.durationInMillis)
        return AuthEvent.Error(errorMessage = null)
    }

    override fun canHandle(event: AuthEvent): Boolean = event is AuthEvent.Error
}

package com.zhiroke.presentation.auth.presentation.auth.interactors

import android.content.Context
import androidx.biometric.BiometricPrompt
import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.presentation.auth.R
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState
import com.zhiroke.presentation.auth.utils.managers.BiometricAuthManager

internal class BiometricAuthInteractor(
    private val biometricAuthManager: BiometricAuthManager,
    private val appContext: Context
) : SuspendableInteractor<AuthState, AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {
        if ((event as? AuthEvent.GetBioAuthSupportState)?.isSupported == false
            || state.screenStatus != AuthState.AuthScreenStatus.LOG_IN
            || !state.loginState.isPINUnlocked) return AuthEvent.None

        return when (biometricAuthManager.auth()) {
            null -> AuthEvent.SucceededToAuth
            BiometricPrompt.ERROR_LOCKOUT -> AuthEvent.Error(errorMessage =
appContext.getString(R.string.error_to_many_attempts))
            else -> AuthEvent.None
        }
    }

    override fun canHandle(event: AuthEvent): Boolean =
        event is AuthEvent.AuthWithBio || event is

```

```
AuthEvent.GetBioAuthSupportState
}
```

```
package com.zhiroke.presentation.auth.presentation.auth.interactors
```

```
import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState
import com.zhiroke.presentation.auth.utils.managers.BiometricAuthManager
```

```
internal class CheckForBioAuthSupportInteractor(
    private val biometricAuthManager: BiometricAuthManager
) : SuspendableInteractor<AuthState, AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {
        val result = biometricAuthManager.isBioAuthSupported()
        return AuthEvent.GetBioAuthSupportState(isSupported = result)
    }

    override fun canHandle(event: AuthEvent): Boolean = event is
AuthEvent.LoadedPIN
}
```

```
package com.zhiroke.presentation.auth.presentation.auth.interactors
```

```
import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.domain.repository.user.UserDataRepository
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState
```

```
internal class ConfirmPINInteractor(
    private val userDataRepository: UserDataRepository
) : SuspendableInteractor<AuthState, AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {
        return if (state.setPINState.pin == state.confirmPINState.pin) {
```

```

        userDataRepository.setPIN(pin = state.confirmPINState.pin)
        AuthEvent.SucceededToAuth
    } else {
        AuthEvent.FailedToConfirmPIN
    }
}

override fun canHandle(event: AuthEvent): Boolean = event is
AuthEvent.ConfirmPIN
}

package com.zhiroke.presentation.auth.presentation.auth.interactors

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.core.navigation.directions.AuthNavDirections
import com.zhiroke.core.navigation.domain.NavigationManager
import com.zhiroke.domain.repository.user.UserDataRepository
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

internal class DropAllDataInteractor(
    private val userDataRepository: UserDataRepository,
    private val navigationManager: NavigationManager
) : SuspendableInteractor<AuthState, AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {
        userDataRepository.dropData()
        CoroutineScope(Dispatchers.Main).launch {
            navigationManager.navigateTo(direction = AuthNavDirections.auth) {
                popUpTo(0) {
                    inclusive = true
                }
            }
        }
        return AuthEvent.None
    }
}

```

```

        override fun canHandle(event: AuthEvent): Boolean = event is
AuthEvent.DropAllData
    }

package com.zhiroke.presentation.auth.presentation.auth.interactors

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.domain.repository.user.UserDataRepository
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState

internal class GetPINInteractor(
    private val userDataRepository: UserDataRepository
) : SuspendableInteractor<AuthState, AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {
        return AuthEvent.LoadedPIN(pin = userDataRepository.getPIN())
    }

    override fun canHandle(event: AuthEvent): Boolean = event is AuthEvent.Load
}

package com.zhiroke.presentation.auth.presentation.auth.interactors

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.domain.repository.user.UserDataRepository
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState

internal class GetUnlockPINTimeInteractor(
    private val userDataRepository: UserDataRepository
) : SuspendableInteractor<AuthState, AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {
        return AuthEvent.LoadedUnlockAttemptTime(time =

```

```

userDataRepository.getUnlockTime() ?: 0)
    }

    override fun canHandle(event: AuthEvent): Boolean = event is AuthEvent.Load
}

```

```
package com.zhiroke.presentation.auth.presentation.auth.interactors
```

```

import android.content.Context
import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.presentation.auth.R
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState

internal class HandleFailedToConfirmPINInteractor(
    private val appContext: Context
) : SuspendableInteractor<AuthState, AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {
        return AuthEvent.Error(errorMessage =
appContext.getString(R.string.error_pins_dont_match))
    }

    override fun canHandle(event: AuthEvent): Boolean = event is
AuthEvent.FailedToConfirmPIN
}

```

```
package com.zhiroke.presentation.auth.presentation.auth.interactors
```

```

import android.content.Context
import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SubscriptionInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.core.common.utils.vibrate
import com.zhiroke.domain.repository.user.UserDataRepository
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState
import kotlinx.coroutines.delay

```

```

import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow
import kotlinx.coroutines.flow.flowOf

internal class HandlePINAuthFailureInteractor(
    private val userDataRepository: UserDataRepository,
    private val appContext: Context
) : SubscriptionInteractor<AuthState, AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent):
Flow<BaseEvent> {
        appContext.vibrate()
        return if (!state.loginState.isPINUnlocked) {
            userDataRepository.setUnlockTime(state.loginState.unlockAttemptTime)
            flow {
                while (!state.loginState.isPINUnlocked) {
                    delay(1000)
                    val leftSeconds = ((state.loginState.unlockAttemptTime -
System.currentTimeMillis()) / 1000).toInt()
                    emit(AuthEvent.ChangeLeftSecondsToBeUnlocked(leftSeconds =
leftSeconds))
                }
                emit(AuthEvent.ChangeLeftSecondsToBeUnlocked(leftSeconds =
null))
            }
        } else flowOf(AuthEvent.None)
    }

    override fun canHandle(event: AuthEvent): Boolean = event is
AuthEvent.FailedPINAuth
        || event is AuthEvent.LoadedUnlockAttemptTime
    }
}

package com.zhiroke.presentation.auth.presentation.auth.interactors

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState

```

```

internal sealed interface PINKeyboardCommand {
    object RemoveCharacter : PINKeyboardCommand
    class WriteCharacter(val character: String) : PINKeyboardCommand
    object AuthWithBio : PINKeyboardCommand
}

internal class KeyboardHandlerInteractor : SuspendableInteractor<AuthState,
AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {

        if (!state.loginState.isPINUnlocked) return AuthEvent.None

        val password = when (state.screenStatus) {
            AuthState.AuthScreenStatus.LOG_IN -> state.loginState.pin
            AuthState.AuthScreenStatus.SET_PIN -> state.setPINState.pin
            AuthState.AuthScreenStatus.CONFIRM_PIN -> state.confirmPINState.pin
        }

        return when (val command = (event as
AuthEvent.ExecuteKeyboardCommand).command) {
            PINKeyboardCommand.RemoveCharacter -> {
                AuthEvent.ChangePIN(newPIN = password.dropLast(1))
            }
            is PINKeyboardCommand.WriteCharacter -> {
                val newPassword = if (password.length >= 4) password else
password + command.character
                AuthEvent.ChangePIN(newPIN = newPassword)
            }
            else -> AuthEvent.None
        }
    }

    override fun canHandle(event: AuthEvent): Boolean = event is
AuthEvent.ExecuteKeyboardCommand
}

package com.zhiroke.presentation.auth.presentation.auth.interactors

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor

```

```

import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.domain.repository.user.UserDataRepository
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState

internal class PINAuthInteractor(
    private val userDataRepository: UserDataRepository
) : SuspendableInteractor<AuthState, AuthEvent>, SuspendableErrorInteractor {

    override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {
        if (state.loginState.pin.length != 4 || state.screenStatus !=
AuthState.AuthScreenStatus.LOG_IN) return AuthEvent.None
        val unlockTimeInMillis = userDataRepository.getUnlockTime()
        if (unlockTimeInMillis != null && unlockTimeInMillis >
System.currentTimeMillis()) return AuthEvent.None

        return if (state.loginState.pin == state.loginState.loadedPIN) {
            AuthEvent.SucceededToAuth
        } else {
            AuthEvent.FailedPINAuth
        }
    }

    override fun canHandle(event: AuthEvent): Boolean = event is
AuthEvent.ChangePIN
}

```

```

package com.zhiroke.presentation.auth.presentation.auth.interactors

```

```

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.core.navigation.directions.MyWalletNavDirections
import com.zhiroke.core.navigation.domain.NavigationManager
import com.zhiroke.presentation.auth.presentation.auth.AuthEvent
import com.zhiroke.presentation.auth.presentation.auth.AuthState
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

```

```

internal class SucceededToAuthInteractor(

```

```

        private val navigationManager: NavigationManager
    ) : SuspendingInteractor<AuthState, AuthEvent>, SuspendingErrorInteractor {

        override suspend fun invoke(state: AuthState, event: AuthEvent): BaseEvent {
            CoroutineScope(Dispatchers.Main).launch {
                navigationManager.navigateTo(direction =
MyWalletNavDirections.cardCarousel) {
                    popUpTo(0) {
                        inclusive = true
                    }
                }
            }
            return AuthEvent.None
        }

        override fun canHandle(event: AuthEvent): Boolean = event is
AuthEvent.SucceededToAuth
    }

```

```

package com.zhiroke.presentation.auth.presentation.auth

```

```

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.error.ErrorEvent
import com.zhiroke.core.components.popup.PopUpState
import
com.zhiroke.presentation.auth.presentation.auth.interactors.PINKeyboardCommand

```

```

internal sealed interface AuthEvent : BaseEvent {
    object None : AuthEvent
    object Load : AuthEvent
    class Error(override val errorMessage: String?, val durationInMillis: Long =
3000) : AuthEvent, ErrorEvent
    class LoadedPIN(val pin: String?) : AuthEvent
    class LoadedUnlockAttemptTime(val time: Long) : AuthEvent
    class GetBioAuthSupportState(val isSupported: Boolean) : AuthEvent
    object DropAllData : AuthEvent
    object ConfirmPIN : AuthEvent
    object AuthWithBio : AuthEvent
    class ExecuteKeyboardCommand(val command: PINKeyboardCommand) : AuthEvent
    class ChangePIN(val newPIN: String) : AuthEvent

```

```

class ChangeDropDataPopUpState(val popUpState: PopUpState) : AuthEvent
class ChangeLeftSecondsToBeUnlocked(val leftSeconds: Int?) : AuthEvent
object SucceededToAuth : AuthEvent
object FailedPINAuth : AuthEvent
object FailedToConfirmPIN : AuthEvent
class ChangeScreenState(val screenStatus: AuthState.AuthScreenState) :
AuthEvent
}

```

```

package com.zhiroke.presentation.auth.presentation.auth

```

```

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.BaseReducer

```

```

internal class AuthReducer : BaseReducer<AuthState> {

    override val initialState: AuthState get() = AuthState.initialState()

    override fun reduce(state: AuthState, event: BaseEvent): AuthState {
        return state.run {
            when(event) {
                AuthEvent.None -> state
                AuthEvent.Load -> state
                AuthEvent.AuthWithBio -> state
                AuthEvent.SucceededToAuth -> state
                AuthEvent.ConfirmPIN -> state
                AuthEvent.DropAllData -> state
                AuthEvent.FailedPINAuth -> state.copy(
                    loginState = loginState.copy(
                        attempt = loginState.attempt + 1,
                        unlockAttemptTime = if (loginState.attempt + 1 >=
AuthState.LoginState.ATTEMPTS_LIMIT) {
                            System.currentTimeMillis() +
AuthState.LoginState.LOCK_DELAY
                        } else loginState.unlockAttemptTime,
                        pin = ""
                    )
                )
                AuthEvent.FailedToConfirmPIN -> state.copy(confirmPINState =
confirmPINState.copy(pin = ""))
            }
        }
    }
}

```

```

        is AuthEvent.Error -> state.copy(errorMessage =
event.errorMessage)
        is AuthEvent.ChangeScreenStatus -> state.copy(screenStatus =
event.screenStatus)
        is AuthEvent.LoadedPIN -> state.copy(
            screenStatus = if(event.pin != null)
AuthState.AuthScreenStatus.LOG_IN else AuthState.AuthScreenStatus.SET_PIN,
            loginState = loginState.copy(loadedPIN = event.pin)
        )
        is AuthEvent.ChangeDropDataPopUpState -> state.copy(
            loginState = loginState.copy(dropDataPopUpState =
event.popUpState)
        )
        is AuthEvent.LoadedUnlockAttemptTime -> state.copy(loginState =
loginState.copy(unlockAttemptTime = event.time))
        is AuthEvent.GetBioAuthSupportState -> state.copy(loginState =
loginState.copy(isBioSupported = event.isSupported))
        is AuthEvent.ChangeLeftSecondsToBeUnlocked -> if
(event.leftSeconds == null) {
            // if we had waited for the blocking delay
            state.copy(loginState = loginState.copy(attempt = 0,
unlockAttemptTime = 0, secondsLeftToBeUnlocked = null))
        } else {
            state.copy(loginState =
loginState.copy(secondsLeftToBeUnlocked = event.leftSeconds))
        }
        is AuthEvent.ChangePIN -> when (state.screenStatus) {
            AuthState.AuthScreenStatus.LOG_IN -> state.copy(loginState =
loginState.copy(pin = event.newPIN))
            AuthState.AuthScreenStatus.SET_PIN -> state.copy(setPINState
= setPINState.copy(pin = event.newPIN))
            AuthState.AuthScreenStatus.CONFIRM_PIN ->
state.copy(confirmPINState = confirmPINState.copy(pin = event.newPIN))
        }
        is AuthEvent.ExecuteKeyboardCommand -> state
        else -> throw IllegalArgumentException("Unsupported Event:
$event")
    }
}
}
}

```

```

package com.zhiroke.presentation.auth.presentation.auth

import androidx.activity.compose.BackHandler
import androidx.compose.foundation.layout.*
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.lifecycle.compose.ExperimentalLifecycleComposeApi
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.zhiroke.core.components.toasts.ErrorToast
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_32
import com.zhiroke.core.theme.dimensions.dp_64
import com.zhiroke.core.theme.utils.MaterialTypography
import com.zhiroke.presentation.auth.R
import
com.zhiroke.presentation.auth.presentation.auth.components.buttons.TertiaryTextB
utton
import
com.zhiroke.presentation.auth.presentation.auth.components.dialkeyboard.DialKeyb
oardBlock
import
com.zhiroke.presentation.auth.presentation.auth.components.overlays.LockedScreen
Overlay
import
com.zhiroke.presentation.auth.presentation.auth.components.passwordfield.PINFiel
d
import
com.zhiroke.presentation.auth.presentation.auth.components.popups.DropDataPopUp
import org.koin.androidx.compose.getViewModel

@Composable
fun AuthRoute() {

    AuthScreen()
}

@OptIn(ExperimentalLifecycleComposeApi::class)

```

```

@Composable
private fun AuthScreen(viewModel: AuthViewModel = getViewModel()) {

    val state by viewModel.state.collectAsStateWithLifecycle()

    Box(modifier = Modifier.fillMaxSize()) {

        when (state.screenStatus) {
            AuthState.AuthScreenStatus.LOG_IN -> LoginBlock(viewModel =
viewModel, state = state.loginState)
            AuthState.AuthScreenStatus.SET_PIN -> SetPINBlock(viewModel =
viewModel, state = state.setPINState)
            AuthState.AuthScreenStatus.CONFIRM_PIN -> ConfirmPINBlock(viewModel
= viewModel, state = state.confirmPINState)
        }
    }

    ErrorToast(text = state.errorMessage)
}

@Composable
private fun BoxScope.LoginBlock(viewModel: AuthViewModel, state:
AuthState.LoginState) {

    Column(
        modifier = Modifier
            .wrapContentSize()
            .padding(bottom = dp_32)
            .align(Alignment.BottomCenter),
        verticalArrangement = Arrangement.spacedBy(space = dp_32, alignment =
Alignment.Bottom)
    ) {

        Text(
            modifier = Modifier.align(alignment = Alignment.CenterHorizontally),
            text = stringResource(R.string.enter_pin_code)
        )

        PINField(
            modifier = Modifier.align(Alignment.CenterHorizontally),
            pin = state.pin,
            visible = false
        )
    }
}

```

```

    )

    DialKeyboardBlock(
        modifier = Modifier.align(Alignment.CenterHorizontally),
        biometricAuthSupported = state.isBioSupported,
        onCommand = viewModel::executeCommand,
        onBioClick = viewModel::authWithBio,
    )

    TextButton(onClick = viewModel::showDropDataPopUp) {

        Text(text = stringResource(R.string.dont_remember_pin))
    }
}

DropDataPopUp(popUpState = state.dropDataPopUpState, onHide =
viewModel::hideDropDataPopUp, onDropData = viewModel::dropAllData)

if (state.secondsLeftToBeUnlocked != null) {
    LockedScreenOverlay(text = stringResource(id =
R.string.placeholder_try_again, state.secondsLeftToBeUnlocked))
}
}

@Composable
private fun BoxScope.SetPINBlock(viewModel: AuthViewModel, state:
AuthState.SetPINState) {

    TertiaryTextButton(
        modifier = Modifier
            .height(height = dp_64)
            .align(Alignment.TopEnd)
            .padding(top = dp_16, end = dp_16),
        resId = R.string.btn_next,
        fontSize = MaterialTypography.labelLarge.fontSize,
        enabled = state.pin.length == 4,
        onClick = viewModel::changeScreenStatusToConfirmPIN
    )

    Column(
        modifier = Modifier
            .wrapContentSize()

```

```

        .padding(bottom = dp_32)
        .align(Alignment.BottomCenter),
        verticalArrangement = Arrangement.spacedBy(space = dp_32, alignment =
Alignment.Bottom)
    ) {

        Text(
            modifier = Modifier.align(alignment = Alignment.CenterHorizontally),
            text = stringResource(R.string.set_pin_code)
        )

        PINField(
            modifier = Modifier.align(Alignment.CenterHorizontally),
            pin = state.pin,
            visible = true
        )

        DialKeyboardBlock(
            modifier = Modifier.align(Alignment.CenterHorizontally),
            biometricAuthSupported = false,
            onCommand = viewModel::executeCommand,
            onBioClick = viewModel::authWithBio,
        )
    }
}

@Composable
private fun BoxScope.ConfirmPINBlock(viewModel: AuthViewModel, state:
AuthState.ConfirmPINState) {

    BackHandler(onBack = viewModel::changeScreenStatusToSetPIN)

    TertiaryTextButton(
        modifier = Modifier
            .align(Alignment.TopStart)
            .padding(top = dp_16, start = dp_16),
        resId = R.string.btn_back,
        fontSize = MaterialTypography.labelLarge.fontSize,
        enabled = true,
        onClick = viewModel::changeScreenStatusToSetPIN
    )
}

```

```

TertiaryTextButton(
    modifier = Modifier
        .height(height = dp_64)
        .align(Alignment.TopEnd)
        .padding(top = dp_16, end = dp_16),
    resId = R.string.btn_confirm,
    fontSize = MaterialTypography.labelLarge.fontSize,
    enabled = state.pin.length == 4,
    onClick = viewModel::confirmPIN
)

Column(
    modifier = Modifier
        .wrapContentSize()
        .padding(bottom = dp_32)
        .align(Alignment.BottomCenter),
    verticalArrangement = Arrangement.spacedBy(space = dp_32, alignment =
Alignment.Bottom)
) {

    Text(
        modifier = Modifier.align(alignment = Alignment.CenterHorizontally),
        text = stringResource(R.string.confirm_pin_code)
    )

    PINField(
        modifier = Modifier.align(Alignment.CenterHorizontally),
        pin = state.pin,
        visible = true
    )

    DialKeyboardBlock(
        modifier = Modifier.align(Alignment.CenterHorizontally),
        biometricAuthSupported = false,
        onCommand = viewModel::executeCommand,
        onBioClick = viewModel::authWithBio,
    )
}
}

```

```

package com.zhiroke.presentation.auth.presentation.auth

import com.zhiroke.core.common.base.BaseState
import com.zhiroke.core.common.base.error.ErrorState
import com.zhiroke.core.components.popup.PopUpState

internal data class AuthState(
    val isLoading: Boolean,
    val loginState: LoginState,
    val setPINState: SetPINState,
    val confirmPINState: ConfirmPINState,
    val screenStatus: AuthScreenStatus,
    override val errorMessage: String?,
) : BaseState, ErrorState {

    internal data class LoginState(
        val attempt: Int,
        val unlockAttemptTime: Long,
        val secondsLeftToBeUnlocked: Int?,
        val loadedPIN: String?,
        val dropDataPopUpState: PopUpState,
        val pin: String,
        val isBioSupported: Boolean,
    ) {

        val isPINUnlocked get() = unlockAttemptTime <=
System.currentTimeMillis()

        companion object {

            fun initialState() = LoginState(
                attempt = 0,
                unlockAttemptTime = 0,
                secondsLeftToBeUnlocked = null,
                pin = "",
                loadedPIN = null,
                dropDataPopUpState = PopUpState(isShown = false),
                isBioSupported = false
            )

            const val ATTEMPTS_LIMIT = 5

```

```

        const val LOCK_DELAY = 30 * 1000 // 30 seconds
    }
}

internal data class SetPINState(
    val pin: String
) {

    companion object {

        fun initialState() = SetPINState(
            pin = ""
        )
    }
}

internal data class ConfirmPINState(
    val pin: String
) {

    companion object {

        fun initialState() = ConfirmPINState(
            pin = "",
        )
    }
}

internal enum class AuthScreenStatus {
    LOG_IN, SET_PIN, CONFIRM_PIN
}

companion object {

    fun initialState() = AuthState(
        isLoading = true,
        loginState = LoginState.initialState(),
        setPINState = SetPINState.initialState(),
        confirmPINState = ConfirmPINState.initialState(),
        screenStatus = AuthScreenStatus.LOG_IN,
        errorMessage = null,
    )
}

```

```

    }
}

package com.zhiroke.presentation.auth.presentation.auth

import androidx.annotation.RestrictTo
import com.zhiroke.core.common.base.BaseViewModel
import com.zhiroke.core.common.coroutines.AppDispatchersProvider
import com.zhiroke.core.common.coroutines.DispatchersProvider
import com.zhiroke.presentation.auth.presentation.auth.interactors.*

internal class AuthViewModel(
    reducer: AuthReducer,
    dispatchers: DispatchersProvider,
    keyboardHandlerInteractor: KeyboardHandlerInteractor?,
    biometricAuthInteractor: BiometricAuthInteractor?,
    checkForBioAuthSupportInteractor: CheckForBioAuthSupportInteractor?,
    succeededToAuthInteractor: SucceededToAuthInteractor?,
    confirmPINInteractor: ConfirmPINInteractor?,
    getPINInteractor: GetPINInteractor?,
    pinAuthInteractor: PINAuthInteractor?,
    authErrorStopperInteractor: AuthErrorStopperInteractor?,
    getUnlockPINTimeInteractor: GetUnlockPINTimeInteractor?,
    handlePINAuthFailureInteractor: HandlePINAuthFailureInteractor?,
    handleFailedToConfirmPINInteractor: HandleFailedToConfirmPINInteractor?,
    dropAllDataInteractor: DropAllDataInteractor?
) : BaseViewModel<AuthState, AuthEvent>(
    dispatchers = dispatchers,
    reducer = reducer,
    interactors = setOfNotNull(keyboardHandlerInteractor,
    biometricAuthInteractor,
        checkForBioAuthSupportInteractor, succeededToAuthInteractor,
    confirmPINInteractor,
        getPINInteractor, pinAuthInteractor, authErrorStopperInteractor,
    getUnlockPINTimeInteractor,
        handlePINAuthFailureInteractor, handleFailedToConfirmPINInteractor,
    dropAllDataInteractor
    ),
    initialEvent = AuthEvent.Load
) {

```

```

fun executeCommand(command: PINKeyboardCommand) =
    sendEvent(AuthEvent.ExecuteKeyboardCommand(command = command))

fun authWithBio(command: PINKeyboardCommand) =
    sendEvent(AuthEvent.AuthWithBio)

fun changeScreenStatusToSetPIN() =
    sendEvent(AuthEvent.ChangeScreenStatus(screenStatus =
    AuthState.AuthScreenStatus.SET_PIN))

fun changeScreenStatusToConfirmPIN() =
    sendEvent(AuthEvent.ChangeScreenStatus(screenStatus =
    AuthState.AuthScreenStatus.CONFIRM_PIN))

fun confirmPIN() = sendEvent(AuthEvent.ConfirmPIN)

fun hideDropDataPopUp() =
    sendEvent(AuthEvent.ChangeDropDataPopUpState(state.value.loginState.dropDataPopU
    pState.copy(isShown = false)))

fun showDropDataPopUp() =
    sendEvent(AuthEvent.ChangeDropDataPopUpState(state.value.loginState.dropDataPopU
    pState.copy(isShown = true)))

fun dropAllData() = sendEvent(AuthEvent.DropAllData)
}

/** This function is used only for compose preview. */
@RestrictTo(RestrictTo.Scope.TESTS)
internal fun getDummyAuthViewModel(): AuthViewModel {
    return AuthViewModel(
        dispatchers = AppDispatchersProvider(),
        reducer = AuthReducer(),
        keyboardHandlerInteractor = null,
        biometricAuthInteractor = null,
        checkForBioAuthSupportInteractor = null,
        succeededToAuthInteractor = null,
        confirmPINInteractor = null,
        getPINInteractor = null,
        pinAuthInteractor = null,
        authErrorStopperInteractor = null,

```

```

        getUnlockPINTimeInteractor = null,
        handlePINAuthFailureInteractor = null,
        handleFailedToConfirmPINInteractor = null,
        dropAllDataInteractor = null
    )
}

package com.zhiroke.presentation.auth.utils.managers

import android.os.Build
import androidx.biometric.BiometricManager
import androidx.biometric.BiometricManager.Authenticators.BIOMETRIC_STRONG
import androidx.biometric.BiometricPrompt
import androidx.core.content.ContextCompat
import androidx.fragment.app.FragmentActivity
import com.zhiroke.core.common.utils.vibrate
import com.zhiroke.presentation.auth.R
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import org.koin.core.context.GlobalContext
import kotlin.coroutines.suspendCoroutine

class BiometricAuthManager {

    private var activity: FragmentActivity? = null

    fun attach(activity: FragmentActivity) {
        this.activity = activity
    }

    fun detach() {
        activity = null
    }

    suspend fun auth(): Int? = suspendCoroutine { continuation ->
        activity?.let { activity ->
            val promptInfo = BiometricPrompt.PromptInfo.Builder()
                .setTitle(activity.getString(R.string.biometric_title))

                .setNegativeButtonText(activity.getString(R.string.biometric_negative_btn))

```

```

        .setAllowedAuthenticators(BIOMETRIC_STRONG)
        .build()

    val executor = ContextCompat.getMainExecutor(activity)

    val biometricPrompt = BiometricPrompt(
        activity,
        executor,
        object : BiometricPrompt.AuthenticationCallback() {

            override fun onAuthenticationError(errorCode: Int,
errString: CharSequence) {
                super.onAuthenticationError(errorCode, errString)
                continuation.resumeWith(Result.success(errorCode))
            }

            override fun onAuthenticationSucceeded(result:
BiometricPrompt.AuthenticationResult) {
                super.onAuthenticationSucceeded(result)
                continuation.resumeWith(Result.success(null))
            }

            override fun onAuthenticationFailed() {
                super.onAuthenticationFailed()
                activity.vibrate()
            }

        }
    )

    CoroutineScope(Dispatchers.Main).launch {
        biometricPrompt.authenticate(promptInfo)
    }
}

fun isBioAuthSupported(): Boolean {
    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.R) return false
    return activity?.let { activity ->
        when
(BiometricManager.from(activity).canAuthenticate(BIOMETRIC_STRONG)) {
            BiometricManager.BIOMETRIC_SUCCESS -> true
            else -> false
        }
    }
}

```

```

        } ?: false
    }
}

fun attachBiometricAuthManager(activity: FragmentActivity) {
    getBiometricAuthManager().attach(activity = activity)
}

fun detachBiometricAuthManager() {
    getBiometricAuthManager().detach()
}

private fun getBiometricAuthManager(): BiometricAuthManager {
    return GlobalContext.get().get<BiometricAuthManager>()
}

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

</manifest>

package com.zhiroke.presentation.mywallet.di.modules

import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors.AddCardI
nteractor
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors.DeleteCa
rdInteractor
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors.LoadCard
sInteractor
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors.SaveCard
Interactor
import org.koin.core.module.dsl.factoryOf
import org.koin.dsl.module

internal fun interactorsModule() = module {

```

```

        factoryOf(::LoadCardsInteractor)
        factoryOf(::AddCardInteractor)
        factoryOf(::SaveCardInteractor)
        factoryOf(::DeleteCardInteractor)
    }

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.button

import androidx.compose.foundation.layout.BoxScope
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.rounded.Add
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Icon
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import com.zhiroke.core.theme.dimensions.dp_0
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_24
import com.zhiroke.core.theme.dimensions.dp_96
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes

@Composable
internal fun BoxScope.AddCardButton(onClick: () -> Unit) {

    val buttonColors = ButtonDefaults.buttonColors(
        containerColor = MaterialColor.secondary,
        contentColor = MaterialColor.onSecondary
    )

    Button(
        modifier = Modifier

```

```

        .size(dp_96)
        .align(Alignment.BottomEnd)
        .padding(dp_16),
        colors = buttonColors,
        shape = MaterialShapes.small,
        onClick = onClick,
        contentPadding = PaddingValues(dp_0)
    ) {

        Icon(
            modifier = Modifier.size(dp_24),
            imageVector = Icons.Rounded.Add,
            contentDescription = null
        )
    }
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.components

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.zhiroke.core.theme.color.Gray
import com.zhiroke.core.theme.dimensions.dp_24

@Composable
internal fun AuthorizedSignatureStripe(modifier: Modifier = Modifier) {

    Spacer(
        modifier = modifier
            .fillMaxWidth()
            .height(dp_24)
            .background(Gray)
    )
}

```

```
package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.components
```

```
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.zhiroke.core.theme.color.Black
import com.zhiroke.core.theme.dimensions.dp_48
```

```
@Composable
internal fun MagneticStripe(modifier: Modifier = Modifier) {

    Spacer(
        modifier = modifier
            .fillMaxWidth()
            .height(dp_48)
            .background(Black)
    )
}
```

```
package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.components
```

```
import androidx.compose.foundation.layout.Row
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
```

```
@Composable
internal fun SignatureAndVerificationNumberRow(
    modifier: Modifier = Modifier,
    verificationNumberContent: @Composable () -> Unit
) {
```

```

        Row(modifier = modifier, verticalAlignment = Alignment.CenterVertically) {

            AuthorizedSignatureStripe(modifier = modifier.weight(weight = 1f))

            VerificationNumberContainer(content = verificationNumberContent)
        }
    }

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.components

import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.comm
on.CardTextContainer

@Composable
internal fun VerificationNumberContainer(modifier: Modifier = Modifier, content:
@Composable () -> Unit) {

    CardTextContainer(modifier = modifier, content = content)
}

@Composable
internal fun VerificationNumberText(modifier: Modifier = Modifier, number:
String) {

    Text(modifier = modifier, text = number)
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.fillMaxSize

```

```

import androidx.compose.foundation.layout.padding
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.graphicsLayer
import androidx.compose.ui.tooling.preview.Preview
import com.zhiroke.core.components.card.EuropeanCardRatioContainer
import com.zhiroke.core.theme.dimensions.dp_4
import com.zhiroke.core.theme.dimensions.dp_8
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes
import com.zhiroke.domain.models.BankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.components.VerificationNumberText

/**
 * Immutable - because we use Text() for content
 */
@Composable
internal fun ImmutableBackSideCard(bankCard: BankCard) {

    EuropeanCardRatioContainer(modifier = Modifier.clip(shape =
MaterialShapes.medium)) {

        StatelessBackSideCard(
            modifier = Modifier
                .fillMaxSize()
                .background(color = MaterialColor.primaryContainer)
                .graphicsLayer(rotationY = 180f),
            verificationNumberContent = {

                VerificationNumberText(
                    modifier = Modifier.padding(vertical = dp_4, horizontal =
dp_8),
                    number = bankCard.verificationNumber
                )
            }
        )
    }
}

```

```

@Preview(showBackground = true)
@Composable
private fun ImmutableBackSideCardPreview() {

    ImmutableBackSideCard(
        bankCard = BankCard.dummy()
    )
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.graphicsLayer
import androidx.compose.ui.tooling.preview.Preview
import com.zhiroke.core.components.animations.ShimmeringTextSize
import com.zhiroke.core.components.card.EuropeanCardRatioContainer
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.components.VerificationNumberContainer

/**
 * For loading state
 */
@Composable
internal fun ShimmeringBackSideCard() {

    EuropeanCardRatioContainer(modifier = Modifier.clip(shape =
MaterialShapes.medium)) {

        StatelessBackSideCard(
            modifier = Modifier
                .fillMaxSize()

```

```

        .background(color = MaterialColor.primaryContainer)
        .graphicsLayer(rotationY = 180f),
verificationNumberContent = {

    VerificationNumberContainer {

        ShimmeringTextSize(text = "123")
    }
}

)

}

}

@Preview
@Composable
private fun ShimmeringBackSideCardPreview() {

    ShimmeringBackSideCard()
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back

import androidx.compose.foundation.layout.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.zhiroke.core.theme.dimensions.dp_8
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.components.MagneticStripe
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.components.SignatureAndVerificationNumberRow

@Composable
internal fun StatelessBackSideCard(
    modifier: Modifier = Modifier,
    verificationNumberContent: @Composable () -> Unit
) {

```

```

Box(modifier = modifier) {

    Column(
        modifier = Modifier.fillMaxHeight(fraction = 0.6f),
        verticalArrangement = Arrangement.SpaceAround
    ) {

        MagneticStripe()

        SignatureAndVerificationNumberRow(
            modifier = Modifier
                .fillMaxWidth()
                .wrapContentHeight()
                .padding(horizontal = dp_8),
            verificationNumberContent = verificationNumberContent
        )
    }
}

```

```

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.common

```

```

import androidx.compose.material3.Surface
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes

```

```

@Composable
fun CardTextContainer(modifier: Modifier = Modifier, content: @Composable () ->
Unit) {

```

```

    Surface(
        modifier = modifier.clip(shape = MaterialShapes.small),

```

```

        color = MaterialColor.primary,
        content = content
    )
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components

import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.zhiroke.core.theme.utils.MaterialTypography
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.comm
on.CardTextContainer

@Composable
internal fun CardholderContainer(modifier: Modifier = Modifier, content:
@Composable () -> Unit) {

    CardTextContainer(modifier = modifier, content = content)
}

@Composable
internal fun CardholderText(modifier: Modifier = Modifier, cardholder: String) {

    Text(
        modifier = modifier,
        text = cardholder,
        style = MaterialTypography.titleMedium
    )
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components

import androidx.compose.material3.Text

```

```
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.common.CardTextContainer
```

```
@Composable
internal fun CardNumberContainer(modifier: Modifier = Modifier, content:
@Composable () -> Unit) {
```

```
    CardTextContainer(modifier = modifier, content = content)
}
```

```
@Composable
internal fun CardNumberText(modifier: Modifier = Modifier, number: String) {
```

```
    Text(
        modifier = modifier,
        text = number.toCardNumberFormat(),
    )
}
```

```
private fun String.toCardNumberFormat(): String {
    val multiplier = 4
    return this.mapIndexed { index, char ->
        if (index % multiplier == 0 && index != 0) " $char" else char
    }.joinToString("")
}
```

```
package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.front.components
```

```
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.zhiroke.core.theme.utils.MaterialTypography
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.common.CardTextContainer
```

```

@Composable
internal fun ExpirationDateContainer(modifier: Modifier = Modifier, content:
@Composable () -> Unit) {

    CardTextContainer(modifier = modifier, content = content)
}

@Composable
internal fun ExpirationDateText(modifier: Modifier = Modifier, expirationDate:
String) {

    Text(
        modifier = modifier,
        text = expirationDate.toExpirationDateFormat(),
        style = MaterialTypography.titleMedium
    )
}

private fun String.toExpirationDateFormat(): String {
    return this.mapIndexed { index, char ->
        if (index == 2) "/$char" else char
    }.joinToString("")
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier

@Composable
internal fun SkinContainer(modifier: Modifier = Modifier, content: @Composable
() -> Unit) {

    Box(modifier = modifier.fillMaxSize()) {

```

```

        content.invoke()
    }
}

```

```

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components

```

```

import androidx.compose.foundation.layout.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import com.zhiroke.core.theme.demensions.dp_16

```

```

@Composable
internal fun BoxScope.ToolsContainer(modifier: Modifier = Modifier,
toolsContent: @Composable (RowScope.() -> Unit)? = null) {

```

```

    Row(
        modifier = modifier
            .wrapContentSize()
            .align(alignment = Alignment.TopEnd)
            .padding(top = dp_16, end = dp_16),
    ) {

        toolsContent?.invoke(this@Row)
    }
}

```

```

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t

```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable

```

```

import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.style.TextAlign
import com.zhiroke.core.components.card.EuropeanCardRatioContainer
import com.zhiroke.core.theme.dimensions.dp_4
import com.zhiroke.core.theme.dimensions.dp_48
import com.zhiroke.core.theme.dimensions.dp_8
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes
import com.zhiroke.presentation.mywallet.R
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.common.CardTextContainer

```

```
@Composable
```

```
internal fun EmptyBankCard(modifier: Modifier = Modifier) {
```

```
    EuropeanCardRatioContainer(modifier = modifier) {
```

```
        Box(
```

```
            modifier = Modifier
```

```
                .fillMaxSize()
```

```
                .clip(shape = MaterialShapes.medium)
```

```
                .background(color = MaterialColor.primaryContainer),
```

```
            contentAlignment = Alignment.Center
```

```
        ) {
```

```
            CardTextContainer(
```

```
                modifier = Modifier
```

```
                    .clip(shape = MaterialShapes.medium)
```

```
                    .padding(horizontal = dp_48)
```

```
            ) {
```

```
                Text(
```

```
                    modifier = Modifier.padding(vertical = dp_4, horizontal =
```

```
dp_8),
```

```
                    text = stringResource(R.string.empty_card_text),
```

```
                    textAlign = TextAlign.Center
```

```
                )
```

```

    }
}
}
}

```

```

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t

```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.RowScope
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.tooling.preview.Preview
import coil.compose.AsyncImage
import com.zhiroke.core.components.card.EuropeanCardRatioContainer
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_4
import com.zhiroke.core.theme.dimensions.dp_8
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes
import com.zhiroke.domain.models.BankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components.CardNumberText
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components.CardholderText
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components.ExpirationDateText
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components.ToolsContainer

```

```

/**

```

```

    * Immutable - because we use Text() for content
    */
@Composable
internal fun ImmutableFrontSideCard(
    bankCard: BankCard,
    toolsContent: @Composable (RowScope.() -> Unit)? = null
) {

    EuropeanCardRatioContainer(modifier = Modifier.clip(shape =
MaterialShapes.medium)) {

        StatelessFrontSideCard(
            modifier = Modifier
                .fillMaxSize()
                .background(color = MaterialColor.primaryContainer)
                .padding(all = dp_16),
            cardNumberContent = {

                CardNumberText(
                    modifier = Modifier.padding(vertical = dp_4, horizontal =
dp_8),

                    number = bankCard.number
                )
            },
            expirationDateContent = {

                ExpirationDateText(
                    modifier = Modifier.padding(vertical = dp_4, horizontal =
dp_8),

                    expirationDate = bankCard.expirationDate
                )
            },
            cardholderContent = {

                CardholderText(
                    modifier = Modifier.padding(vertical = dp_4, horizontal =
dp_8),

                    cardholder = bankCard.cardholderName
                )
            },
            skinContent = {

```

```

        AsyncImage(
            modifier =
Modifier.fillMaxSize().clip(MaterialShapes.medium),
            model = bankCard.skinUri,
            contentDescription = null,
            contentScale = ContentScale.Crop,
        )
    }
)

ToolsContainer(toolsContent = toolsContent)
}
}

```

```

@Preview(showBackground = true)
@Composable
private fun FrontSideCardPreview() {

    ImmutableFrontSideCard(
        bankCard = BankCard.dummy()
    )
}

```

```

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t

```

```

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.tooling.preview.Preview
import com.zhiroke.core.components.animations.ShimmeringTextSize
import com.zhiroke.core.components.card.EuropeanCardRatioContainer
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.core.theme.utils.MaterialShapes

```

```

/**
 * For loading state
 */
@Composable
internal fun ShimmeringFrontSideCard() {

    EuropeanCardRatioContainer(modifier = Modifier.clip(shape =
MaterialShapes.medium)) {

        StatelessFrontSideCard(
            modifier = Modifier
                .fillMaxSize()
                .background(color = MaterialColor.primaryContainer)
                .padding(all = dp_16),
            cardNumberContent = {

                ShimmeringTextSize(text = "1234567812345678")
            },
            expirationDateContent = {

                ShimmeringTextSize(text = "11/99")
            },
            cardholderContent = {

                ShimmeringTextSize(text = "Egor Zhir")
            }
        )
    }
}

@Preview
@Composable
private fun ShimmeringFrontSideCardPreview() {

    ShimmeringFrontSideCard()
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t

```

```

import androidx.compose.foundation.layout.Box
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components.CardNumberContainer
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components.CardholderContainer
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components.ExpirationDateContainer
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.components.SkinContainer

```

```
@Composable
```

```

internal fun StatelessFrontSideCard(
    modifier: Modifier = Modifier,
    cardNumberContent: @Composable () -> Unit,
    expirationDateContent: @Composable () -> Unit,
    cardholderContent: @Composable () -> Unit,
    skinContent: (@Composable () -> Unit)? = null
) {

    Box(modifier = modifier) {

        skinContent?.let { content ->
            SkinContainer(content = content)
        }

        CardNumberContainer(
            modifier = Modifier.align(alignment = Alignment.Center),
            content = cardNumberContent
        )

        ExpirationDateContainer(
            modifier = Modifier.align(alignment = Alignment.BottomStart),
            content = expirationDateContent

```

```

    )

    CardholderContainer(
        modifier = Modifier.align(alignment = Alignment.BottomEnd),
        content = cardholderContent
    )
}
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card

import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.wrapContentSize
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.rounded.Delete
import androidx.compose.material.icons.rounded.Edit
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.zhiroke.core.common.utils.rememberLambda
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_24
import com.zhiroke.core.theme.dimensions.dp_32
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.domain.models.BankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.ImmutableBackSideCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.ImmutableFrontSideCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.cardwrapp
er.RotatingCardWrapper

@Composable
internal fun BankCard(

```

```

card: BankCard, onCopy: (String) -> Unit,
onEditClick: (BankCard) -> Unit,
onDeleteClick: (BankCard) -> Unit
) {

RotatingCardWrapper(
    modifier = Modifier
        .wrapContentSize()
        .padding(horizontal = dp_16, vertical = dp_32),
    onLongPress = rememberLambda { isFrontSide ->
        if (isFrontSide) {
            onCopy.invoke(card.number)
        }
    }
) { isFrontSide ->

    if (isFrontSide) {
        ImmutableFrontSideCard(
            bankCard = card,
            toolsContent = {

                IconButton(onClick = { onEditClick.invoke(card) }) {
                    Icon(
                        modifier = Modifier.size(dp_24),
                        imageVector = Icons.Rounded.Edit,
                        contentDescription = null,
                        tint = MaterialColor.primary
                    )
                }

                IconButton(onClick = { onDeleteClick.invoke(card) }) {
                    Icon(
                        modifier = Modifier.size(dp_24),
                        imageVector = Icons.Rounded.Delete,
                        contentDescription = null,
                        tint = MaterialColor.primary
                    )
                }
            }
    } else {
        ImmutableBackSideCard(bankCard = card)
    }
}

```

```

    }
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card

import android.content.Intent
import android.net.Uri
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.PickVisualMediaRequest
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.layout.size
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.rounded.AddPhotoAlternate
import androidx.compose.material.icons.rounded.Close
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import com.zhiroke.core.theme.dimensions.dp_24
import com.zhiroke.core.theme.utils.MaterialColor
import com.zhiroke.domain.models.BankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.ImmutableBackSideCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.ImmutableFrontSideCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.cardwrapp
er.RotatingCardWrapper

@Composable
internal fun EditableBankCard(bankCard: BankCard, isFrontSide: Boolean,
onPhotoUriChange: (Uri?) -> Unit) {

    val context = LocalContext.current
    val singlePhotoPickerLauncher = rememberLauncherForActivityResult(

```

```

        contract = ActivityResultContracts.PickVisualMedia() ,
        onResult = { uri ->
            uri?.let {
                val flag = Intent.FLAG_GRANT_READ_URI_PERMISSION
                context.contentResolver.takePersistableUriPermission(it, flag)
            }
            onPhotoUriChange.invoke(uri)
        }
    )

    RotatingCardWrapper(initialIsFrontSide = isFrontSide) { _isFrontSide ->

        if (_isFrontSide) {
            ImmutableFrontSideCard(bankCard = bankCard) {

                if (bankCard.skinUri == null) {
                    PickSkinButton {

singlePhotoPickerLauncher.launch(PickVisualMediaRequest(ActivityResultContracts.
PickVisualMedia.ImageOnly))

                    }
                } else {
                    DeleteSkinButton {
                        onPhotoUriChange.invoke(null)
                    }
                }
            }
        } else {
            ImmutableBackSideCard(bankCard = bankCard)
        }
    }
}

@Composable
private fun PickSkinButton(modifier: Modifier = Modifier, onClick: () -> Unit) {

    IconButton(modifier = modifier, onClick = onClick) {

        Icon(
            modifier = Modifier.size(dp_24),
            imageVector = Icons.Rounded.AddPhotoAlternate,
            contentDescription = null,

```

```

        tint = MaterialColor.primary
    )
}

@Composable
private fun DeleteSkinButton(modifier: Modifier = Modifier, onClick: () -> Unit)
{
    IconButton(modifier = modifier, onClick = onClick) {
        Icon(
            modifier = Modifier.size(dp_24),
            imageVector = Icons.Rounded.Close,
            contentDescription = null,
            tint = MaterialColor.primary
        )
    }
}

```

```

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card

import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.wrapContentSize
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_32
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.back
.ShimmeringBackSideCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.fron
t.ShimmeringFrontSideCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.cardwrapp
er.RotatingCardWrapper

@Composable

```

```

internal fun ShimmeringBankCard() {

    RotatingCardWrapper(
        modifier = Modifier
            .wrapContentSize()
            .padding(horizontal = dp_16, vertical = dp_32),
    ) { isFrontSide ->

        if (isFrontSide) {
            ShimmeringFrontSideCard()
        } else {
            ShimmeringBackSideCard()
        }
    }
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.cardwrapp
er

import androidx.compose.animation.core.animateFloatAsState
import androidx.compose.animation.core.tween
import androidx.compose.runtime.*
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.ui.Modifier
import androidx.compose.ui.geometry.Offset
import com.zhiroke.core.components.cardwrapper.StatelessCardWrapper

/**
 * Stateful rotating card
 */
@Composable
fun RotatingCardWrapper(
    modifier: Modifier = Modifier,
    initialIsFrontSide: Boolean = true,
    onLongPress: ((isFront: Boolean) -> Unit)? = null,
    cardContent: @Composable (isFrontSide: Boolean) -> Unit
) {
    var isFrontSide by rememberSaveable(initialIsFrontSide) {
        mutableStateOf(initialIsFrontSide) }

```

```

val rotateX by remember { mutableStateOf(0f) }
val rotateY by remember { mutableStateOf(0f) }

val onDoubleTap: (Offset) -> Unit = {
    isFrontSide = !isFrontSide
}

rotateY = animateFloatAsState(
    targetValue = if (isFrontSide) 0f else 180f,
    animationSpec = tween(durationMillis = 1500),
).value

StatelessCardWrapper(
    modifier = modifier,
    rotateX = rotateX,
    rotateY = rotateY,
    onLongPress = {
        onLongPress?.invoke(isFrontSide)
    },
    onDoubleTap = onDoubleTap,
    cardContent = {
        cardContent.invoke(isFrontSide = rotateY <= 90)
    }
)
}

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups.co
mponents

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import com.zhiroke.core.components.card.EuropeanCardRatioContainer
import com.zhiroke.core.theme.utils.MaterialShapes
import com.zhiroke.core.cardrecognition.composable.CardRecognizer

```

```

@Composable
fun CardRecognizerPage(
    modifier: Modifier = Modifier,
    forceToLeaveComposition: Boolean,
    onReturnToPreviousPage: () -> Unit,
    onFrontSideCardRecognized: (cardNumber: String, expirationDate: String) ->
Unit,
    onBackSideCardRecognized: (verificationNumber: String) -> Unit
) {
    Box(
        modifier = modifier,
        contentAlignment = Alignment.Center
    ) {
        EuropeanCardRatioContainer(
            modifier = Modifier
                .clip(shape = MaterialShapes.medium)
                .background(color = Color.Black)
        ) {

            /* Current page doesn't leave composition, after changing paging to
the previous one,
            * so we have to force to leave, to call onDispose.
            */
            if (!forceToLeaveComposition) {
                CardRecognizer(
                    modifier = Modifier.fillMaxSize(),
                    onDismissRequest = onReturnToPreviousPage,
                    onFrontSideCardRecognized = onFrontSideCardRecognized,
                    onBackSideCardRecognized = onBackSideCardRecognized
                )
            }
        }
    }
}

package
com.zhroke.presentation.mywallet.presentation.cardcarousel.components.popups.co
mponents

```

```

import android.net.Uri
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import androidx.compose.ui.focus.FocusDirection
import androidx.compose.ui.focus.onFocusChanged
import androidx.compose.ui.platform.LocalFocusManager
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.text.input.KeyboardCapitalization
import androidx.compose.ui.text.input.KeyboardType
import com.zhiroke.core.components.buttons.ButtonText
import com.zhiroke.core.components.column.ScrollableColumn
import com.zhiroke.core.theme.demensions.dp_16
import com.zhiroke.domain.models.BankCard
import com.zhiroke.presentation.mywallet.R
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.Edit
ableBankCard
import com.zhiroke.presentation.mywallet.utils.*

internal class EditBankCardCallbacks(
    val onNumberChange: (String) -> Unit,
    val onExpirationDateChange: (String) -> Unit,
    val onCardholderChange: (String) -> Unit,
    val onVerificationNumberChange: (String) -> Unit,
    val onPhotoUriChange: (Uri?) -> Unit,
)

@OptIn(ExperimentalMaterial3Api::class)
@Composable
internal fun EditableCardPage(
    modifier: Modifier = Modifier,
    bankCard: BankCard,
    editBankCardCallbacks: EditBankCardCallbacks,
    onDone: () -> Unit,

```

```

) {

    var isFrontSide by remember { mutableStateOf(true) }
    val localFocusManager = LocalFocusManager.current

    ScrollableColumn(
        modifier = modifier,
        verticalArrangement = Arrangement.SpaceBetween
    ) {

        Column {

            EditableBankCard(bankCard = bankCard, isFrontSide = isFrontSide,
onPhotoUriChange = editBankCardCallbacks.onPhotoUriChange)

            Spacer(modifier = Modifier.height(dp_16))

            OutlinedTextField(
                modifier = Modifier
                    .fillMaxWidth()
                    .onFocusChanged {
                        if (it.isFocused) isFrontSide = true
                    },
                value = bankCard.number,
                onValueChange = editBankCardCallbacks.onNumberChange,
                label = {
                    Text(text = stringResource(id = R.string.card_number_label))
                },
                singleLine = true,
                keyboardOptions = KeyboardOptions(
                    keyboardType = KeyboardType.Number,
                    imeAction = ImeAction.Next
                ),
                keyboardActions = KeyboardActions(onNext = {
                    localFocusManager.moveFocus(FocusDirection.Down)
                }),
                isError = !bankCard.number.isNumberValid()
            )

            OutlinedTextField(
                modifier = Modifier
                    .fillMaxWidth()

```

```

        .onFocusChanged {
            if (it.isFocused) isFrontSide = true
        },
        value = bankCard.expirationDate,
        onValueChange = editBankCardCallbacks.onExpirationDateChange,
        label = {
            Text(text = stringResource(id =
R.string.expiration_date_label))
        },
        singleLine = true,
        keyboardOptions = KeyboardOptions(
            keyboardType = KeyboardType.Number,
            imeAction = ImeAction.Next
        ),
        keyboardActions = KeyboardActions(onNext = {
            localFocusManager.moveFocus(FocusDirection.Down)
        }),
        isError = !bankCard.expirationDate.isExpirationDateValid()
    )

    OutlinedTextField(
        modifier = Modifier
            .fillMaxWidth()
            .onFocusChanged {
                if (it.isFocused) isFrontSide = true
            },
        value = bankCard.cardholderName,
        onValueChange = editBankCardCallbacks.onCardholderChange,
        label = {
            Text(text = stringResource(id =
R.string.cardholder_name_label))
        },
        singleLine = true,
        keyboardOptions = KeyboardOptions(
            capitalization = KeyboardCapitalization.Words,
            imeAction = ImeAction.Next
        ),
        keyboardActions = KeyboardActions(onNext = {
            localFocusManager.moveFocus(FocusDirection.Down)
        }),
        isError = !bankCard.cardholderName.isCardholderNameValid()
    )

```

```

        OutlinedTextField(
            modifier = Modifier
                .fillMaxWidth()
                .onFocusChanged {
                    if (it.isFocused) isFrontSide = false
                },
            value = bankCard.verificationNumber,
            onValueChange =
editBankCardCallbacks.onVerificationNumberChange,
            label = {
                Text(text = stringResource(id =
R.string.verification_number_label))
            },
            singleLine = true,
            keyboardOptions = KeyboardOptions(
                keyboardType = KeyboardType.Number,
                imeAction = ImeAction.Done
            ),
            keyboardActions = KeyboardActions(onDone = { onDone.invoke() }),
            isError =
!bankCard.verificationNumber.isVerificationNumberValid()
        )

        Spacer(modifier = Modifier.height(dp_16))
    }

    ButtonText(
        modifier = Modifier.fillMaxWidth(),
        textResId = R.string.edit_card_popup_button_save,
        onClick = onDone,
        enabled = bankCard.isValid()
    )
}
}

```

```

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups

```

```

import androidx.compose.foundation.layout.*
import androidx.compose.runtime.*

```

```

import androidx.compose.ui.Modifier
import com.google.accompanist.pager.ExperimentalPagerApi
import com.google.accompanist.pager.HorizontalPager
import com.google.accompanist.pager.rememberPagerState
import com.zhiroke.core.components.popup.PopUp
import com.zhiroke.core.components.popup.PopUpState
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.domain.models.BankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups.co
mponents.CardRecognizerPage
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups.co
mponents.EditBankCardCallbacks
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups.co
mponents.EditableCardPage
import com.zhiroke.presentation.mywallet.utils.isValid
import kotlinx.coroutines.launch

/**
 * PS: Sequence of displaying a page will be the same
 */
private enum class CardEditMode {
    Manual, Scanner
}

@OptIn(ExperimentalPagerApi::class)
@Composable
internal fun AddCardPopUp(popUpState: PopUpState, onHide: () -> Unit,
onAddClick: (BankCard) -> Unit) {

    var bankCardState by remember { mutableStateOf(BankCard.empty()) }
    val pagerState = rememberPagerState(initialPage =
CardEditMode.Manual.ordinal)
    val scope = rememberCoroutineScope()

    PopUp(popUpState = popUpState, onHide = onHide) {

        Column(
            modifier = Modifier

```

```

        .fillMaxSize()
        .padding(bottom = dp_16),
verticalArrangement = Arrangement.SpaceBetween
    ) {

HorizontalPager(
    modifier = Modifier.weight(weight = 1f),
    state = pagerState,
    count = CardEditMode.values().size,
    itemSpacing = dp_16,
    contentPadding = PaddingValues(horizontal = dp_16),
    ) { page ->

    when(page) {

        CardEditMode.Manual.ordinal -> EditableCardPage(
            modifier = Modifier.fillMaxSize(),
            bankCard = bankCardState,
            editBankCardCallbacks = EditBankCardCallbacks(
                onNumberChange = { value ->
                    bankCardState = bankCardState.copy(number =
value)

                },
                onExpirationDateChange = { value ->
                    bankCardState =
bankCardState.copy(expirationDate = value)

                },
                onCardholderChange = { value ->
                    bankCardState =
bankCardState.copy(cardholderName = value)

                },
                onVerificationNumberChange = { value ->
                    bankCardState =
bankCardState.copy(verificationNumber = value)

                },
                onPhotoUriChange = { value ->
                    bankCardState = bankCardState.copy(skinUri =
value)

                }
            ),
            onDone = {
                if (!bankCardState.isValid())

```



```

package
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups

import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.runtime.*
import androidx.compose.ui.Modifier
import com.zhiroke.core.components.popup.PopUp
import com.zhiroke.core.components.popup.PopUpState
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.domain.models.BankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups.co
mponents.EditBankCardCallbacks
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups.co
mponents.EditableCardPage
import com.zhiroke.presentation.mywallet.utils.isValid

@Composable
internal fun EditCardPopUp(popUpState: PopUpState, bankCard: BankCard, onHide:
() -> Unit, onSaveClick: (BankCard) -> Unit) {

    var bankCardState by remember(bankCard.number) { mutableStateOf(bankCard) }

    PopUp(popUpState = popUpState, onHide = onHide) {

        EditableCardPage(
            modifier = Modifier
                .fillMaxSize()
                .padding(bottom = dp_16, start = dp_16, end = dp_16),
            bankCard = bankCardState,
            editBankCardCallbacks = EditBankCardCallbacks(
                onNumberChange = { value ->
                    bankCardState = bankCardState.copy(number = value)
                },
                onExpirationDateChange = { value ->
                    bankCardState = bankCardState.copy(expirationDate = value)
                },
                onCardholderChange = { value ->
                    bankCardState = bankCardState.copy(cardholderName = value)
                }
            )
        )
    }
}

```

```

    },
    onVerificationNumberChange = { value ->
        bankCardState = bankCardState.copy(verificationNumber =
value)
    },
    onPhotoUriChange = { value ->
        bankCardState = bankCardState.copy(skinUri = value)
    }
),
onDone = {
    if (!bankCardState.isValid()) return@EditableCardPage
onHide.invoke()
onSaveClick.invoke(bankCardState)
}
)
}
}

```

```
package com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors
```

```

import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.domain.models.BankCard
import com.zhiroke.domain.repository.wallet.bankcard.BankCardRepository
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.CardCarouselEvent
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.CardCarouselState
import java.util.*

```

```

internal class AddCardInteractor(private val bankCardRepository:
BankCardRepository) :
    SuspendableInteractor<CardCarouselState, CardCarouselEvent>,
    SuspendableErrorInteractor {

    override suspend fun invoke(state: CardCarouselState, event:
CardCarouselEvent) = tryCatch(
        onFailureHandler = { e ->
            CardCarouselEvent.FailedToLoadCards(errorMessage = e.message)
        }
    )
}

```

```

    ) {
        val rawBankCard = (event as CardCarouselEvent.AddCard).bankCard

        val bankCard = BankCard(
            id = UUID.randomUUID().toString(),
            number = rawBankCard.number,
            cardholderName = rawBankCard.cardholderName.trim(),
            expirationDate = rawBankCard.expirationDate,
            verificationNumber = rawBankCard.verificationNumber,
            skinUri = rawBankCard.skinUri
        )

        bankCardRepository.addBankCard(bankCard = bankCard)
        CardCarouselEvent.AddedCardSuccessfully
    }

    override fun canHandle(event: CardCarouselEvent) = event is
CardCarouselEvent.AddCard
}

package com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors

import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.domain.repository.wallet.bankcard.BankCardRepository
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.CardCarouselEvent
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.CardCarouselState

internal class DeleteCardInteractor(private val bankCardRepository:
BankCardRepository) :
    SuspendableInteractor<CardCarouselState, CardCarouselEvent>,
    SuspendableErrorInteractor {

    override suspend fun invoke(state: CardCarouselState, event:
CardCarouselEvent) = tryCatch(
        onFailureHandler = { e ->
            CardCarouselEvent.FailedToLoadCards(errorMessage = e.message)
        }
    )
}

```

```

    ) {
        val bankCard = (event as CardCarouselEvent.DeleteCard).bankCard

        bankCardRepository.deleteBankCard(bankCard = bankCard)
        CardCarouselEvent.DeletedCardSuccessfully
    }

    override fun canHandle(event: CardCarouselEvent) = event is
CardCarouselEvent.DeleteCard
}

package com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.SubscriptionInteractor
import com.zhiroke.core.common.base.error.SubscriptionErrorInteractor
import com.zhiroke.core.components.utils.toStableList
import com.zhiroke.domain.repository.wallet.bankcard.BankCardRepository
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.CardCarouselEvent
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.CardCarouselState
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flatMapLatest
import kotlinx.coroutines.flow.flowOf

internal class LoadCardsInteractor(private val bankCardRepository:
BankCardRepository)
    : SubscriptionInteractor<CardCarouselState, CardCarouselEvent>,
SubscriptionErrorInteractor {

    @OptIn(ExperimentalCoroutinesApi::class)
    override suspend fun invoke(state: CardCarouselState, event:
CardCarouselEvent): Flow<BaseEvent> = tryCatch(
        onFailureHandler = { e ->
            CardCarouselEvent.FailedToLoadCards(errorMessage = e.message)
        }
    ) {
        bankCardRepository.fetchAll().flatMapLatest { list ->

```

```

        flowOf(CardCarouselEvent.LoadedCards(cards = list.toStableList()))
    }
}

    override fun canHandle(event: CardCarouselEvent) = event is
CardCarouselEvent.LoadCards
}

package com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors

import com.zhiroke.core.common.base.SuspendableInteractor
import com.zhiroke.core.common.base.error.SuspendableErrorInteractor
import com.zhiroke.domain.repository.wallet.bankcard.BankCardRepository
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.CardCarouselEvent
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.CardCarouselState

internal class SaveCardInteractor(private val bankCardRepository:
BankCardRepository) :
    SuspendableInteractor<CardCarouselState, CardCarouselEvent>,
    SuspendableErrorInteractor {

    override suspend fun invoke(state: CardCarouselState, event:
CardCarouselEvent) = tryCatch(
        onFailureHandler = { e ->
            CardCarouselEvent.FailedToLoadCards(errorMessage = e.message)
        }
    ) {
        val rawBankCard = (event as CardCarouselEvent.SaveCard).bankCard

        val bankCard = rawBankCard.copy(cardholderName =
rawBankCard.cardholderName.trim())

        bankCardRepository.updateBankCard(bankCard = bankCard)
        CardCarouselEvent.SavedCardSuccessfully
    }

    override fun canHandle(event: CardCarouselEvent) = event is

```

```
CardCarouselEvent.SaveCard
}
```

```
package com.zhiroke.presentation.mywallet.presentation.cardcarousel

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.error.ErrorEvent
import com.zhiroke.core.components.utils.StableList
import com.zhiroke.domain.models.BankCard

internal sealed interface CardCarouselEvent : BaseEvent {
    object LoadCards : CardCarouselEvent
    class LoadedCards(val cards: StableList<BankCard>) : CardCarouselEvent
    class FailedToLoadCards(override val errorMessage: String?) :
CardCarouselEvent, ErrorEvent
    class ChangeStateOfAddPopUp(val show: Boolean) : CardCarouselEvent
    class ChangeStateOfEditPopUp(val show: Boolean, val cardToEdit: BankCard? =
null) : CardCarouselEvent
    class AddCard(val bankCard: BankCard) : CardCarouselEvent
    object AddedCardSuccessfully : CardCarouselEvent
    class SaveCard(val bankCard: BankCard) : CardCarouselEvent
    object SavedCardSuccessfully : CardCarouselEvent
    class DeleteCard(val bankCard: BankCard) : CardCarouselEvent
    object DeletedCardSuccessfully : CardCarouselEvent
}
```

```
package com.zhiroke.presentation.mywallet.presentation.cardcarousel

import com.zhiroke.core.common.base.BaseEvent
import com.zhiroke.core.common.base.BaseReducer

internal class CardCarouselReducer : BaseReducer<CardCarouselState> {

    override val initialState: CardCarouselState get() =
CardCarouselState.initialState()

    override fun reduce(state: CardCarouselState, event: BaseEvent):
CardCarouselState {
```

```

        return state.run {
            when(event) {
                CardCarouselEvent.LoadCards -> copy(cardsState =
CardsState.Loading)
                is CardCarouselEvent.LoadedCards -> copy(cardsState = if
(event.cards.isNotEmpty()) CardsState.Loaded(event.cards) else CardsState.Empty)
                is CardCarouselEvent.FailedToLoadCards -> copy(errorMessage =
event.errorMessage)
                is CardCarouselEvent.ChangeStateOfAddPopUp -> copy(addCardPopUp
= addCardPopUp.copy(isShown = event.show))
                is CardCarouselEvent.ChangeStateOfEditPopUp ->
copy(editCardPopUp = editCardPopUp.copy(isShown = event.show), cardToEdit =
event.cardToEdit)
                is CardCarouselEvent.AddCard -> this
                CardCarouselEvent.AddedCardSuccessfully -> this
                is CardCarouselEvent.SaveCard -> this
                CardCarouselEvent.SavedCardSuccessfully -> this
                is CardCarouselEvent.DeleteCard -> this
                is CardCarouselEvent.DeletedCardSuccessfully -> this
                else -> throw IllegalArgumentException("Unsupported Event!")
            }
        }
    }
}

```

```

package com.zhiroke.presentation.mywallet.presentation.cardcarousel

import androidx.compose.foundation.layout.*
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.tooling.preview.Preview
import androidx.lifecycle.compose.ExperimentalLifecycleComposeApi
import androidx.lifecycle.compose.collectAsStateWithLifecycle
import com.google.accompanist.pager.ExperimentalPagerApi
import com.zhiroke.core.common.utils.copyToClipboardWithVibration
import com.zhiroke.core.components.pager.HorizontalPagerWithTransition
import com.zhiroke.core.theme.dimensions.dp_16
import com.zhiroke.core.theme.dimensions.dp_32
import

```

```

com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.button.AddCardButton
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.BankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.ShimmeringBankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.card.front.EmptyBankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups.AddCardPopUp
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.components.popups.EditCardPopUp
import org.koin.androidx.compose.getViewModel

@Composable
fun CardCarouselRoute() {

    CardCarouselScreen(viewModel = getViewModel())
}

@OptIn(ExperimentalLifecycleComposeApi::class)
@Composable
private fun CardCarouselScreen(viewModel: CardCarouselViewModel) {

    val state by viewModel.state.collectAsStateWithLifecycle()

    Box(modifier = Modifier.fillMaxSize()) {

        when(state.cardsState) {
            is CardsState.Loaded -> CardsAreLoaded(viewModel, state =
state.cardsState as CardsState.Loaded)
            CardsState.Loading -> CardsAreLoading()
            CardsState.Empty -> CardsAreEmpty()
        }

        AddCardButton(onClick = viewModel::showAddPopUp)
    }
}

```

```

        AddCardPopUp(
            popUpState = state.addCardPopUp,
            onHide = viewModel::hideAddPopUp,
            onAddClick = viewModel::addBankCard
        )

        state.cardToEdit?.let { bankCard ->
            EditCardPopUp(
                popUpState = state.editCardPopUp,
                bankCard = bankCard,
                onHide = viewModel::hideEditPopUp,
                onSaveClick = viewModel::saveBankCard
            )
        }
    }
}

@OptIn(ExperimentalPagerApi::class)
@Composable
private fun CardsAreLoaded(
    viewModel: CardCarouselViewModel,
    state: CardsState.Loaded
) {

    val context = LocalContext.current

    HorizontalPagerWithTransition(
        modifier = Modifier
            .fillMaxWidth()
            .wrapContentHeight(),
        count = state.cards.size
    ) { page ->

        BankCard(
            card = state.cards[page],
            onCopy = context::copyToClipboardWithVibration,
            onEditClick = viewModel::showEditPopUp,
            onDeleteClick = viewModel::deleteBankCard
        )
    }
}

```

```

@OptIn(ExperimentalPagerApi::class)
@Composable
private fun CardsAreLoading() {

    HorizontalPagerWithTransition(
        modifier = Modifier
            .fillMaxWidth()
            .wrapContentHeight(),
        count = 1
    ) {

        ShimmeringBankCard()
    }
}

@Composable
private fun CardsAreEmpty() {

    EmptyBankCard(
        modifier = Modifier
            .wrapContentSize()
            .padding(horizontal = dp_16, vertical = dp_32),
    )
}

@Preview
@Composable
private fun CardCarouselScreenPreview() {

    CardCarouselScreen(viewModel = getDummyCardCarouselViewModel())
}

package com.zhiroke.presentation.mywallet.presentation.cardcarousel

import com.zhiroke.core.common.base.BaseState
import com.zhiroke.core.common.base.error.ErrorState
import com.zhiroke.core.components.popup.PopUpState
import com.zhiroke.core.components.utils.StableList
import com.zhiroke.domain.models.BankCard

```

```

internal data class CardCarouselState(
    override val errorMessage: String?,
    val cardsState: CardsState,
    val addCardPopUp: PopUpState,
    val editCardPopUp: PopUpState,
    val cardToEdit: BankCard?
) : BaseState, ErrorState {

    companion object {

        fun initialState() = CardCarouselState(
            errorMessage = null,
            cardsState = CardsState.Loading,
            addCardPopUp = PopUpState.initialState(),
            editCardPopUp = PopUpState.initialState(),
            cardToEdit = null
        )
    }
}

internal sealed interface CardsState {
    object Loading : CardsState
    class Loaded(val cards: StableList<BankCard>) : CardsState
    object Empty : CardsState
}

package com.zhiroke.presentation.mywallet.presentation.cardcarousel

import androidx.annotation.RestrictTo
import com.zhiroke.core.common.base.BaseViewModel
import com.zhiroke.core.common.coroutines.AppDispatchersProvider
import com.zhiroke.core.common.coroutines.DispatchersProvider
import com.zhiroke.domain.models.BankCard
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors.AddCardI
nteractor
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors.DeleteCa
rdInteractor

```

```

import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors.LoadCard
sInteractor
import
com.zhiroke.presentation.mywallet.presentation.cardcarousel.interactors.SaveCard
Interactor

internal class CardCarouselViewModel(
    reducer: CardCarouselReducer,
    dispatchers: DispatchersProvider,
    loadCardsInteractor: LoadCardsInteractor?,
    addCardInteractor: AddCardInteractor?,
    saveCardInteractor: SaveCardInteractor?,
    deleteCardInteractor: DeleteCardInteractor?,
) : BaseViewModel<CardCarouselState, CardCarouselEvent>(
    dispatchers = dispatchers,
    reducer = reducer,
    interactors = setOfNotNull(loadCardsInteractor, addCardInteractor,
saveCardInteractor, deleteCardInteractor),
    initialEvent = CardCarouselEvent.LoadCards
) {

    fun showAddPopUp() = sendEvent(event =
CardCarouselEvent.ChangeStateOfAddPopUp(show = true))

    fun hideAddPopUp() = sendEvent(event =
CardCarouselEvent.ChangeStateOfAddPopUp(show = false))

    fun showEditPopUp(bankCard: BankCard) = sendEvent(event =
CardCarouselEvent.ChangeStateOfEditPopUp(show = true, cardToEdit = bankCard))

    fun hideEditPopUp() = sendEvent(event =
CardCarouselEvent.ChangeStateOfEditPopUp(show = false))

    fun addBankCard(bankCard: BankCard) = sendEvent(event =
CardCarouselEvent.AddCard(bankCard = bankCard))

    fun saveBankCard(bankCard: BankCard) = sendEvent(event =
CardCarouselEvent.SaveCard(bankCard = bankCard))

    fun deleteBankCard(bankCard: BankCard) = sendEvent(event =

```

```

CardCarouselEvent.DeleteCard(bankCard = bankCard))
}

/** This function is used only for compose preview. */
@RestrictTo(RestrictTo.Scope.TESTS)
internal fun getDummyCardCarouselViewModel(): CardCarouselViewModel {
    return CardCarouselViewModel(
        dispatchers = AppDispatchersProvider(),
        reducer = CardCarouselReducer(),
        loadCardsInteractor = null,
        addCardInteractor = null,
        saveCardInteractor = null,
        deleteCardInteractor = null
    )
}

package com.zhiroke.presentation.mywallet.utils

infix fun Float.roundUpToMultipleOf(multiplier: Float): Float {
    val diff = this % multiplier
    return if (multiplier / 2 > diff) {
        this - diff
    } else {
        this + multiplier - diff
    }
}

fun Float.reverseSignIf(predicate: Boolean): Float = if (predicate) this * -1
else this

package com.zhiroke.presentation.mywallet.utils

import com.zhiroke.domain.models.BankCard

internal fun String.isNumberValid(): Boolean {
    return length == 16 && all { it.isDigit() }
}

```

```
internal fun String.isExpirationDateValid(): Boolean {
    return length == 4 && all { it.isDigit() } && take(2).toInt() in (1..12)
}
```

```
internal fun String.isCardholderNameValid(): Boolean {
    return all { it.isLetter() || it.isWhitespace() } && isNotBlank()
}
```

```
internal fun String.isVerificationNumberValid(): Boolean {
    return length in (3..4) && all { it.isDigit() }
}
```

```
internal fun BankCard.isValid(): Boolean {
    return number.isNumberValid() &&
        expirationDate.isExpirationDateValid() &&
        cardholderName.isCardholderNameValid() &&
        verificationNumber.isVerificationNumberValid()
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="add_card_popup_button_add">Add</string>
    <string name="edit_card_popup_button_save">Save</string>
    <string name="card_number_label">Card number</string>
    <string name="expiration_date_label">Expiration date</string>
    <string name="cardholder_name_label">Cardholder name</string>
    <string name="verification_number_label">Verification number</string>
    <string name="empty_card_text">It looks like, you haven't added any cards
yet.</string>

</resources>
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="add_card_popup_button_add">Додати</string>
    <string name="edit_card_popup_button_save">Зберегти</string>
    <string name="card_number_label">Номер картки</string>
    <string name="expiration_date_label">Термін придатності</string>
    <string name="cardholder_name_label">Ім'я власника картки</string>
    <string name="verification_number_label">Номер підтвердження</string>
```

```

        <string name="empty_card_text">Здається, ви ще не додали карти.</string>

</resources>

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="dont_remember_pin">Don\'t remember your PIN?</string>
    <string name="enter_pin_code">Enter PIN code</string>
    <string name="biometric_title">Scan to log in</string>
    <string name="biometric_negative_btn">Enter PIN code</string>
    <string name="set_pin_code">Set PIN code</string>
    <string name="confirm_pin_code">Confirm PIN code</string>
    <string name="btn_next">Next</string>
    <string name="btn_confirm">Confirm</string>
    <string name="btn_back">Back</string>
    <string name="error_to_many_attempts">Too many attempts. Please try again
later.</string>
    <string name="error_pins_dont_match">Entered PIN codes don\'t match, please
try again.</string>
    <string name="placeholder_try_again">Try again in %ds.</string>
    <string name="placeholder_timer">" (%ds)"</string>
    <string name="drop_data_warning">If you want to reset the PIN code, all your
data in the application will be erased. Are you sure about this, there will be
no possibility to restore the data?</string>
    <string name="button_drop_data_cancel">No, cancel.</string>
    <string name="button_drop_data_confirm">Yes, erase all app\'s data</string>
</resources>

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="dont_remember_pin">Не пом\'ятаєте ваш пін?</string>
    <string name="enter_pin_code">Введіть ПІН-код</string>
    <string name="biometric_title">Відскануйте, щоб увійти</string>
    <string name="biometric_negative_btn">Ввести ПІН-код</string>
    <string name="set_pin_code">Задати ПІН-код</string>
    <string name="confirm_pin_code">Підтвердіть ПІН-код</string>
    <string name="btn_next">Далі</string>
    <string name="btn_confirm">Підтвердити</string>
    <string name="btn_back">Назад</string>
    <string name="error_to_many_attempts">Забаргато спроб. Зачекайте щоб

```

повторити.</string>

<string name="error_pins_dont_match">Введенні ПІН-коди відрізняються, будь ласка спробуйте ще раз.</string>

<string name="placeholder_try_again">Повторіть спробу через %dc.</string>

<string name="placeholder_timer">" (%dc)"</string>

<string name="drop_data_warning">Якщо ви хочите скинути ПІН-код, тоді усі ваші дані у додатку будуть видалені. Ви впевнені в цьому, можливості відновити дані не буде?</string>

<string name="button_drop_data_cancel">Ні, не треба.</string>

<string name="button_drop_data_confirm">Так, видали всі дані додатку</string>

</resources>

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="placeholder_timer">" (%ds)"</string>

</resources>

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="placeholder_timer">" (%dc)"</string>

</resources>

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="rational_permission_request_title">Permission request</string>

<string name="rational_permission_request_confirm_button">OK</string>

<string name="rational_permission_request_rational_text">Camera permission required for the bank card recognition feature to be available.</string>

</resources>

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="rational_permission_request_title">Запит на дозвіл</string>

<string name="rational_permission_request_confirm_button">Добре</string>

<string name="rational_permission_request_rational_text">Для роботи функції розпізнавання банківської картки потрібен дозвіл на камеру.</string>

</resources>

```

plugins {
    id("com.android.application") version "7.3.1" apply false
    id("com.android.library") version "7.3.1" apply false
    kotlin("android") version "1.7.20" apply false
    id("org.jetbrains.kotlin.jvm") version "1.7.20" apply false
}

import java.io.FileInputStream
import java.util.Properties

plugins {
    id("com.android.application")
    kotlin("android")
}

val signingProps = Properties()
val signingPropertiesFile = rootProject.file("app/signing-configs.properties")

if (signingPropertiesFile.exists()) {
    signingProps.load(FileInputStream(signingPropertiesFile))
}

android {
    namespace = Configs.packageName
    compileSdk = Configs.compileSdk

    defaultConfig {
        applicationId = Configs.packageName
        minSdk = Configs.minSdk
        targetSdk = Configs.targetSdk
        versionCode = Configs.versionCode
        versionName = Configs.versionName

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables {
            useSupportLibrary = true
        }
    }

    buildTypes {

```

```

    getByName("release") {
        isDebuggable = false
        isMinifyEnabled = true
        isShrinkResources = true
        proguardFiles(getDefaultProguardFile("proguard-android-
optimize.txt"), "proguard-rules.pro")
    }
}

signingConfigs {
    register("prod") {
        storeFile =
signingProps["prodSigningFileName"]?.let(rootProject::file)
        storePassword = "${signingProps["prodKeystorePassword"]}"
        keyAlias = "${signingProps["prodAlias"]}"
        keyPassword = "${signingProps["prodAliasPassword"]}"
    }
    register("dev") {
        storeFile =
signingProps["devSigningFileName"]?.let(rootProject::file)
        storePassword = "${signingProps["devKeystorePassword"]}"
        keyAlias = "${signingProps["devAlias"]}"
        keyPassword = "${signingProps["devAliasPassword"]}"
    }
}

flavorDimensions("default")
productFlavors {
    register("prod") {
        dimension = "default"
        resValue("string", "app_name", "JetWallet")
        signingConfig = signingConfigs.getByName("prod")
    }
    register("dev") {
        dimension = "default"
        applicationIdSuffix = ".dev"
        versionNameSuffix = "-dev"
        resValue("string", "app_name", "JetWallet Dev")
        signingConfig = signingConfigs.getByName("dev")
    }
}

```

```

compileOptions {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}
kotlinOptions {
    jvmTarget = "11"
}
buildFeatures {
    compose = true
}
composeOptions {
    kotlinCompilerExtensionVersion = Versions.composeCompiler
}
packagingOptions {
    resources {
        excludes += "/META-INF/{AL2.0,LGPL2.1}"
    }
}
}

dependencies {
    implementation(project(":core:navigation"))
    implementation(project(":core:common"))
    implementation(project(":core:theme"))
    implementation(project(":data"))
    implementation(project(":domain"))
    implementation(project(":presentation:mywallet"))
    implementation(project(":presentation:auth"))

    implementation(Libs.AndroidX.coreKtx)
    implementation(Libs.AndroidX.activityCompose)
    implementation(Libs.Lifecycle.runtimeKtx)
    implementation(Libs.Compose.ui)
    implementation(Libs.Compose.material3)
    implementation(Libs.Navigation.navigationCompose)
    implementation(Libs.Koin.core)
    implementation(Libs.Koin.android)
}

plugins {
    `kotlin-dsl`

```

```

}

repositories {
    google()
    mavenCentral()
}

dependencies {
    compileOnly("com.android.tools.build:gradle:7.0.0")
    gradleApi()
}

plugins {
    id("com.android.library")
    kotlin("android")
}

android {
    namespace = "com.zhiroke.core.cardrecognition"
    compileSdk = Configs.compileSdk

    defaultConfig {
        minSdk = Configs.minSdk

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles("consumer-rules.pro")
    }

    buildTypes {
        release {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }

    flavorDimensions("default")
    productFlavors {
        register("prod")
    }
}

```

```

        register("dev")
    }

    buildFeatures {
        compose = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = Versions.composeCompiler
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }

    kotlinOptions {
        jvmTarget = "11"
    }
}

dependencies {

    implementation(project(":core:common"))

    implementation(Libs.AndroidX.coreKtx)
    implementation(Libs.Compose.ui)
    implementation(Libs.Compose.toolingPreview)
    implementation(Libs.Compose.material3)

    implementation(Libs.Coroutines.core)

    implementation(Libs.Gms.textRecognizerMlKit)

    implementation(Libs.CameraX.camera2)
    implementation(Libs.CameraX.view)
    implementation(Libs.CameraX.lifecycle)
    implementation(Libs.CameraX.mlKit)
}

plugins {
    id("com.android.library")
    kotlin("android")
}

```

```

android {
    namespace = "com.zhiroke.core.common"
    compileSdk = Configs.compileSdk

    defaultConfig {
        minSdk = Configs.minSdk

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles("consumer-rules.pro")
    }

    buildTypes {
        getByName("release") {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }

    flavorDimensions("default")
    productFlavors {
        register("prod")
        register("dev")
    }

    buildFeatures {
        compose = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = Versions.composeCompiler
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }

    kotlinOptions {
        jvmTarget = "11"
    }
}

```

```

dependencies {

    implementation(Libs.AndroidX.coreKtx)
    implementation(Libs.AndroidX.activityCompose)
    implementation(Libs.Lifecycle.runtimeKtx)
    implementation(Libs.Coroutines.core)

    implementation(Libs.Permissions.accompanist)

    implementation(Libs.Koin.core)
    implementation(Libs.Koin.android)
}

plugins {
    id("com.android.library")
    kotlin("android")
}

android {
    namespace = "com.zhiroke.core.components"
    compileSdk = Configs.compileSdk

    defaultConfig {
        minSdk = Configs.minSdk

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles("consumer-rules.pro")
    }

    buildTypes {
        getByName("release") {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
}

```

```

flavorDimensions("default")
productFlavors {
    register("prod")
    register("dev")
}

buildFeatures {
    compose = true
}

composeOptions {
    kotlinCompilerExtensionVersion = Versions.composeCompiler
}

compileOptions {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}

kotlinOptions {
    jvmTarget = "11"
}
}

dependencies {
    implementation(project(":core:common"))
    implementation(project(":core:theme"))

    implementation(Libs.AndroidX.coreKtx)
    implementation(Libs.AndroidX.activityCompose)
    implementation(Libs.Lifecycle.runtimeKtx)
    implementation(Libs.Compose.ui)
    implementation(Libs.Compose.uiUtils)
    implementation(Libs.Compose.toolingPreview)
    implementation(Libs.Compose.material3)
    implementation(Libs.Compose.accompanistPager)
    implementation(Libs.Compose.accompanistPagerIndicator)

    testImplementation(Libs.Test.jUnit)
    androidTestImplementation(Libs.Test.androidJUnit)
    androidTestImplementation(Libs.Test.espresso)
    androidTestImplementation(Libs.Compose.testJUnit4)
}

```

```

plugins {
    id("com.android.library")
    kotlin("android")
}
//apply<plugins.KotlinAndroidConfigurationPlugin>() //TODO: Finalize

android {
    namespace = "com.zhiroke.core.navigation"
    compileSdk = Configs.compileSdk

    defaultConfig {
        minSdk = Configs.minSdk

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles("consumer-rules.pro")
    }

    buildTypes {
        getByName("release") {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }

    flavorDimensions("default")
    productFlavors {
        register("prod")
        register("dev")
    }

    buildFeatures {
        compose = true
    }
    composeOptions {
        kotlinCompilerExtensionVersion = Versions.composeCompiler
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }

```

```

    }
    kotlinOptions {
        jvmTarget = "11"
    }
}

dependencies {

    implementation(Libs.AndroidX.coreKtx)
    implementation(Libs.Koin.core)
    implementation(Libs.Navigation.navigationCompose)
}

plugins {
    id("com.android.library")
    kotlin("android")
}

android {
    namespace = "com.zhiroke.core.theme"
    compileSdk = Configs.compileSdk

    defaultConfig {
        minSdk = Configs.minSdk

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles("consumer-rules.pro")
    }

    buildTypes {
        getByName("release") {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }

    flavorDimensions("default")
    productFlavors {

```

```

        register("prod")
        register("dev")
    }

    buildFeatures {
        compose = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = Versions.composeCompiler
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }

    kotlinOptions {
        jvmTarget = "11"
    }
}

dependencies {

    implementation(Libs.AndroidX.coreKtx)
    implementation(Libs.AndroidX.appCompat)
    implementation(Libs.Compose.ui)
    implementation(Libs.Compose.material3)
}

plugins {
    id("com.android.library")
    kotlin("android")
    kotlin("kapt")
    id("kotlin-parcelize")
}

android {
    namespace = "com.zhiroke.data"
    compileSdk = Configs.compileSdk

    defaultConfig {
        minSdk = Configs.minSdk
    }
}

```

```

testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
consumerProguardFiles("consumer-rules.pro")

kapt {
    arguments {
        arg("room.schemaLocation", "$projectDir/schemas")
    }
}

buildTypes {
    getByName("release") {
        isMinifyEnabled = true
        proguardFiles(
            getDefaultProguardFile("proguard-android-optimize.txt"),
            "proguard-rules.pro"
        )
    }
}

flavorDimensions("default")
productFlavors {
    register("prod")
    register("dev")
}

compileOptions {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}
kotlinOptions {
    jvmTarget = "11"
}

dependencies {

    implementation(Libs.AndroidX.coreKtx)

    implementation(Libs.Room.runtime)
    implementation(Libs.Room.ktx)
    kapt(Libs.Room.compiler)
}

```

```

implementation(Libs.SqlCipher.cipher)
implementation(Libs.Security.crypto)

implementation(Libs.Koin.core)
implementation(Libs.Koin.android)

androidTestImplementation(Libs.Test.jUnit)
androidTestImplementation(Libs.Test.androidJUnit)
androidTestImplementation(Libs.Test.espresso)
androidTestImplementation(Libs.Test.googleTruth)
}

plugins {
    id("com.android.library")
    kotlin("android")
}

android {
    namespace = "com.zhiroke.domain"
    compileSdk = Configs.compileSdk

    defaultConfig {
        minSdk = Configs.minSdk

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles("consumer-rules.pro")
    }

    buildTypes {
        getByName("release") {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
}

flavorDimensions("default")
productFlavors {
    register("prod")
}

```

```

        register("dev")
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }
    kotlinOptions {
        jvmTarget = "11"
    }
}

dependencies {
    implementation(project(":core:common"))
    implementation(project(":data"))

    implementation(Libs.AndroidX.coreKtx)
    implementation(Libs.Lifecycle.runtimeKtx)
    implementation(Libs.Compose.runtime) // need to make ui models stable

    implementation(Libs.Koin.core)
    implementation(Libs.Koin.android)
}

plugins {
    id("com.android.library")
    kotlin("android")
}

android {
    namespace = "com.zhiroke.presentation.auth"
    compileSdk = Configs.compileSdk

    defaultConfig {
        minSdk = Configs.minSdk
        targetSdk = Configs.targetSdk

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles("consumer-rules.pro")
    }
}

```

```

buildTypes {
    release {
        isMinifyEnabled = true
        proguardFiles(
            getDefaultProguardFile("proguard-android-optimize.txt"),
            "proguard-rules.pro"
        )
    }
}

flavorDimensions("default")
productFlavors {
    register("prod")
    register("dev")
}

buildFeatures {
    compose = true
}

composeOptions {
    kotlinCompilerExtensionVersion = Versions.composeCompiler
}

compileOptions {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}

kotlinOptions {
    jvmTarget = "11"
}
}

dependencies {

    implementation(project(":core:common"))
    implementation(project(":core:components"))
    implementation(project(":core:navigation"))
    implementation(project(":core:theme"))
    implementation(project(":domain"))

    implementation(Libs.AndroidX.coreKtx)
    implementation(Libs.AndroidX.biometric)

```

```

implementation(Libs.Compose.ui)
implementation(Libs.Compose.toolingPreview)
implementation(Libs.Compose.material3)
implementation(Libs.Lifecycle.runtimeCompose)
implementation(Libs.Lifecycle.viewModelCompose)

implementation(Libs.Coroutines.core)

implementation(Libs.Koin.android)
implementation(Libs.Koin.compose)
implementation(Libs.Koin.core)
}

plugins {
    id("com.android.library")
    kotlin("android")
}

android {
    namespace = "com.zhiroke.presentation.mywallet"
    compileSdk = Configs.compileSdk

    defaultConfig {
        minSdk = Configs.minSdk

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles("consumer-rules.pro")
    }

    buildTypes {
        getByName("release") {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
}

flavorDimensions("default")
productFlavors {

```

```

        register("prod")
        register("dev")
    }

    buildFeatures {
        compose = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = Versions.composeCompiler
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }

    kotlinOptions {
        jvmTarget = "11"
        freeCompilerArgs = freeCompilerArgs + "-P"
        freeCompilerArgs = freeCompilerArgs +
("plugin:androidx.compose.compiler.plugins.kotlin:reportsDestination=" +
project.buildDir.absolutePath + "/compose_metrics")
        freeCompilerArgs = freeCompilerArgs + "-P"
        freeCompilerArgs = freeCompilerArgs +
("plugin:androidx.compose.compiler.plugins.kotlin:metricsDestination=" +
project.buildDir.absolutePath + "/compose_metrics")
    }
}

dependencies {

    implementation(project(":domain"))
    implementation(project(":core:common"))
    implementation(project(":core:components"))
    implementation(project(":core:navigation"))
    implementation(project(":core:theme"))
    implementation(project(":core:cardrecognition"))

    implementation(Libs.AndroidX.coreKtx)
    implementation(Libs.Lifecycle.runtimeKtx)
    implementation(Libs.Lifecycle.runtimeCompose)

    implementation(Libs.Compose.ui)
    implementation(Libs.Compose.toolingPreview)

```

```

        implementation(Libs.Compose.material3)
        implementation(Libs.Compose.accompanistPager)
        implementation(Libs.Compose.materialIconsExtended)
        implementation(Libs.Compose.materialIconsExtended)

        implementation(Libs.Coil.compose)

        implementation(Libs.Koin.core)
        implementation(Libs.Koin.android)
        implementation(Libs.Koin.compose)
    }

    pluginManagement {
        repositories {
            google()
            mavenCentral()
            gradlePluginPortal()
        }
    }

    dependencyResolutionManagement {
        repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
        repositories {
            google()
            mavenCentral()
        }
    }

    rootProject.name = "JetWallet"
    include(":app")
    include(":core:common")
    include(":core:components")
    include(":core:navigation")
    include(":core:theme")
    include(":core:cardrecognition")
    include(":domain")
    include(":data")
    include(":presentation:mywallet")
    include(":presentation:auth")

```

ДОДАТОК Б
Слайди презентації

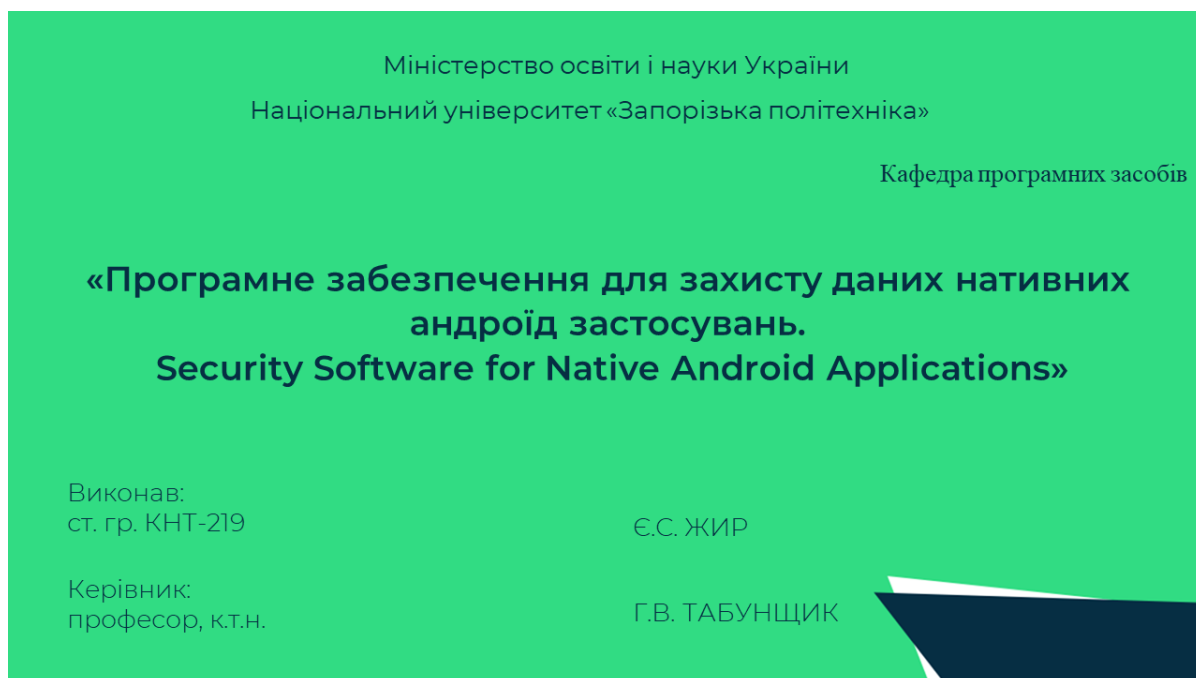


Рисунок Б.1 – Слайд 1

Актуальність, мета роботи, об'єкт та предмет дослідження

- Мета роботи полягає у підвищенні захищеності даних користувача в андроїд додатку.
- Об'єкт дослідження – безпека даних.
- Предмет дослідження – програмне забезпечення для захисту даних нативних андроїд застосувань.

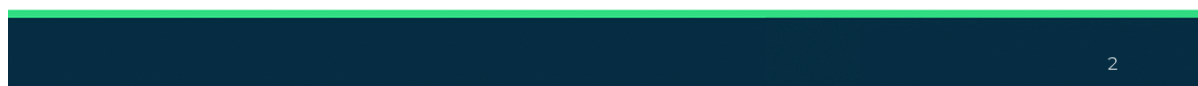


Рисунок Б.2 – Слайд 2

Завдання дипломної роботи

- Дослідити якими способами можна здобути дані додатка;
- Вивчити як протидіяти спробам здобути дані додатка;
- Розробити нативний Android додаток який спроможній захистити внутрішні дані;
- Оформити пояснювальну записку.

3

Рисунок Б.3 – Слайд 3

Аналіз існуючих додатків

Таблиця 1 – Порівняльна таблиця додатків-аналогів

Критерії оцінювання	Назва додатка	
	SecureCard	Google Wallet
Розблокування PIN кодом	+	-
Розблокування PIN скануванням відбитка пальця	+	-
Обфускація	+	+
Захист від запису екрану	+	+
Шифрування	+	+
Ручне додавання карток	+	+
Додавання карток скануванням	-	+
NFC розрахунок;	-	+

4

Рисунок Б.4 – Слайд 4

Функціонал розробленого додатка

- Додавання банківської картки вручну.
- Додавання банківської картки скануванням .
- Редагування картки.
- Видалення картки.
- Налаштування зовнішнього вигляду картки.
- Встановлення PIN коду.
- Розблокування додатка за допомогою відбитка пальця.
- Скидання PIN коду.
- Блокування можливість розблокування на 30 секунд.
- Блокувати можливість розблокування за допомогою відбитка пальця на 30 секунд.
- Валідація картки.
- Копіювання номера картки.
- Шифрування даних, що зберігаються.
- Обфускація APK додатка.
- Заборона перегляду внутрішньої пам'яті додатка.

5

Рисунок Б.5 – Слайд 5

Вибір програмних засобів

Таблиця 2 – Програмні засоби

Галузь	Програмний засіб
Інтегроване середовище розробки (IDE)	Android Studio
Мова програмування	Kotlin
Патерни проектування	MVI
Сховища даних	Room (SQLite), EncryptedSharedPreferences
Інструмент для створення інтерфейсу	Jetpack Compose
Впровадження залежності (DI)	Koin
Шифрування даних	SqlCipher, Java Cipher
Обфускація	R8

6

Рисунок Б.6 – Слайд 6

Вибір програмних засобів

Таблиця 3 – Порівняння мов програмування

Критерії	Мова програм	
	Kotlin	Java
Типізованість	неявне оголошення типів	явне оголошення типів
Безпека від NPE	+	-
ООП	+	+
Функціональне програмування	+	+
Ефективність	+-	+-
Лаконічність	Має перевагу	Поступається
Швидкість написання коду	Має перевагу	Поступається
Кількість помилок	Менша	Більша
Функціональність	Більша	Менша
Складність навчання	+-	+-

7

Рисунок Б.7 – Слайд 7

Вибір програмних засобів

Таблиця 4 – Порівняння симетричних алгоритмів

Критерії	Назва алгоритму		
	DES	AES	Blowfish
Розмір блока (bits)	64	128	64
Розмір ключа (bits)	56	128, 192, 256	32-448
Безпека	Слабий рівень захисту	Сильніший рівень захисту ніж у Blowfish	Сильний рівень захисту
Швидкість	Повільний	Швидкий	Швидше ніж AES
Стандартизація	Вважається застарілим	Добре тестований та стандартизований	Менш популярний ніж AES

8

Рисунок Б.8 – Слайд 8

Вибір програмних засобів

Таблиця 5 – Порівняння асиметричних алгоритмів

Критерії	Назва алгоритму	
	RSA	Діффі-Геллмана
Розмір ключа (bits)	1024-4096	1024-4096
Генерація ключа	На основі складності розкладання чисел	На основі складності обчислення дискретних логарифмів
Безпека	Сильний рівень захисту	Сильний рівень захисту
Швидкість	Повільний	Швидший ніж RSA
Стандартизація	Використовується для цифрових підписів	Використовується переважно для обміну ключами

9

Рисунок Б.9 – Слайд 9

Огляд екрану «Блокування»:

- розблокування введенням PIN (рис. 1);
- розблокування скануванням (рис. 2);
- діалогове вікно попередження;
- створення PIN;
- підтвердження PIN;
- заблокований екран блокування (рис. 3).

Огляд інтерфейсу

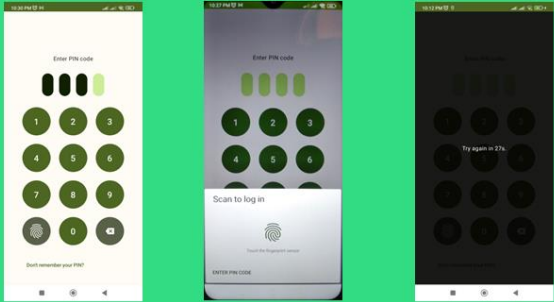


Рисунок 1 – Розблокування PIN кодом

Рисунок 2 – Розблокування скануванням

Рисунок 3 – Заблокований екран розблокування

10

Рисунок Б.10 – Слайд 10

Огляд екрану «Каруселька карток»:

- каруселька (рис. 4);
- додавання/редагування карток (рис. 5);
- видалення картки;
- додавання картинки для картки
- пояснення на запит дозволу;
- сканування картки (рис. 6).

Огляд інтерфейсу



Рисунок 4 –
Каруселька



Рисунок 5 –
Додавання/редагування карток



Рисунок 6 –
Сканування картки

11

Рисунок Б.11 – Слайд 11

Захист даних. Зберігання

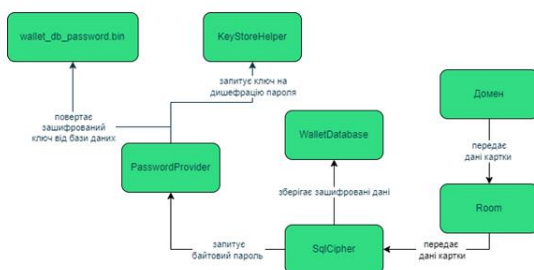


Рисунок 7 – Зображення логіки збереження картки в таблицю

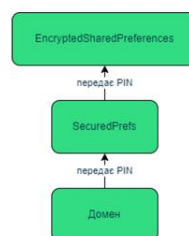


Рисунок 8 – Зображення логіки збереження PIN коду в EncryptedSharedPreferences

12

Рисунок Б.12 – Слайд 12

Захист даних. UI Захист

- Екран Блокування (рис. 1 - 2).
- Заблокувати можливість розблокування додатка (рис. 3).
- Блокування запису екрану (рис. 9).
- Блокування знімків екрану (рис. 10).



Рисунок 9 – Блокування запису екрану

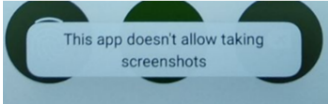


Рисунок 10 – Блокування знімків екрану

Рисунок Б.13 – Слайд 13

Захист даних. Захист APK

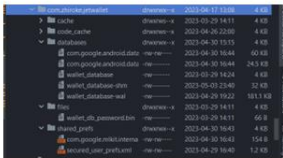


Рисунок 11 – Внутрішні дані додатка без захисту

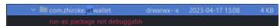


Рисунок 12 – Внутрішні дані додатка з захистом

Type: APK File
Size: 24.9 MB
Рисунок 13 – Розмір APK без R8

Type: APK File
Size: 15.4 MB
Рисунок 14 – Розмір APK з R8

```
/* access modifiers changed from: private */  
public static final BankCardDao provideBankCardDao(WalletDatabase walletDatabase) {  
    return walletDatabase.getBankCardDao();  
}
```

Рисунок 15 – Код APK без R8

```
/* JADX INFO: super call moved to the top of the method (can break code semantics)  
public C0003d(C0000a aVar, C0236e eVar, String str, C1015a aVar2, C110311 j1Var) {  
    super();  
    this.f4o = aVar;  
    this.f5p = eVar;  
    this.f6q = str;  
    this.f7r = aVar2;  
    this.f8s = j1Var;  
}
```

Рисунок 16 – Код APK з R8

Рисунок Б.14 – Слайд 14

Висновки

Разом із бакалаврською дипломною роботою, було розроблено нативний Android додаток JetWallet для безпечного зберігання карток.

Перед початком розробки, було досліджено вимоги до застосування і проведено аналіз існуючих аналогів. Також було порівнянні існуючі бібліотеки, архітектурні патерни, середовища розробки та мови програмування.

Розробка велась в IDE Android Studio. При виконанні розробки, були дотримані принципи Чистої Архітектури та було реалізовано патерн MVI. Завдяки цим рішенням, було досягнуто стабільної, з меншою кількістю багів і більш підтримуваної системи.

Стосовно безпеки даних, були використані такі методи:

- шифрування даних;
- обфускація APK;
- обмеження доступу до внутрішніх файлів додатка;
- екран блокування;
- блокування запису екрана.

15

Рисунок Б.15 – Слайд 15

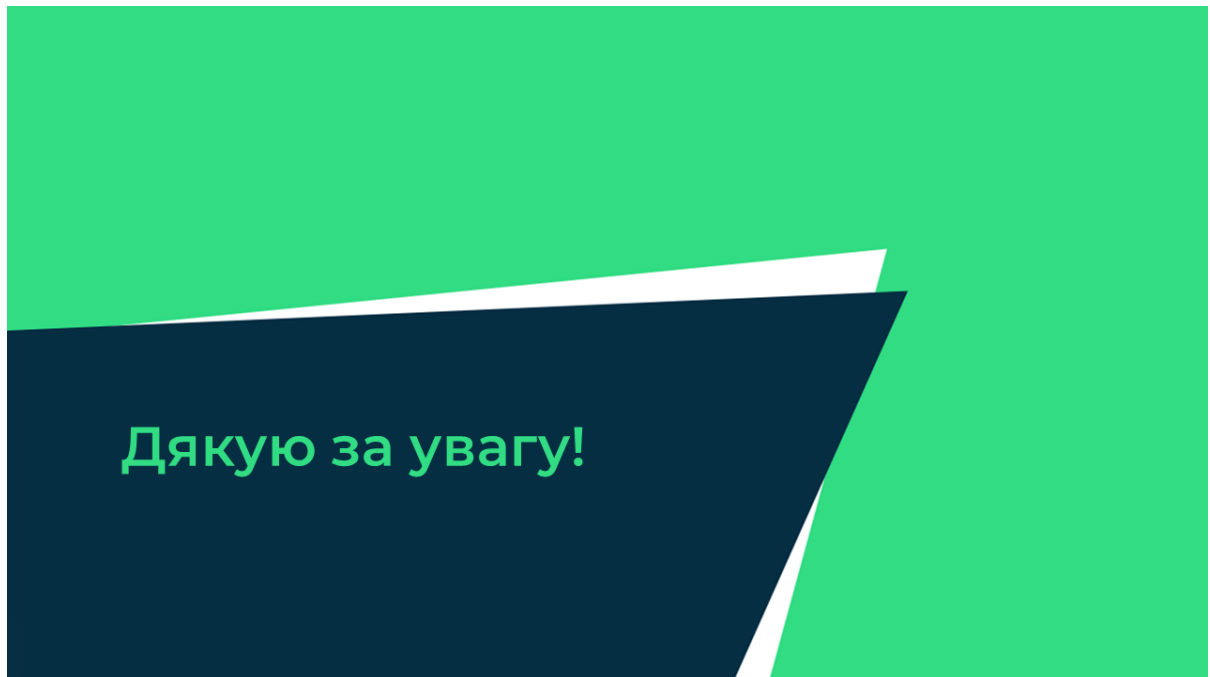


Рисунок Б.16 – Слайд 16